УДК 681.324

М.А. Волк

Харьковский национальный университет радиоэлектроники, Харьков

ЖУРНАЛИЗАЦИЯ СОСТОЯНИЙ ПРОГРАММНЫХ РАСПРЕДЕЛЕННЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ И ЕЕ ИСПОЛЬЗОВАНИЕ В ОПТИМИСТИЧЕСКИХ АЛГОРИТМАХ СИНХРОНИЗАЦИИ

В работе представлено развитие формального аппарата описания распределенных имитационных моделей на основе их программного представления в терминах процессной алгебры. Вводится множество активностей журнализации состояний модели, которые составляют основу менеджера памяти имитационных моделей. Показано, что реализованные активности могут выполнять функции федератов в оптимистических алгоритмах синхронизации распределенных имитационных моделей.

Ключевые слова: *распределенные имитационные модели, процессная алгебра, состояние модели, данные, активность, менеджер памяти, оптимистические алгоритмы синхронизации*

Введение

В настоящее время существующие формальные аппараты описания распределенных имитационных моделей, систем моделирования и алгоритмов их взаимодействия имеют общий недостаток, заключающийся в высоком уровне абстракций, который не позволяет достаточно формализовать процесс построения моделирующей среды с целями создания распределенных моделей и автоматизации управления ими. Наиболее известными способами описания являются структурно-алгоритмический [1], на основе процессной алгебры [2 – 5], объектно-ориентированный, например, на основе стандарта HLA (High Level Architecture) [6, 7].

Структурно-алгоритмический способ показывает жесткую структуру имитационной модели, объединяющую в одном вычислительном процессе, как модель, так и систему моделирования. Такой подход эффективен либо в случае написания закрытых моделей, либо в том случае, когда модель создается непосредственно внутри моделирующей среды.

Процессная алгебра дает хороший математический аппарат описания процессов, но не отвечает на вопрос практической реализации этих процессов.

Появление стандартов, таких как HLA, направлено, главным образом, на унификацию обмена данными между моделями и не затрагивает особенности внутренней организации моделей.

В работе [8] вводиться формальное представление имитационной модели, реализованной программно. Показано, что любая программная модель может быть представлена в виде двух структурных составляющих: активности (A), отражающей поведение системы, и данных (dm), отражающих состояние модели в конкретный момент времени. Для реализации функций управления состоянием имитационной модели во времени необходима журнализация (сохранение информации о состоянии модели в конкретные моменты времени). Целью данной работы является формальное рассмотрение реализации такого механизма на основе ограниченного набора активностей, реализующих журнализацию.

Множество активностей, реализующих журнализацию

Переход модели из одного состояния в другое связан напрямую с изменением данных модели: $\mathrm{dm} \xrightarrow{\mathrm{A}} \mathrm{dm}'$. В связи с тем, что продвижение модельного времени выполняет, как правило, специальная активность модели, изменение состояния модели происходит в фиксированный момент модельного времени Tm_k, где *k* является целым числом и $k \in [1, \mathrm{Tm}_{\text{стоп}}]$, $\mathrm{Tm}_{\text{стоп}}$ – время останова процесса моделирования, может принимать значения $\mathrm{Tm}_{\text{стоп}} \in [1, \infty]$. Таких изменений состояния в момент времени Tm_k может быть несколько со стороны одной или множества активностей модели.

В работе [1] введены две активности, реализующие запись состояния модели одним из двух способов:

1. Сохранением всех данные модели (дамп памяти модели). Данный способ дает надежный механизм возврата в любой момент в прошлом, для которого существует дамп памяти. Данному способу поставим в соответствие активность $\mathrm{A}_{\text{дамп}}(\mathrm{Tm}_k)$ – активность, позволяющую модели запомнить все свои данные в момент Tm_k.

2. Запоминанием изменений данных модели. Если объем данных модели значительно превышает объем изменяемых значений, то можно запоминать адрес изменяемых данных и их новое значение. Со временем образуется цепочка таких изменений. Активность, соответствующая данному способу – $\mathrm{A}_{\text{фикс}}(\mathrm{dm}_i, t_k)$ – фиксирует изменение значения элемента данных dm_i ($\mathrm{dm}_i \in \mathrm{dm}\ i = \overline{1, N}$) в момент Tm_k.

Здесь N – количество частных моделей, на которые разбита исходная модель.

Сопоставим каждой из этих активностей обратную, то есть такую, которая возвращает модель в одно из состояний в прошлом: $A'_{дамп}(Tm_k)$ и $A'_{фикс}(dm_i, t_k)$. Фактически, обратная активность A' реализует операцию $dm' \xrightarrow{A'} dm$, возвращая модель в состояние до запуска активности A. Наличие зависимости от времени позволяет возвращать состояние модели в какой-либо момент в прошлом, начиная с которого были выполнены несколько активностей.

Каждый из приведенных способов является достаточным, чтобы сохранить информацию о состояниях модели. Однако каждый из них имеет свои достоинства и недостатки.

Первый способ ведет к большим временным и ресурсным затратам, так как требуется значительное процессорное время на копирование данных и линейно растущий объем памяти для хранения всех состояний модели на протяжении времени моделирования.

Второй, хоть и лишен значительных ресурсных затрат при сохранении, требует повышенных вычислительных затрат при реализации отката модели во времени. Действительно, для того, чтобы вернуть модель в прошлое, необходимо пройтись по цепочке событий в обратном порядке. А тот факт, что изменение данных модели при фиксированном модельном времени происходит неоднократно, увеличивает время работы алгоритма, построенного по этому способу.

Рациональным видеться построение такой системы журнализации, которая реализует оба способа.

Для формального определения вышесказанного введем оператор sizeof(dm_i), определяющий размер памяти (V, байт), необходимой для хранения элемента данных dm_i и оператор time(A), определяющий реальное время выполнения некоторой активности (T, секунд). Запись sizeof(dm_i,t_k) подчеркивает тот факт, что одновременно с данными необходимо сохранить и значение времени, когда это изменение произошло.

Очевидными являются следующие выражения:

$$\sum_{i=1}^{N} \text{sizeof}(dm_i) = \text{sizeof}(dm), \quad \text{если } dm_i \cap dm_j = \varnothing \ \ \forall i, j \in \overline{1,N}, i \neq j; \qquad (1)$$

$$\sum_{i=1}^{N} \text{sizeof}(dm_i) > \text{sizeof}(dm), \quad \text{если } \exists dm_i \cap dm_j \neq \varnothing, \ \ \forall i, j \in \overline{1,N}, i \neq j; \qquad (2)$$

$$\text{time}(A_{дамп}) \geq \text{time}(A_{фикс}(dm_i, t_k)). \qquad (3)$$

Выражение (1) говорит о том, что если все частные модели проведут сохранение своего состояния, то размер памяти, который потребуется для этого будет равен размеру памяти, необходимому для сохранения состояния всей модели при условии, что данные модели не пересекаются. В противном же случае, эта память будет больше (выражение (2)).

Выражение (3) подчеркивает тот факт, что время, затрачиваемое системой моделирования на выполнение дампа памяти в общем случае больше времени, которое затрачивается на сохранение состояния частной подмодели.

Менеджер памяти

Большинство современных имитационных моделей реализуют механизм управления памятью своими средствами. Например, любая модель, написанная в рамках стандарта HLA должна предоставлять системе моделирования федераты (federates), отвечающие за возврат модели во времени, а как реализован федерат – дело разработчика модели или системы моделирования (рис. 1, а).

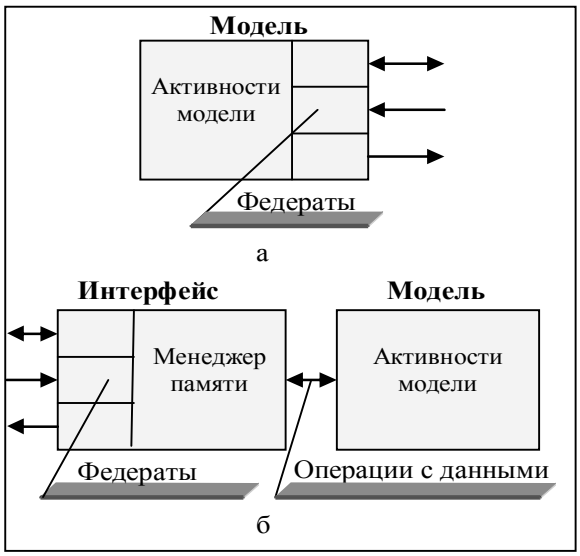

Рис. 1. Реализация федератов: а – внутри модели; б – посредством менеджера памяти

Предлагается использовать другой подход – ввести модуль менеджера памяти, который выполняет журнализацию изменения данных модели (рис. 1, б). Реализация этого действия открывает принципиальную возможность реализовать любой из федератов, управляющих состоянием модели во времени и освободить разработчика модели от необходимости реализации этого интерфейса вручную.

Обозначим через A_{mem} множество активностей менеджера памяти. Используя введенные ранее активности, определим данное множество как

$$A_{mem} = \{A_{дамп}(T_k), A_{фикс}(dm_i, t_k), A'_{дамп}(T_k), A'_{фикс}(dm_i, t_k), \ldots\}. \qquad (4)$$

Как утверждается выше, частной модели достаточно пользоваться одним из видов активностей журнализации. Поэтому, дальнейшие рассуждения будем проводить на минимальном наборе:

$$A_{mem} = \{A_{дамп}(T_k), A'_{дамп}(T_k), ...\}. \quad (5)$$

Рассмотрим перечень федератов HLA и покажем возможность реализовать их в терминах активностей (5).

Оптимистические федераты

Согласно стандарту HLA федераты, поддерживающие оптимистические алгоритмы управления модельным временем должны предоставлять две интерфейсные функции [7]:

ReflectAttributeValues ();

FlushQueueGrant (T).

В момент завершения всех активностей в текущий момент времени T_k, федерат вызывает функцию *FlushQueuetRequest*(T_k), чем сообщает системе RTI (Runtime Infrastructure) о готовности продолжать моделирование и увеличить значение модельного времени. RTI передает оптимистическому федерату очередь сообщений с временными метками TSO (Time Stamp Order), существующую в системе на этот момент времени. Передача осуществляется путем вызова интерфейсной функции *ReflectAttributeValues*(). Далее федерат продвигает свое модельное время на основе этой промежуточной упорядоченной очереди сообщений. Если федерат обнаруживает сообщение, с временной меткой меньшей текущего значения локальных часов, он самостоятельно осуществляет откат, рассылая при необходимости антисообщения вызовом функции RTI *Retract* (). Система RTI время от времени посылает федерату значение GVT(Global Virtual Time) при помощи функции *FlushQueueGrant(GVT)*. Сопоставим описанным функциям некоторые активности для упрощения представления процесса моделирования (выполним переименование, [9]):

$P[f_1]=P[FlushQueuetRequest/A_{fqr}$

$ReflectAttributeValues/A_{rav}$,

$FlushQueueGrant /A_{fqg}$,

$Retract/A_{retr}$].

В случае использования менеджера памяти, для простоты объяснения, примем, что дамп памяти частная модель осуществляет после выполнения всех активностей в текущий момент времени T_k, следовательно, с точки зрения логики работы RTI, выполнение активности $A_{дамп}(T_k)$ из множества (5) может являться префиксным процессом к выполнению функции *FlushQueuetRequest(*T_k*)*, или, другими словами, совершение моделью дампа памяти (сохранение состояния) влечет за собой информирование системы RTI о завершении всех действий частной модели в момент модельного времени T_k.

Менеджер памяти обязан предоставить RTI функцию *ReflectAttributeValues()*, которая вызывается для передачи частной модели очереди сообщений. При обнаружении сообщения с временной меткой, меньшей текущего значения, менеджер памяти обязан осуществить откат. Представление данного процесса:

$$dm_{T_k} \xrightarrow{(A_{дамп}(T_k).A_{fqr}.A_{rav})} dm_{T_k} \cup dm_{TSO} \xrightarrow{A_{TSO}} \begin{cases} dm_{T_k}, если T_{min(TSO)} \geq T_k \\ \xrightarrow{A'дамп(T_{min(TSO)})} dm_{T'_{min(TSO)}}, если T_{min(TSO)} < T_k \end{cases}, \quad (6)$$

где A_{TSO} – активность менеджера памяти, осуществляющая просмотр очереди сообщений; dm_{TSO} – сама очередь сообщений; $T_{min(TSO)}$ – минимальное значение временной метки сообщений в очереди TSO.

Выражение (6) показывает, что при соблюдении частной моделью правила выполнения дампа состояния в конце цикла моделирования, менеджер памяти способен совершить откат к состоянию частной модели в конкретный момент в прошлом самостоятельно, без участия самой модели. Обозначение $dm_{T'_{min(TSO)}}$ в выражении (6) подчеркивает тот факт, что в момент модельного времени $T_{min(TSO)}$ частная модель могла не совершать действий. В этом случае откат производиться до ближайшего дампа состояния частной модели $T'_{min(TSO)} \leq T_{min(TSO)}$.

Заметим, что активность A_{retr} является активностью модели, причем она может поддерживаться или нет, что не влияет на функциональность менеджера памяти по продвижению модельного времени.

Подводя итог, разобьем все множество активностей на группы по принадлежности одному из объектов моделирования: RTI, менеджеру памяти и частной модули:

$$\begin{cases} RTI : \{A_{fqr}, A_{rav}\} \\ A_{mem} = \{A_{дамп}(T_k), A'_{дамп}(T_k), A_{TSO}, A_{fqg}\}, \\ M : \{A_{retr} + \varnothing\} \end{cases} \quad (7)$$

где операция + – альтернативная композиция [2, 9].

Вызов активности A_{fqg} может быть проигнорирован либо использован менеджером памяти для удаления всех дампов памяти, временная метка которых меньше GVT (с целью экономии памяти).

Из выражения (7) видно, что разработчику частной модели из всего множества федератов необходимо реализовать только одну активность – A_{retr} и то при условии, что она необходима для обеспечения логики работы всей модели. Остальные активности являются стандартными либо для RTI либо

для предложенного менеджера памяти.

Выводы

В данной работе получил развитие формальный аппарат представления программных распределенных имитационных моделей на основе процессной алгебры. Введены базисные активности журнализации, обосновано введение менеджера памяти как интерфейса программных моделей, реализующего управление состоянием модели. Показана возможность построения на основе введенных активностей оптимистических алгоритмов синхронизации распределенных имитационных моделей на основе стандарта HLA.

Практическая значимость работы заключается в возможной унификации программных имитационных моделей, упрощении и ускорении процесса их построения на основе простого интерфейса, основанного на управлении данными модели. В дальнейшем планируется рассмотреть различные алгоритмы поведения моделей и алгоритмы синхронизации, созданные на базе описанного интерфейса.

Список литературы

1. Максимей И.В. Имитационное моделирование на ЭВМ / И.В. Максимей. – М.: Радио и связь, 1988. – 222 с.

2. Хоар Ч. Взаимодействующие последовательные процессы: пер. с англ. / Ч. Хоар. – М.: Мир, 1989. – 264 с., ил. ISBN 5-03-001043-2.

3. Bergstra J.A., Ponse A. and Smolka S.A., editors: Handbook of Process Algebra. North-Holland, Amsterdam, 2001.

4. A Calculus for Communicating Systems, LNCS92, 1980. – 171 p.

5. The Space and Motion of Communicating Agents, to appear, Cambridge University Press, 2009. – 200p.

6. IEEE STD 1278.1-1995. IEEE Standard for Distributed Interactive Simulation — Application Protocols. N.Y.: Institute of Electrical and Electronics Engineers, Inc., 1995.

7. ALLEN R. Formal modeling and analysis of the HLA component integration standard / R. ALLEN, D. GARLAN, J. IVERS // Proc. of the 6th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. – 1998. – P. 70-79.

8. Волк М.А. Процессное представление состояний распределенных имитационных моделей с учетом специфики их программной реализации / М.А. Волк // Вісник Національного технічного університету "Харківський політехнічний інститут": зб. наук. пр. Тематичний випуск: Інформатика і моделювання. – Х.: НТУ „ХПІ", 2009. – №13. – С. 23-33.

9. Миронов А.М. Теория процессов. – [Электронный ресурс]. – Режим доступа к статье: http://www.intsys.msu.ru.

Поступила в редколлегию 8.12.2009

Рецензент: д-р техн. наук, проф. О.Г. Руденко, Харьковский национальный университет радиоэлектроники Харьков.

ЖУРНАЛІЗАЦІЯ СТАНІВ ПРОГРАМНИХ РОЗПОДІЛЕНИХ ІМІТАЦІЙНИХ МОДЕЛЕЙ ТА ЇЇ ВИКОРИСТАННЯ В ОПТИМІСТИЧНИХ АЛГОРИТМАХ СИНХРОНІЗАЦІЇ

М.О. Волк

У роботі представлено розвиток е формального опису розподілених імітаційних моделей на основі їхнього програмного представлення в термінах процесної алгебри. Уводиться множина активностей журналізації станів моделі, що складають основу менеджера пам'яті імітаційних моделей. Показано, що реалізовані активності можуть виконувати функції федератів в оптимістичних алгоритмах синхронізації розподілених імітаційних моделей.

Ключові слова: *розподілені імітаційні моделі, процесна алгебра, стан моделі, дані, активність, менеджер пам'яті, оптимістичні алгоритми синхронізації.*

JOURNALIZING OF STATES OF THE PROGRAM DISTRIBUTED SIMULATION MODELS AND ITS USAGE IN OPTIMISTIC ALGORITHMS OF SYNCHRONISATION

M.A. Volk

In article the development of the formal description of the distributed simulation models on the basis of their program representation in terms of process algebra is presented. The set journalizing activities of states of model which make a basis of the memory manager of simulation models is entered. It is shown that realized activity can fulfill functions federates in optimistic algorithms of synchronisation of the distributed simulation models.

Keywords: *up-diffused simulation models, process algebra, state of model, information, activity, manager of memory, optimistic algorithms of synchronization.*