электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
издатель ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана»

Метод реализации оптимизированного имитационного моделирующего алгоритма

77-48211/649826

 # 11, ноябрь 2013

Черненький М. В., Садовников В. В.

УДК 681.5

Россия, МГТУ им. Н.Э. Баумана
sunday@nxt.ru

При построении среды имитационного моделирования существуют два типовых подхода к построению моделирующего алгоритма: линейный и сканирующий [1]. С точки зрения оптимальной компьютерной реализации моделирующий алгоритм линейного типа более предпочтителен, чем моделирующий алгоритм сканирующего типа, так как его затратность близка к единице [2]. Под затратностью понимается отношение полного количества операций, необходимых для реализации одного класса одновременных событий, к общему количеству операций в подпрограммах событий. Но помимо затратности также необходимо учитывать и, так называемую, вычислительную сложность алгоритма, состоящую из временной и пространственной сложности.

Вычислительная сложность является показателем масштабируемости алгоритма. В практическом смысле этот показатель характеризует степень увеличения времени работы и объёма памяти, необходимых алгоритму при увеличении объёма входных данных [3]. Если мы хотим построить универсальный алгоритм для системы имитационного моделирования, мы должны гарантировать, что наш алгоритм будет устойчиво работать и не превышать разумные временные рамки даже в тех случаях, когда число инициаторов процессов в системе будет порядка миллиона. Это также, но в меньшей степени, относится и к количеству дискретных периодов времени, для которых производится моделирование.

Учитывая специфику области имитационного моделирования, а именно её универсальность и динамичность, при выборе среды разработки целесообразно отдавать предпочтения современным объектно-ориентированным языкам с нестрогой типизацией, динамическими классами, ассоциативными массивами и ссылочными моделями памяти. Эти возможности существенно облегчат процесс разработки системы имитационного моделирования. Хотя такие высокоуровневые языки сами по себе менее производительны,

чем языки со статическими моделями памяти, но с их помощью мы сможем рационально использовать аппаратные ресурсы платформы. Ссылочные модели памяти позволяют хранить и получать наиболее быстрый доступ к значительным объёмам статистической информации, накопленной в ходе работы алгоритма.

Например, с помощью ассоциативного массива (словаря множества пар «ключ-значение») можно с наименьшими затратами организовать динамическую по своей природе локальную среду инициатора или блока, имена параметров которых будут не числовыми а строковыми. Кроме того, нетипизированные ассоциативные массивы способны хранить произвольные форматы данных, что, кроме всего прочего, предполагает также вложенность массивов. В итоге, мы получаем возможность организовать смешанную (скаляр, строка, метка, ссылка) и многомерную (вектор, таблица, пространство) структуру данных [4].

При использовании объектно-ориентированного подхода к реализации моделирующего алгоритма линейного типа вся логическая нагрузка, включающая подпрограммы активных и пассивных событий, переносится в команды Блоков. А такие элементы системы, как Таблица Будущих Времён, Инициаторы, Таблица Условий, Класс Одновременных Событий, выступают в качестве глобальных параметров для этих команд Блоков.

Прежде чем перейти к схеме работы алгоритма рассмотрим логические структуры объектов алгоритма, представленные на рисунке 1.

Рисунок 1. Структура объектов алгоритма

Приведём некоторые пояснения к обозначениям на рисунке 1:

• И (Инициатор) — объект, содержащий ссылку на локальную среду, следующую команду и время активации;

• Расписание — объект, включающий в себя функции и свойства Таблицы Будущих Времён, Таблицы Текущих Событий, Класса Одновременных Событий и Календаря;

- Команда — объектный аналог оператора языка проектирования (например: уничтожить ИНИЦИАТОР; TERMINATE);
- Буфер команды — аналог Таблицы Условий, который хранит инициаторы, ожидающие выполнения условия указанного в команде.

Рассмотрим объект Расписание подробнее. Расписание является упорядоченным по времени списком инициаторов и реализует следующие методы (функции):

- добавление инициатора — процедура включает в себя сортировку нового инициатора по времени, что происходит довольно быстро, потому как список инициаторов всегда отсортирован, а новый объект не может повлиять на отношение порядка уже имеющихся элементов;
- получение следующего активного инициатора — процедура возвращает первый инициатор в списке и переводит системное время к моменту, соответствующему времени активизации возвращаемого инициатора.

Представленный на рисунке 2 алгоритм является обобщённым и может быть применен для имитации моделей, основанных как на языке ПОСП [5], так и GPSS [6].

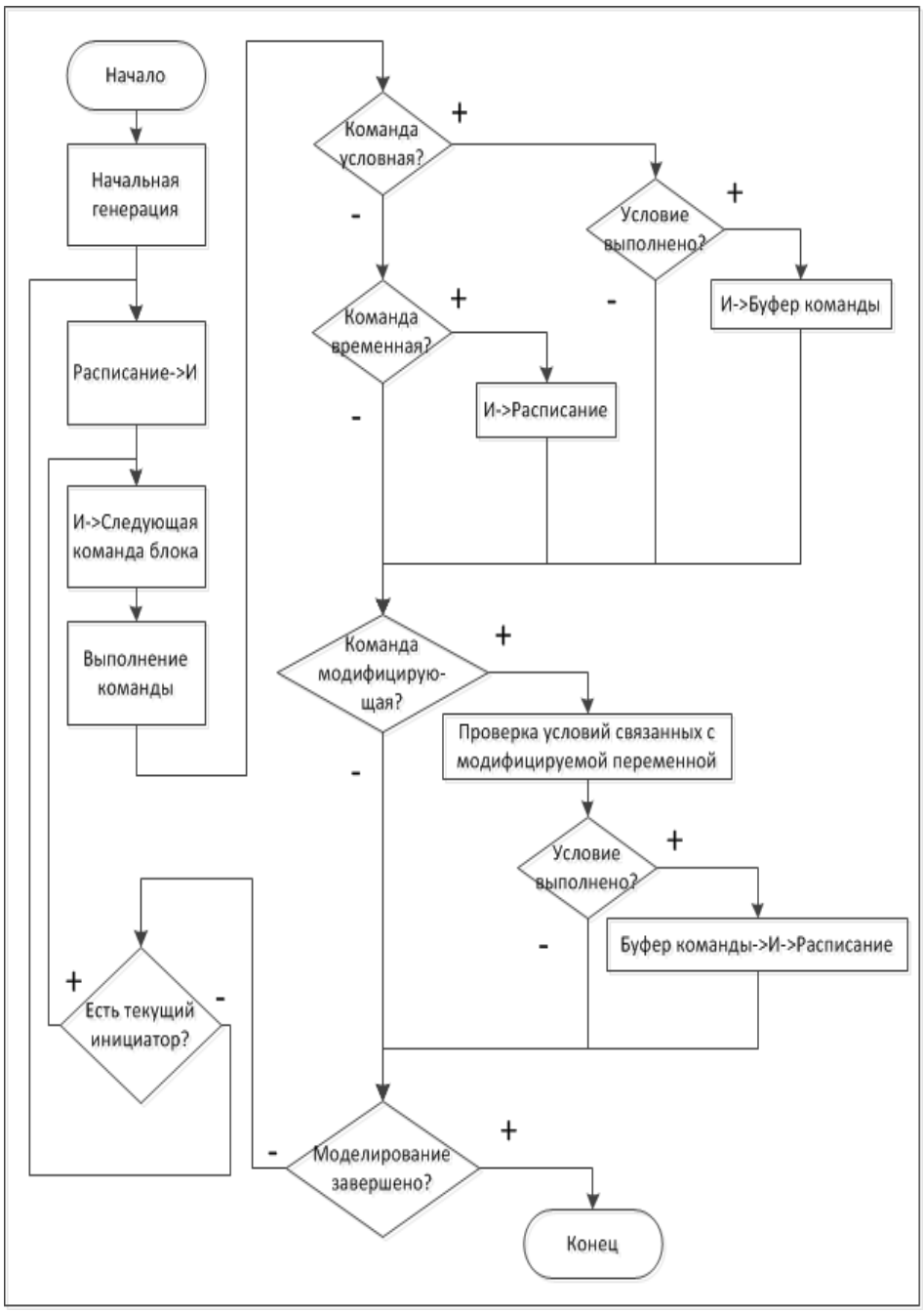

Рисунок 2. Блок-схема реализации моделирующего алгоритма

Рассмотрим схему работы алгоритма. В момент начальной генерации формируется список генераторов или агрегатов, и Расписание заполняется первыми инициаторами. Далее извлекается из Расписания первый по времени Инициатор, который затем активирует следующую Команду. Затем исполняется код команды. На данном этапе может происходить модификация параметров системы и, по возможности, определение адреса следующей команды для Инициатора. Далее развитие алгоритма зависит от типа Команды. В рамках данного алгоритма Команды делятся на три типа: временные, условные, модифицирующие:

- Условные Команды содержат логическое условие. Если условие не выполнено, то

Инициатор помещается в Буфер текущей Команды до тех пор, пока условие не будет выполнено.

- Модифицирующие Команды изменяют параметры текущего состояния системы. Если в Буферах условных Команд есть инициаторы, которые ожидают изменения данных параметров, то мы проверяем соответствующие условия. Если условия выполнены, то соответствующие инициаторы извлекаются из Буфера и помещаются в Расписание.

- Временные Команды задерживают инициатор на определённое время. В таком случаи Инициатор помещается в Расписание. Следует также отметить, что условно-временная Команда считается просто условной.

В итоге одна итерация алгоритма завершается, что соответствует одному элементарному классу одновременных событий. Далее необходимо проверить, не наступило ли условие конца работы моделирующего алгоритма. Если да, то в таком случае работа моделирующего алгоритма закончена, и мы можем приступать к обработке статистических данных, накопленных в результате моделирования. Если же нет, то происходит переход на следующую итерацию алгоритма. Причём, если последняя команда была модифицирующая, то инициатор без промедления может отправиться на активацию следующей команды, иначе следующий инициатор извлекается из расписания.

Теперь рассмотрим некоторые оптимизационные решения, связанные с вышеизложенным алгоритмом. Расписание намеренно представлено в виде единой структуры, поскольку линейный порядок инициаторов в Таблице Будущих Времён не может быть нарушен, а порядок следования инициаторов в Таблице Текущих Событий не может быть однозначно определён, вследствие того, что мы рассматриваем процессы как часть квазипараллельного процесса. Учитывая это, мы можем ускорить работу алгоритма Календаря за счёт упорядочивания инициаторов по времени методом двоичных вставок при их добавлении в Расписание. Причём первый элемент в Расписании гарантированно ссылается на один из параллельных процессов текущего момента времени. Таким образом, временная сложность добавления в Расписание зависит от $\log_2 n$, где n — число инициаторов в Расписании, а операция извлечения пропорциональна n. Помимо этого единая таблица позволяет существенно сократить объёмы выделяемой памяти.

Предположим, что на выполнение одной команды и определение ее типа уходит некоторое постоянное число операций K. Что касается Буферов условных Команд, то они намеренно не сводятся в общую Таблицу Условий, так как временная сложность проверки условия равна $C * k$, где C — некая константа, зависящая от сложности условия и алгоритма синтаксического анализа выражения, а k — количество условий, которые нужно проверить.

В итоге временная сложность одной итерации алгоритма [7] может быть выражена

формулой

$$T_i(n,k) = K + \log_2 n + C * k. \quad (1)$$

Причём если текущая Команда

- временная, тогда $T_i = K + \log_2 n$,
- условная, тогда $T_i = K$,
- модифицирующая, тогда $T_i = K + \log_2 n + C * k$.

На практике, для упрощения формулы используют, так называемую, полиномиальную временную сложность для лучшего и худшего случаев. Полиномиальная временная сложность – это оценочная функция, которая выражается степенью многочлена (полинома) от аргумента, который характеризует объём входных данных. Соответственно, в нашем случае, полиномиальная временная сложность равна:

- в худшем случае $O(k) = k$, где k — количество проверяемых условий;
- в лучшем случае $\Omega(n) = \log_2 n$, где n — количество объектов находящихся в Расписании.

В результате мы определили, что вычислительная сложность представленного алгоритма в худшем случае будет возрастать линейно относительно объёма входных данных, что является вполне приемлемым результатом. Также следует отметить, что сложность алгоритма стремится к лучшему случаю $\Omega(n)$, когда моделируемая система содержит минимальное число условий, затрагивающих локальную среду инициатора, то есть, когда основные проверки связаны с параметрами блоков или агрегатов. В таком случае нет необходимости проверять условия для каждого инициатора, а достаточно лишь одной проверки.

На практике, мы довольно редко сталкиваемся с худшим случаем. Если говорить в терминах конкретного языка, то это происходит только с применением команды вида:

ждать ИНИЦИАТОР –> <*локальная среда*> <*условие*> -- для языка ПОСП;

TEST <*условие*> -- для языка GPSS.

Теперь проанализируем затратность алгоритма, которая может быть оценена по формуле, предложенной в работе [2]:

$$q \approx 1 + \frac{L * P}{2 * a}, \quad (2)$$

где

q — затратность;

L — количество проверяемых условий;

P — количество действий, необходимых для проверки одного условия;

a — количество действий, необходимых для выполнения события.

Если выразить оценку затратности в терминах данной статьи, то мы получим формулу в виде

$$q \approx 1 + \frac{\tilde{k} * C}{2 * K}, \quad (3)$$

где $\tilde{k}$ — среднее количество проверяемых условий.

Учитывая, что C и K константы и допуская, что $C \cong K$ вне зависимости от конкретной реализации алгоритма, можно сделать вывод, что оценка затратности сводится к:

$$q \approx 1 + \frac{\tilde{k}}{2}. \quad (4)$$

Если проанализировать k, то оно с большой долей вероятности равно 1, за исключением тех случаев, когда алгоритм доходит до “проблемной” команды (**ждать** <*условие*>**, TEST**), которая затрагивает локальную среду. В этом случае k может быть много больше единицы. Но поскольку мы распределяем условия по соответствующим блокам, то k получается много меньше размера полной Таблицы Условий. Тогда $\tilde{k}$ можно представить в виде

$$\tilde{k} \sim \frac{K_{hard}}{K_{total}} * \frac{k_{total}}{K_{hard}}, \quad (5)$$

где

K_{hard} — количество “проблемных” команд;

K_{total} — полное число команд в моделируемой системе;

k_{total} — полное количество необработанных условий;

$\frac{K_{hard}}{K_{total}}$ — приблизительная вероятность входа Инициатора в “проблемную” команду;

$\frac{k_{total}}{K_{hard}}$ — приблизительный размер Буфера Команды.

Подставляя полученную оценку для $\tilde{k}$ в формулу (4), оценка затратности сведется к виду

$$q \sim 1 + \frac{k_{total}}{2 * K_{total}}. \quad (6)$$

Исходя из итоговой формулы, можно сделать вывод, что при моделировании довольно крупной имитационной модели, затратность нашего алгоритма будет стремиться к единице. Это стало возможным благодаря децентрализации Таблицы Условий и перенесению логики управления временем активации инициаторов и анализа условий в операторы Команд системы.

В заключении следует отметить, что конечная реализация метода создания интерпретатора, описываемого данным алгоритмом, будет несколько различаться по степени затратности, но не сильно зависеть от сложности имитационной модели.

Список литературы

1. Черненький В.М. Алгоритмы генерации процесса имитации //Наука и образование, МГТУ им. Н.Э. Баумана. Электрон.журн. 2011. № 11. URL: http://technomag.edu.ru/doc/292147.html
2. Черненький В.М. Формирование имитационного процесса на основе алгоритмического описания функционирования информационной системы //Наука и образование, МГТУ им. Н.Э. Баумана. Электрон.журн. 2011. № 11. URL: http://technomag.edu.ru/doc/291975.html
3. Sanjeev Arora, Boaz Barak. Computational Complexity: A Modern Approach: Dated January 2007, Princeton University
4. Черненький В.М. Алгоритмическая модель описания дискретного процесса функционирования системы. //Наука и образование, МГТУ им. Н.Э. Баумана. Электрон.журн. 2011. № 11. URL: http://technomag.edu.ru/doc/292620.html
5. Учебное пособие по GPSS World /пер. с англ./ Казань: Изд-во «Мастер Лайн», 2002. 272 с.
6. Руководство пользователя по GPSS World /пер. с англ./ Казань: Изд-во «Мастер Лайн», 2002. 384 с.
7. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: Построение и анализ, 2-е изд. М.: Издательский дом "Вильямс", 2005. 1296 с.