УДК 519.711.3

ВЕРИФИКАЦИЯ СОБЫТИЙНЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ

© 2012 Е. А. Бабкин[1], В. В. Разиньков[2]

[1]канд. техн. наук, доцент, профессор каф. программного обеспечения и администрирования информационных систем
e-mail: eababkin@gmail.com
[2]аспирант каф. программного обеспечения и администрирования информационных систем
e-mail: razinkov86@gmail.com

Курский государственный университет

Рассматривается верификация имитационных моделей дискретных систем, представляемых в виде событийных графов. Определяются требования-ограничения синтаксической и семантической правильности: общие ограничения, структурные ограничения исходного событийного графа, структурные ограничения событийного графа в программно-реализуемой форме, семантические ограничения событийной имитационной модели и локальные ограничения.

Ключевые слова: верификация, имитационное моделирование, структурные ограничения, семантические ограничения, событийный граф.

Рост сложности задач, решаемых методами имитационного моделирования, требует разработки более эффективных способов взаимодействия человека с компьютером в процессе построения имитационной модели.

При создании событийной имитационной модели формулируется концептуальная модель системы, преобразуемая затем в событийный граф (СГ). Полученный исходный событийный граф в дальнейшем преобразуется в программно-реализуемую форму (макрособытийный граф), которая затем транслируется в программную модель [Бабкин 2005; 2006]. На всех этапах построения модели могут возникать ошибки, которые устраняются путем валидации, верификации и тестирования [Balci 1998].

Верификация (verification) – это процесс подтверждения соответствия построенной модели её спецификации, где степень соответствия определяется с достаточной для разработчика точностью [Balci 1998; Пепеляев 2003]. Верификация выполняется на протяжении всего жизненного цикла модели: начиная от формулировки проблемы до получения исполняемого компьютерного приложения.

Ошибки на этапе формулировки концептуальной модели носят принципиальный характер и могут сделать напрасной всю последующую работу. Однако не существует строгих методов их обнаружения, поскольку, вообще говоря, формулировка задачи – в большой мере искусство и зависит от субъективных взглядов разработчика модели и заказчика.

Верификация СГ производится на основе документированной концептуальной модели, являющейся спецификацией СГ.

Верификация программной модели производится на основе СГ, который, в свою очередь, является спецификацией программной модели.

При построении исходного СГ по содержательному описанию системы (концептуальной модели) возникает задача анализа синтаксической (структурной) и семантической правильности событийного графа [Замятина 2011].

Верификация СГ производится на основе набора ***общих требований-ограничений***, исходящих из природы и свойств графа, справедливых для всех СГ, и документированной концептуальной модели, являющейся спецификацией СГ. Спецификация СГ включает набор ***локальных требований-ограничений***, справедливых только для рассматриваемой моделируемой системы, и описание всех функциональных требований к системе. Ограничения описывают допустимые (или недопустимые) события, состояния и последовательности событий и состояний системы, допустимые структуры модели. Ограничения могут быть использованы также в качестве основы описания синтаксиса и семантики языка моделирования событийных имитационных моделей.

Ошибки разработки событийной имитационной модели выявляются поэтапно в процессе верификации при построении исходного СГ, преобразовании его в программно-реализуемую форму и при исполнении программной модели (в процессе тестирования).

На основе описанных ранее [Бабкин 2005] принципов построения СГ, использующихся при разработке имитационных моделей, были выработаны следующие требования-ограничения синтаксической и семантической правильности событийных имитационных моделей дискретных систем.

Общие ограничения

Общие ограничения событийной имитационной модели можно разделить на ограничения СГ и ограничения программной модели.

Процесс верификации СГ заключается в проверке выполнения множества ограничивающих условий. Эта проверка позволяет избежать ошибок разработки событийной модели как на начальном этапе её построения, так и в завершающей стадии проектирования, перед преобразованием в программную модель. Все ограничения, накладываемые на СГ, можно разделить на три основные группы:

1) ограничения отдельных элементов событийного графа;

2) ограничения исходного событийного графа, представляющего процесс функционирования системы в терминах событий;

3) ограничения макрособытийного графа, представляющего программно-реализуемую форму событийного графа.

Ограничения могут быть заданы перечислением в той или иной форме разрешенных либо запрещенных ситуаций.

Для формального описания ограничений разработана метамодель графического языка построения событийных графов (рис. 1). Метамоделирование позволяет описать абстрактный синтаксис языка с помощью диаграмм классов UML, глубже формализовать структуру языка и описывать более сложные отношения между его конструкциями [Кознов 2007].

На рисунке 1 представлена метамодель языка событийных графов в виде диаграммы классов UML. Каждый класс на этой диаграмме представляет определенный графический элемент событийного графа:

- *Event_vertex* – событийная вершина (вершин первого типа);
- *Condition* – логическая вершина P_i (вершина второго типа);
- *Parallel* – вершина начала параллельных участков (вершина третьего типа);
- *End_parallel* – вершина окончания параллельных участков (вершина четвертого типа);
- *Vertex* – вершина любого типа событийного графа (базовый класс вершины);

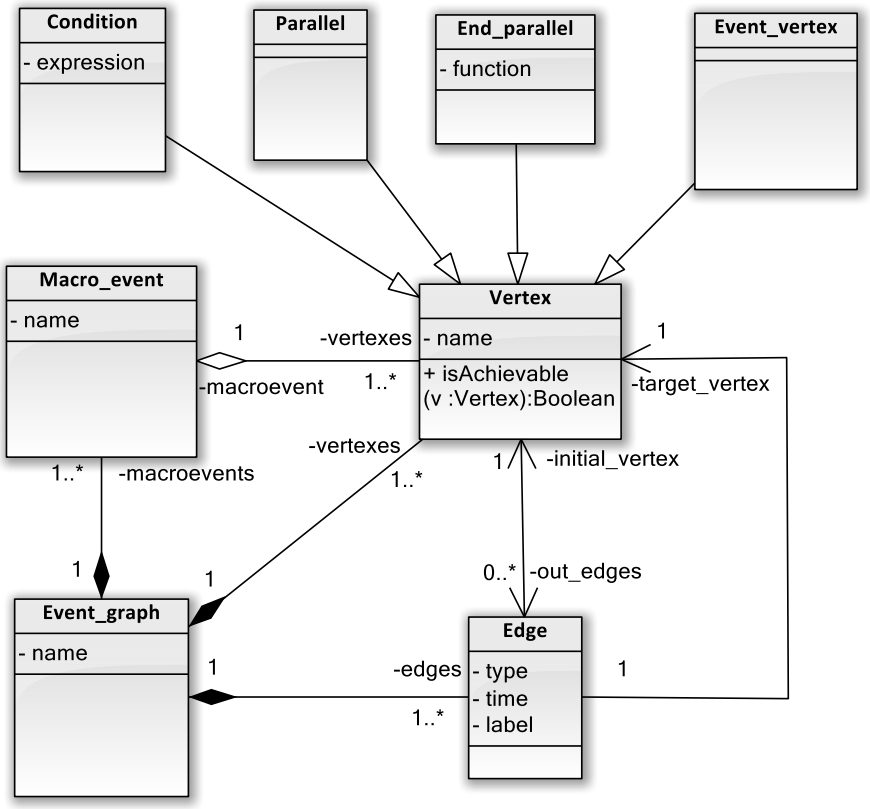

Рис. 1. Метамодель языка событийных графов

- *Edge* – дуга любого типа (тип определен как атрибут класса);
- *Macro_event* – подмножество вершин событийного графа, объединенных в одно макрособытие;
- *Event_graph* – событийный граф в целом.

На диаграмме приведены атрибуты классов, предназначенные для спецификации графических элементов языка:

- *name* – уникальное имя графического элемента;
- *expression* – логическое выражение вершины второго типа;
- *function* – логическая функция определения окончания параллельных участков выполнения процесса;
- *type* – атрибут класса дуги, определяющий её тип, множество значений атрибута: {1, 2, 3, 4}, где «1» – дуга мгновенного следования, «2» – дуга следования с задержкой, «3» – дуга мгновенной отмены, «4» – дуга отмены с задержкой;
- *time* – временная задержка для дуг второго и четвёртого типа;
- *label* – метка спецификации дуги.

Метамодель не всегда полно отражает синтаксис некоторого языка, поэтому она дополняется ограничениями в виде утверждений на языке OCL [Warmer 1999].

Для полного описания ограничений языка событийных графов на рисунке 2 представлена диаграмма классов UML, демонстрирующая связь графических элементов языка и объектов имитационной модели, которые не имеют визуального представления.

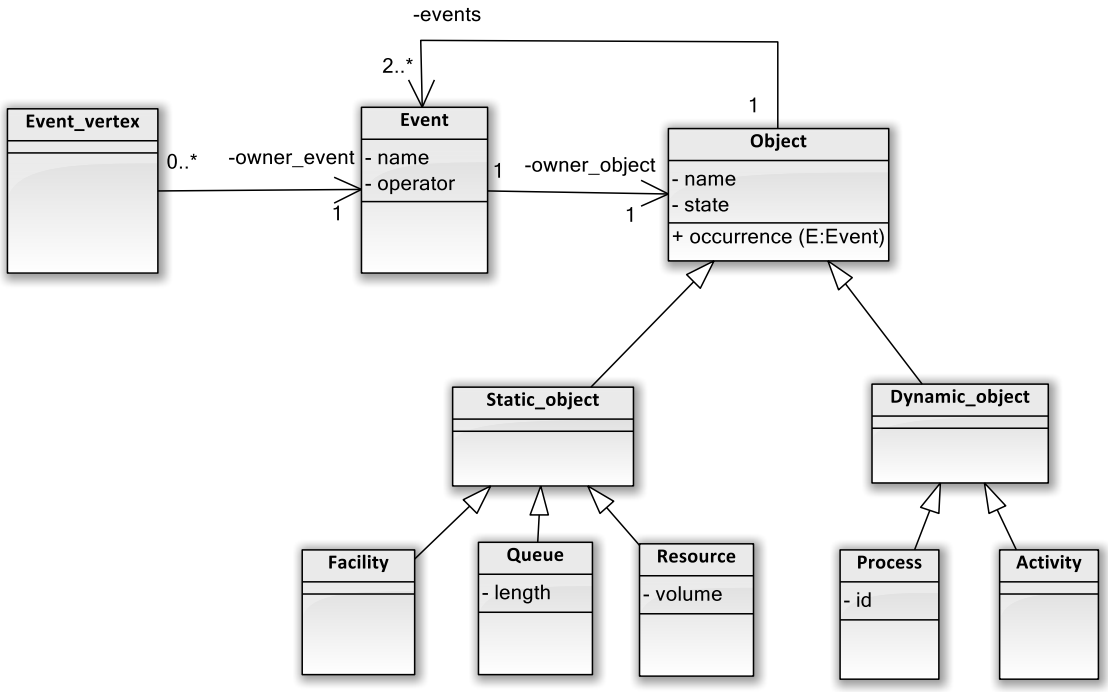

Рис. 2. Связь графических элементов СГ и объектов имитационной модели

На представленной диаграмме приведена иерархия объектов имитационной модели и показана их связь с событийными вершинами графа событий, где

- *Object* – объект модели любого типа (базовый класс объекта, инкапсулирующий общие свойства всех типов объектов);
- *Static_object* – статический объект модели, постоянно присутствующий в ней;
- *Dinamic_object* – динамический объект модели;
- *Facility* – объект-средство;
- *Queue* – объект-очередь;
- *Resource* – объект-ресурс (разделяемый ресурс);
- *Process* – объект-процесс (транзакт);
- *Activity* – объект-активность (элементарная деятельность);
- *Event* – объект-событие, определяющий функцию изменения состояния связанного с ним статического или динамического объекта модели;
- *Event_vertex* – событийная вершина из предыдущей диаграммы, определяемая некоторым одним объектом-событием.

На диаграмме также представлены следующие атрибуты классов:

- *name* – уникальное имя объекта модели;
- *state* – атрибут, определяющий текущее состояние объекта;
- *operator* – функция (оператор) изменения состояния связанного объекта;
- *length* – максимально-допустимый размер очереди;
- *volume* – максимальный объём ресурса, предоставляемый для использования процессами;
- *id* – идентификатор процесса (процесс с нулевым идентификатором является процессом моделирования).

Общие ограничения отдельных элементов СГ

Первую группу ограничений составляют требования спецификации вершин и дуг и структурные ограничения для вершин по числу входящих и исходящих дуг.

Требования спецификации вершин и дуг

1. Все вершины (e_i, p, a и b) и дуги второго s_{ij} и четвертого c_{ij} типов должны быть помечены.
2. Все метки должны быть специфицированы.

Следующую группу ограничений составляют требования к числу входящих и исходящих вершин для всех типов вершин событийного графа.

Событийная вершина e_i может иметь 0 или более входящих дуг и 0 или 1 исходящую дугу. Ограничение на число входящих и исходящих дуг событийной вершины зависит от ее типа.

3. Событийная вершина начала процесса моделирования $e_{нПм}$ не должна иметь входящих дуг и должна иметь одну исходящую дугу следования (первого или второго типа). Эта вершина является особой, инициализирующей вершиной в событийном графе. Формальное представление ограничения:

Context *Event_vertex* ***inv:***

*Process->**select**(id=0).events ->**includes**(self.owner_event)* ***and***
self.owner_event.operator= «Начало» ***and***
*Edge ->**forAll** (target_vertex<>self)* ***and***
*self.out_edges ->**size()** = 1* ***and***
self.out_edges.type< 3

Здесь *operator* – тип изменения состояния объекта при наступлении события *Event,* определяющего вершину *Event_vertex* (для динамического объекта: *«Начало»* или *«Конец»)*;

events – множество объектов-событий, изменяющих состояние соответствующего статического или динамического объекта;

owner_event – родительский объект-событие для событийной вершины *Event_vertex*;

target_vertex – целевая вершина дуги;

out_edges.type – тип исходящей дуги из событийной вершины *Event_vertex*.

4. Событийная вершина конца процесса моделирования $e_{кПм}$ должна иметь не менее одной входящей дуги следования (первого или второго типа) и не должна иметь исходящих дуг. Эта вершина является терминальной, заключительной, вершиной в событийном графе. Формальное представление ограничения:

Context *Event_vertex* ***inv****:*

*Process->**select**(id=0).events ->**includes**(self.owner_event)* ***and***
self.owner_event.operator= « Конец» ***and***
*Edge ->**exists** (target_vertex = self* ***and*** *type < 3)* ***and***
*self.out_edges ->**isEmpty**()*

5. Остальные событийные вершины e_i должны иметь не менее одной входящей дуги следования (первого или второго типа) и не более одной исходящей дуги любого типа. Формальное представление ограничения:

Context *Event_vertex****inv:***

*Edge -> **exists(**target_vertex = self* ***and*** *type <3)* ***and***
*self.out_edges -> **size()** <= 1*

6. Логическая вершина P_i должна иметь не менее одной входящей дуги следования (первого или второго типа) и одну или две исходящие дуги любого типа.

Исходящие дуги помечаются значением логического выражения вершины P_i (например, 0 – ложно, 1 – истинно). Формальное представление ограничения:

Context *Condition*

inv: *Edge ->**exists (**target_vertex = self **and** type < 3)*

inv: *(self.out_edges->**size()** = 1 **or** self.out_edges ->**size()** =2) **and***
*self.out_edges->**forAll(**label <>“”)*

7. Вершина распараллеливания A_i должна иметь не менее одной входящей дуги следования (первого или второго типа) и не менее двух исходящих дуг любого типа. Формальное представление ограничения:

Context *Parallel* ***inv:***

*Edge ->**exists (**target_vertex = self **and** type < 3**) and***
*self.out_edges ->**size()**>= 2*

8. Вершина окончания распараллеливания B_i должна иметь не менее двух входящих дуг следования (первого или второго типа) и одну исходящую дугу любого типа. Вершина окончания распараллеливания B_i не может иметь входящую дугу отмены (третьего или четвертого типа).

Context *End_parallel*

inv: *Edge ->**select (**target_vertex = self **and** type <3**) ->size()**>=2*
inv:*self.out_edges ->**size()** = 1*
inv: *Edge ->**select (**target_vertex = self**) ->forAll (**type < 3**)*

Структурные ограничения событийного графа

Вторую группу ограничений составляют структурные ограничения совокупностей элементов (фрагментов) событийного графа и графа в целом, которые в свою очередь можно разделить на синтаксические и семантические ограничения.

Синтаксические ограничения

9. В имитационном СГ должна быть одна и только одна инициирующая событийная вершина, представляющая событие начала процесса моделирования $e_{нПм}$, которое является стартовым.

Формальное представление ограничения:

Context *Event_graph* ***inv:***

*self.vertexes ->**collect(**Event_vertex**) ->select(**ev: Event_vertex | Process-*
*>**select**(id=0).events ->**includes**(ev.owner_event) **and***
*ev.owner_event.operator= «Начало») ->**size()** = 1*

10. В имитационном СГ должно быть не менее одной завершающей событийной вершины, представляющей событие конца процесса моделирования $e_{кПм}$.

Формальное представление ограничения:

Context *Event_graph* ***inv:***

*self.vertexes ->**collect(**Event_vertex**) ->select(**ev: Event_vertex | Process-*
*>**select**(id=0).events ->**includes**(ev.owner_event) **and***
*ev.owner_event.operator= «Конец») ->**size()**>0*

11. Вершина распараллеливания A_i может быть соединена с одной вершиной любого типа несколькими дугами второго типа и только одной дугой первого, третьего и четвертого типа.

Формальное представление ограничения:

Context *Parallel* ***inv:***

*self.out_edges ->**select (**e1:Edge, e2:Edge | e1.target_vertex =*
*e2.target_vertex **and** (e1 <>e2 **implies** e1.id<>e2.id))->forAll(**type = 2**)*

12. Все вершины СГ должны быть достижимы из инициирующей событийной вершины $e_{нПм}$ после удаления из графа дуг отмены событий (третьего и четвертого типа).

Формальное представление ограничения:

Context *Event_graph* ***inv:***

*self.vertexes ->**forAll(**v:Vertex | Event_vertex ->**select(**ev: Event_vertex | Process->**select**(id=0).events ->**includes**(ev.owner_event)* ***and*** *ev.owner_event.operator= «Конец»).isAchievable (v))*

Метод *isAchievable(v:Vertex)* проверяет, достижима ли вершина *v*, указываемая в качестве параметра, из вызывающей данный метод вершины, не учитывая дуг отмены событий (третьего и четвертого типа). Оператор *select* в данном контексте вернет не множество вершин, а один экземпляр вершины (событийная вершина $e_{нПМ}$), из которой вызывается вышеуказанный метод.

13. Если в СГ есть событие изменения состояния некоторого объекта системы, то обычно, но не всегда, должно быть и событие противоположного изменения состояния: например, событие начала процесса $e_{нП}$ и событие конца процесса $e_{кП}$. Это требование соблюдения ***событийной полноты*** модели. Иногда событий с противоположными изменениями нет, если они происходят за пределами моделируемого фрагмента поведения системы. Но это требует специального объяснения, и в таком случае необходимо разработчику модели дополнительно проверить модель на возможность ошибки пропуска события.

Семантические ограничения

Третью группу ограничений составляют семантические ограничения совокупностей элементов (фрагментов) событийного графа и графа в целом.

14. В графе не должно быть безусловных циклов, содержащих дуги только первого типа, то есть циклов без логической (условной) вершины P_i, предусматривающей выход из цикла по некоторому условию. При наличии таких циклов возникает бесконечная генерация событий без продвижения модельного времени – модель «зависает». Кроме того, в этих циклах в одной из событийных вершин должна изменяться хотя бы одна переменная (например, переменная цикла), участвующая в логическом выражении P_i. В СГ могут быть безусловные циклы, содержащие дуги второго типа, такие циклы могут быть прерваны при наступлении события окончания моделирования.

15. В графе, удовлетворяющем ограничению 12, могут быть недостижимые вершины из-за наличия фиктивных путей. Фиктивным путем будем называть поток

$E^*=(e_1 \& P_1 \rightarrow^0 e_2 \& P_2 \rightarrow^0 \ \ldots\ e_{n-1} \& P_{n-1} \rightarrow^0 e_n)$ [Бабкин, Разиньков 2011],

состоящий из дуг первого типа, для которого логическое выражение

$P_1 \& P_2 \& \ldots \& P_{n-1} = 0.$

Поэтому все вершины СГ должны быть достижимы из инициирующей событийной вершины $e_{нПм}$ после удаления из графа дуг отмены событий (третьего и четвертого типа) при выполнении

$P_1 \& P_2 \& \ldots \& P_{n-1} = 1$

для всех потоков графа вида

$E^*=(e_1 \& P_1 \rightarrow^0 e_2 \& P_2 \rightarrow^0 \ \ldots\ e_{n-1} \& P_{n-1} \rightarrow^0 e_n).$

Структурные ограничения событийного графа в программно-реализуемой форме

Для СГ в программно-реализуемой форме справедливы все вышеперечисленные ограничения. Кроме того, выделенные макрособытия должны удовлетворять

следующим ограничениям, которые являются ограничениями программной реализации модели, то есть программно-реализационными ограничениями.

16. В событийном графе каждая вершина должна принадлежать одному и только одному макрособытию.

Формальное представление ограничения:

Context *Vertex* ***inv:***

*Macro_event ->**select(**me: Macro_event | me.vertexes ->**includes(**self**))**->*
size() *= 1*

17. Макрособытие должно содержать хотя бы одну вершину.

Формальное представление ограничения:

Context *Macro_event* ***inv:***

*self.vertexes ->**size()**>=1*

18. В каждом макрособытии должна быть единственная точка входа. Точкой входа называется вершина, в которую входит дуга любого типа из другого макрособытия событийного графа или входит дуга второго, третьего или четвертого типа из данного макрособытия [Бабкин 2005].

Формальное представление ограничения:

Context *Macro_event* ***inv****:*

*self.vertexes->**select**(v:Vertex | Edge->**exist(**ed:Edge | ed.target_vertex = v*
and *(ed.type> 1* ***or*** *ed.initial_vertex.macroevent<> self)))->**size()**<= 1*

19. Вершины одного макрособытия между собой должны соединяться только дугами первого типа, за исключением дуг второго, третьего и четвёртого типа, соединяющих вершину с точкой входа в данное макрособытие.

Формальное представление ограничения:

Context *Edge* ***inv:***

self.initial_vertex.macroevent = self.target_vertex.macroevent
and *self.type = 1*

20. Внутри макрособытия не должно быть параллельных участков, то есть не должно быть вершины третьего типа A_i, у которой более одной выходящей дуги первого типа.

Формальное представление ограничения:

Context *Parallel* ***inv:***

*self.out_edges ->**select(**type = 1**)**->**size()**<= 1*

21. Если в макрособытии присутствует вершина четвертого типа $\boldsymbol{B_i}$, то она должна быть точкой входа в это макрособытие, то есть в данную вершину должны входить только дуги второго типа или дуги первого типа из других макрособытий.

Формальное представление ограничения:

Context *End_parallel* ***inv:***

*Edge ->**select**(target_vertex = self**)**->*
forAll (*type = 2****or****(type = 1* ***and*** *initial_vertex.macroevent<>self.macroevent))*

Семантические ограничения событийной имитационной модели

Базисом событийного графа является множество объектов и переменных модели. Множество значений переменных модели определяется типом переменных. Например, для среды моделирования ESimPL [Бабкин, Разиньков 2011а] типы переменных включают типы языка C++ и специальные типы: Histogram – гистограмма, TUserProcess – указатель процесса.

Объекты модели могут быть статическими (средство, очередь, ресурс) и динамическими (процесс и активность). Каждому объекту модели соответствует переменная состояния с тем же названием. Переменные состояния объектов могут

иметь два значения: для статических – «занят», «свободен», для динамических – «выполняется», «не выполняется». Это ограничения множеств значений переменных состояния объектов.

Событие является динамическим объектом, не имеющим переменной состояния, поскольку выполнение события мгновенно. Событие является мгновенным изменением состояния одного из объектов модели [Бабкин 2005]. Минимальное число типов событий для каждого типа объекта два: для статических – «занятие», «освобождение», для динамических – «начало», «конец». Возможно добавление в модель дополнительных событий, которые могут быть либо типом, обобщающим введенные события, либо подтипом введенных событий, либо дополнительными событиями, не следующими из изменений переменных состояний модели. Для объекта *Очередь* в ESimPL генерируется три события: «занятие», «освобождение» – процесс уходит из очереди поскольку дождался обслуживания, «удаление» – ожидание обслуживания процесса прерывается и процесс удаляется из очереди.

Рассмотрим ограничения, связанные с объектом-событием *Event*.

22. Событие может быть запланировано только на будущий момент времени, поскольку нельзя планировать события в прошлое. Поэтому время в спецификации дуг второго типа (дуг планирования) и дуг четвертого типа (дуг отмены) должно быть неотрицательным. Время в спецификации может либо явно указываться, либо определяться в результате вычисления.

23. Может быть отменено только то событие, которое запланировано на будущий момент времени. Однако иногда удобно, чтобы упростить логику модели, отменять возможно запланированное событие. Поэтому может не являться нарушением ограничения (ошибкой), если несколько объектов отменяют событие по принципу «кто раньше успеет». В этом случае отменяется событие, которое не запланировано. Но это также может свидетельствовать и об ошибках в алгоритме взаимодействия объектов.

24. В списке событий всегда должно находиться хотя бы одно запланированное событие, например событие окончания процесса моделирования $e_{\text{кПм}}$.

Кроме графически представляемых элементов событийных графов, в событийной имитационной модели существуют скрытые (вспомогательные) объекты: статические (средство *Facility*, очередь *Queue*, ресурс *Resource*) и динамические (процесс *Process*, активность *Activity*) (рис. 2). С ними связаны соответствующие переменные состояний.

Для объектов *Facility*, *Queue*, *Resource* и *Process* ограничено множество возможных состояний. Ограничение на возможные состояния объекта можно задать перечислением этих состояний тем или иным способом. С другой стороны, поскольку проверка этого ограничения возможна только на стадии выполнения событийной имитационной модели, это ограничение можно сформулировать как ограничение на совершение определенных событий в определенном состоянии объекта.

Рассмотрим ограничения, связанные с объектом-средством *Facility*.

25. Средство *Facility* может находиться в двух состояниях: «занято» и «свободно».

Формальное представление ограничения:

Context *Facility* ***inv:***

self.state=0 ***or*** *self.state=1*

Дополнительные ограничения на изменения состояний объекта: нельзя занять занятое средство и нельзя освободить свободное средство.

Рассмотрим ограничения, связанные с объектом-очередью *Queue*.

26. Очередь *Queue* может находиться во множестве состояний $\{n \in N \mid n <= Queue::length\}$.

Формальное представление ограничения:

Context *Queue* ***inv:***

self.state>= 0 ***and*** *self.state<= self.length*

Дополнительные ограничения на изменения состояний объекта: нельзя поставить процесс в заполненную очередь и взять процесс из пустой очереди.

Рассмотрим ограничения, связанные с объектом-ресурсом *Resource*.

27. Множество состояний ресурса *Resource* принадлежит отрезку *[0, Resource::volume]*.

Формальное представление ограничения:

Context *Resource* ***inv:***

self.state>= 0 ***and*** *self.state<= self.volume*

Дополнительные ограничения на изменения состояний объекта: нельзя занять ресурс процессом при недостаточном свободном объёме и освободить ресурс от процесса, который не занимал данный ресурс.

Рассмотрим ограничения, связанные с объектом-процессом *Process*.

28. Объекты данного типа являются динамическими, за исключением процесса моделирования (процесса с идентификатором 0), который присутствует в модели на протяжении всего моделирования. Процесс может находиться в двух состояниях: «выполняется» и «не выполняется».Формальное представление ограничения:

Context *Process* ***inv:***

self.state=0 ***or*** *self.state=1*

Дополнительные ограничения на изменения состояний объекта: нельзя удалить текущий процесс, если он является процессом моделирования, то есть имеет идентификатором 0).

Локальные ограничения

В модели дискретной системы могут быть дополнительные ограничения, определяемые спецификой предметной области. К ним могут относиться справедливые только для рассматриваемой системы ограничения на множество значений переменных модели, на события, происходящие только при определенных значениях переменных, и ограничения на допустимые последовательности событий. Поскольку в рассматриваемом подходе в среде моделирования ESimPL активности не сопоставляется переменная состояния, ограничения, связанные с объектом-активностью, также относятся к локальным ограничениям.

Заключение

На основе рассмотренной системы ограничений разработана подсистема проверки правильности событийных моделей среды моделирования ESimPL, включающая выявление ошибок на этапах разработки исходного событийного графа и преобразования его в программно-реализуемую форму, а также на этапе прогона программных моделей.

Библиографический список

Бабкин Е.А. Событийные модели дискретных систем / Курск.гос. ун-т. Курск, 2005. 18 с. Деп. в ВИНИТИ 14.01.05, № 30–В2005.

Бабкин Е.А. О синтезе событийных моделей дискретных систем [Электронный ресурс] // Ученые записки. Электронный научный журнал Курского государственного университета. 2006. № 1. URL: http://www.scientific-notes.ru/pdf/s15.pdf (дата обращения: 12.05.2012). № 77-26463

Бабкин Е.А., *Разиньков* В.В. *О формальном представлении* событийных моделей дискретных систем // Информационные системы: Теория и практика: сб. науч. работ. Вып. 2 / редкол.: Е.А. Бабкин, В.А. Кудинов, И.В. Селиванова; отв. ред. Е.А. Бабкин; фак. информатики и вычислит. техники Курск. гос. ун-та. Курск, 2011. С. 60–67.

Бабкин Е.А., *Разиньков* В.В. *Учебная система* имитационного событийного моделирования // Международный научный журнал Acta Universitatis Pontica Euxinus. Спец. вып.: сб. материалов VII междунар. конф. «Стратегия качества в промышленности и образовании» – 3–10 июня 2011 г., Варна, Болгария, Т. 3, Днепропетровск, Варна, 2011а. С. 562–564.

Замятина Е.Б., Шафранов А.В. Построение синтаксически и семантическиправильной QueueNetwork-модели в Triad.Net* // Вестник Пермского университета. Математика. Механика. Информатика. 2011. Вып. 4(8). С. 83–91.

Кознов Д. В. Основы визуального моделирования. – М: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. 246 с.

Пепеляев В. А., *Чёрный* Ю. М. О современных подходах к оценке достоверности имитационных моделей // Первая всероссийская научно-практическая конференция «Опыт практического применения языков и программных систем имитационного моделирования в промышленности и прикладных разработках» / «ИММОД-2003»: сб. докл. Т. 1. СПб.: ФГУП ЦНИИ технологии судостроения, 2003. С. 142–146.

Balci O. Verification, validation and accreditation // Proceedings of the 1998 Winter Simulation Conference. 1998. P. 41–48.

Warmer J., Kleppe A. The Object Constraint Language. Precise Modeling with UML. Addison Wesley, 1999. 144 p.