УДК 681.324

М.А. ВОЛК**,** канд. техн. наук, доц. ХНУРЭ (г. Харьков)

ПРОЦЕССНОЕ ПРЕДСТАВЛЕНИЕ СОСТОЯНИЙ РАСПРЕДЕЛЕННЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ С УЧЕТОМ СПЕЦИФИКИ ИХ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Рассмотрен формальный аппарат описания распределенных имитационных моделей на основе процессной алгебры и структурной реализации программного представления моделей. Приводится структура распределенной имитационной модели, в которой выделяются программные данные модели как средство представления ее состояний; вводится множество активностей, управляющих данными имитационной модели.

Ключевые слова: распределенные имитационные модели, процессная алгебра, данные модели, активность.

Постановка проблемы. В настоящее время нет достаточно эффективного формального аппарата описания распределенных имитационных моделей, систем моделирования и алгоритмов их взаимодействия. Существующие имеют общий недостаток, заключающийся в высоком уровне абстракций, который не позволяет достаточно формализовать процесс построения моделирующей среды с целью создания распределенных моделей и автоматизации управления ими.

Анализ литературы. Наиболее известными способами описания процессов в таких системах являются структурно-алгоритмический [1], на основе процессной алгебры [2 – 4], объектно-ориентированный, например, на основе стандарта HLA (High Level Architecture) [5, 6]. Все они обладают недостатками, сформулированными выше.

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 (например, системы транзактного моделирования, такой как GPSS [7]).

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.

Цель статьи. В данной статье предлагается формальный аппарат представления имитационных моделей на основе процессной алгебры [2, 3], который позволяет представить вычислительные процессы, протекающие в имитационной системе моделирования, с естественной (программной) точки

зрения. Он позволяет описывать не только элементы имитационной среды, но и учитывать специфику операционной системы, аппаратных ресурсов и других компонентов, участвующих в процессе эксперимента. Основной идеей предлагаемого аппарата является единое представление таких важных понятий имитационных моделей как состояние, модельное время, входные и выходные переменные, в качестве данных программы.

Структура имитационной модели. Введем понятие распределенной имитационной модели как некоторой программы (совокупности программ), исполняемой на вычислительном ресурсе:

$$\begin{cases} ИМ = \{ИМ_i,\ i = \overline{1,I}\}, \\ ИМ_i = \{dm_i,\ A_i,\ i = \overline{1,I}\}, \\ A_i\ = \{A_i^j,\ j = \overline{1,J}\}, \end{cases} \qquad (1)$$

где $ИМ_i$ – частные имитационные модели; I – количество частных имитационных моделей в системе; dm_i – данные модели (или, с точки зрения программы, данные программы); A_i – множество из J активностей (подпрограмм), обслуживающих i-ю имитационную модель.

В случае монолитной имитационной модели (I = 1) ее декомпозиция формальными методами невозможна. В остальных случаях предполагается возможность исполнения любой частной модели $ИМ_i$ на отдельном ресурсе.

Основываясь на особенностях программного представления модели, будем утверждать, что состояние модели полностью определяется содержимым данных модели (программы), а изменение состояния модели заключается в изменении данных. Изменение данных модели могут производить два субъекта вычислительного процесса: управляющая программа моделирования (УПМ) и активности самой модели A_i. Построение систем, в которых данные одной частной модели непосредственно изменяются другой, рассматривать не будем. Также не будем рассматривать модели, изменяемые под воздействием операционной системы и других программ. Такие модели очень редки и практически не используются.

Важными понятиями для любой имитационной модели являются событие (event) *e*, которое приводит, в общем случае, к изменению состояния модели, и процесс *P*, который является ответом на это событие. В обозначениях процессной алгебры [1] последовательность *событие→процесс* обозначается как *e→P* (за событием *e* следует выполнение процесса *P*). В нашем случае непосредственно выполнение процесса реализуется активностями, таким образом, $P = \{A_i\}$.

Большинство расширений процессной алгебры включает понятие *состояния s* объекта (модели, программы, процесса) и понятие перехода

объекта из одного состояния (i) в другое (j): $s_i \xrightarrow{P} s_j$. Осуществление перехода происходит под воздействием какого-либо процесса P.

При помощи перечисленных понятий возможно представление динамических особенностей поведения сложных параллельных систем. Однако, с точки зрения практики, важно не столько отразить смысловую функциональность процессов, протекающих в моделируемой системе (это произойдет автоматически), сколько уделить внимание реализации процессов в терминах реальной среды – программного обеспечения. В связи с этим введем еще дополнительные обозначения.

Состояние модели s и события e будем непосредственно связывать с данными модели dm: $s,e \subset dm$. Действительно, любая программа (в том числе и программная модель) определяется как совокупность данных и кода, который эти данные изменяет в процессе исполнения программы (моделирования). В этом случае понятия *изменения состояния модели* и *изменения данных модели* будем считать эквивалентными. Или, другими словами, можно сказать, что данные модели являются физической реализацией абстрактного понятия состояния модели.

Определим, что состояние и данные модели являются разными понятиями с точки зрения аналитики моделируемой системы. Так, любая система в одном и том же состоянии может иметь разные входные и выходные данные, что отражается на данных модели. Таким образом, понятие данных модели несколько шире и включает в себя как данные, отвечающие за состояние модели, так и различные переменные, отвечающие, например, за связь с системой моделирования, журнализацию событий и др. Термин эквивалентности лучше употреблять в сочетании *состояние* ***программной*** *модели*. В этом случае понятно, что состояние программы однозначно определяется совокупностью всех переменных (данных) и текущей исполняемой командой. Любые изменения, происходящие в модели, приводят к изменению данных под влиянием одной из активностей:

$$dm \xrightarrow{A_i} dm' .$$

Описанное выше представление открывает широкие возможности для реализации имитационной модели в физической программной среде.

Локальные и глобальные данные модели. Элементами имитационной модели являются активности A_i и данные (dm_i) (см. рис. 1). Активности A_i осуществляют функционирование на основе данных модели. Выделяя среди всей совокупности данных dm_i те, которые доступны только из самой частной модели ИМ$_i$, мы приходим к понятию локальных данных модели $dm_i^{\text{л}}$. Управляющая программа моделирования

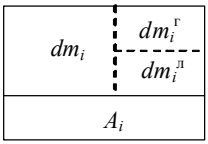

Рис. 1. Структура частной имитационной модели

(УПМ) выделяет требуемую область памяти данных для активности A_i и передает все полномочия по ее использованию алгоритмам *i*-й подмодели. В случае, если все операции с $dm_i^{л}$ осуществляются исключительно кодом активностей из множества A_i, мы имеем дело с *обычными локальными данными модели*. Отнесем к этому же классу данных те данные, которые может использовать УПМ только для функций чтения, так как они не изменяют состояния частной модели. Обычные $dm_i^{л}$ размещаются совместно с кодом активности, реализуя, таким образом, свойство инкапсуляции [7].

В результате декомпозиции системы, в модели могут оказаться две функционально одинаковые активности (две частные модели одного и того же объекта). В этом случае, формально должны быть сформированы два множества активностей A_i и A_j. Алгоритмы функционирования двух активностей эквивалентны друг другу, а конкретное состояние активности однозначно определяется множеством переменных состояния, которые и образуют локальные данные модели. Таким образом, наличие эквивалентных по функциональному назначению активностей позволяет нам реализовать возможность использования одной и той же активности для моделирования двух (и более) объектов. Однако, в этом случае, обе имитационные подмодели должны находиться на одном ресурсе. A_i и A_j, используют разные массивы dm_i и dm_j (см. рис. 2). Использование такого механизма приводит нас к *распределенным локальным данным модели*. Если эквивалентные активности моделируются на одном ресурсе (т.е. последовательно в рамках этого ресурса), для повышения эффективности его использования целесообразно хранить только одну копию алгоритма A_i и коллекцию данных $\bigcup_i dm_i^{л}$, где i – соответствует числу активностей данного функционального типа. Если же есть возможность исполнять данные активности параллельно на нескольких ресурсах, то УПМ может создать несколько экземпляров A_i, число которых, в общем случае, не равно числу моделируемых эквивалентных активностей A_i, реализуя, тем самым, механизм распределенных локальных данных модели.

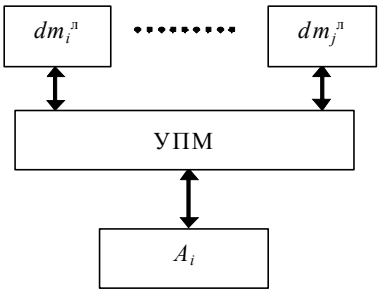

Рис. 2. Использование одного алгоритма для моделирования одинаковых активностей

Во многих случаях при построении систем моделирования необходимо учитывать специфику управления локальными данными модели $dm_i^{л}$. При моделировании сложных систем в конкретных режимах, отдельные элементы системы могут не менять своего состояния на протяжении значительного

интервала времени $t_э$. В этом случае $dm_i^л$, соответствующие долгое время неисполняемым активностям, могут быть временно выгружаемыми из оперативной памяти, реализуя механизм *виртуальных локальных данных модели*. Данный механизм может быть реализован на основе распространенных способов управления виртуальной памятью, широко применяемых в операционных системах [9]. В общем случае, каждому блоку памяти $dm_i^л$ ставится в соответствие идентификатор активности, несущий информацию либо о частоте использования $dm_i^л$, либо о времени последнего обращения к памяти. Те $dm_i^л$, обращение к которым происходит редко, могут быть выгружены из оперативной во внешнюю память. Эти же рассуждения можно перенести и на редко используемые активности. Современные операционные системы, как правило, реализуют свои механизмы управления виртуальной памятью, поэтому указанные свойства реализуются автоматически.

Очевидно, что частная модель любого элемента системы должна взаимодействовать с другими частными моделями. Иначе этот элемент может быть удален из системы без ущерба ее функционирования. Каким же образом осуществляется эта связь? Посредством изменения данных модели. Данные модели, которые могут быть изменены активностями, не входящими в активности частной модели, будем называть *глобальными данными модели* $dm_i^г$.

Аналогично локальным данным, в системе моделирования для глобальных данных могут быть использованы механизмы распределенных и виртуальных данных. Кроме того, система моделирования (УПМ) должна выполнять функции по диспетчеризации, синхронизации, ограничению доступа к таким данным. На рис. 3 представлена схема организации обращения к глобальным и локальным данным модели, где модуль ОС соответствует функциям операционной системы по обеспечению доступа к памяти.

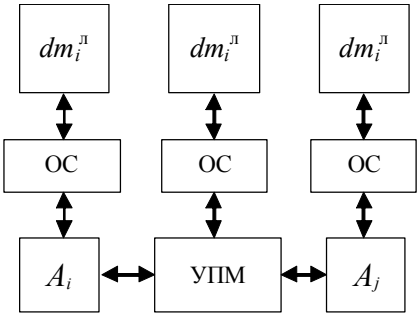

Рис. 3. Схема обращения к локальным и глобальным данным

Если допускается непосредственное обращение к памяти, то доступ к глобальным данным осуществляется средствами управляющей программы моделирования.

В условиях распределенных ресурсов использование глобальной памяти данных усложняется из-за следующих причин:

– возможно одновременное обращение к памяти со стороны нескольких активностей и УПМ;

– данные, обращение к которым происходит из активности, расположенной на одном из ресурсов, могут находиться на другом ресурсе;

– порядок и момент обращения к памяти со стороны активностей, в общем случае, носит стохастический характер;

– модификация данных, выполняемая одной из активностей, в некоторых случаях не должна влиять на функционирование других активностей.

Наиболее рациональным является сосредоточение функций управления глобальными данными в рамках одного ресурса (хотя сами данные могут быть распределены между ресурсами). Целесообразно размещать механизм глобальных данных совместно с главной УПМ, хотя возможен вариант выделения глобальным данным отдельного ресурса.

В процессе своего функционирования, активность A_i обращается к глобальным данным в целях реализации одного из возможных действий: чтения либо записи. Со стороны A_i локальные и глобальные данные одинаковы. Их различие воспринимается системой моделирования при компиляции модели, для чего в языке описания модели должны быть предусмотрены соответствующие синтаксические конструкции. УПМ$_i$ удаленной системы моделирования отслеживает данное обращение к глобальной памяти и пересылает запрос главной УПМ. Послав запрос, удаленная УПМ$_i$ приостанавливает выполнение алгоритма A_i-й активности до получения ответа о доступе к данным, и может в это время перейти к выполнению алгоритма активности A_j.

Запрос на обращение к глобальной памяти поступает в очередь обращений главной УПМ. Главная УПМ представляет собой последовательную машину и в определенный момент времени может модифицировать только один из модулей глобальной памяти, таким образом, исключая одновременную модификацию данных несколькими активностями. Диспетчер памяти выбирает из очереди обращений очередной запрос и обращается уже непосредственно к модулям глобальной памяти.

Конфликты взаимосвязи моделей. При реализации описанной организации имитационного моделирования возможны конфликты. Рассмотрим такой пример (рис. 4). Пусть в имитационной модели ИМ выделены две частные подмодели ИМ$_1$ и ИМ$_2$; выход подмодели ИМ$_1$ является входом ИМ$_2$. Реализация связи ИМ$_1$ и ИМ$_2$ осуществляется путем хранения в глобальной памяти данных значения $dm^{\Gamma}_{\text{ИМ}_1\text{ИМ}_2}$. При изменении внутреннего

состояния, одна из активностей A_1^j, соответствующая $ИМ_1$, меняет значение $dm^{г}_{ИМ_1ИМ_2}$. $ИМ_2$ в процессе своего функционирования отслеживает значения входа $dm^{г}_{ИМ_1ИМ_2}$ для реализации ветвления алгоритма.

Рассмотрим два момента времени t_k и t_l ($t_k > t_l$). Пусть в t_k активна $ИМ_1$, а в момент времени t_l активны $ИМ_1$ и $ИМ_2$, причем $ИМ_1$ меняет значение $dm^{г}_{ИМ_1ИМ_2}$ в оба момента времени, а $ИМ_2$ анализирует состояние $dm^{г}_{ИМ_1ИМ_2}$ только в момент времени t_l. Вследствие того, что со стороны системы моделирования порядок выполнение активностей, в общем случае, является случайным, $ИМ_2$ может с равной вероятностью прочитать из памяти как значения $dm^{г}_{ИМ_1ИМ_2}(t_k)$ так и $dm^{г}_{ИМ_1ИМ_2}(t_l)$. Такая ситуация порождает неопределенность, которая в большинстве реальных случаев недопустима вследствие того, что развитие процесса в моделируемых системах является детерминированным. Решением подобного конфликта является инициализация моделью или УПМ одной из активностей $ИМ_2$, либо помещение в очередь событий нового события с текущим временем t_k.

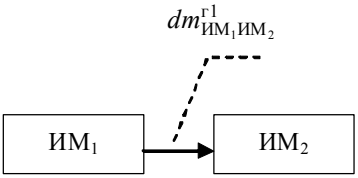

Рис. 4. Взаимодействие компонент типа
"Выход-вход"

Подобных конфликтных ситуаций в системе с распределенными глобальными данными может быть множество. Для их разрешения предлагается выделить в системе моделирования активность разрешения конфликтов. Разрешение наиболее распространенных конфликтов можно реализовать непосредственно в УПМ. Например, для рассмотренного выше примера можно предложить следующую схему. Выделить в глобальной памяти две копии $dm^{г}_{ИМ_1ИМ_2}$, причем одну из них использовать для чтения, а другую для записи при условии $t_M = const$. При изменении t_M производить копирование нового значения в блок памяти $dm^{г}_{ИМ_1ИМ_2}$, предназначенный для чтения.

Однако, в системе моделирования при оперировании глобальными данными могут возникать конфликты, не предусмотренные алгоритмами

УПМ. Например, в ситуации (см. рис. 5), когда подмодели ($ИМ_1$ и $ИМ_2$) имеют общий выход и обе одновременно (с точки зрения модельного времени) записывают в $dm^{г}_{ИМ_1ИМ_2}$ новые значения.

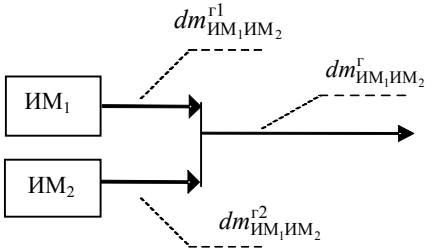

Рис. 5. Взаимодействие компонент типа "общий выход"

В большинстве систем моделирования новое значение $dm^{г}_{ИМ_1ИМ_2}$ определяется тем, какая из двух активностей выполнилась последней. Но общее правило может не удовлетворять логике работы модели в конкретной ситуации. С другой стороны, разрешение подобного конфликта может быть очевидным только со стороны разработчика модели. Поэтому предлагается в УПМ ввести возможность создания средств разрешения конфликтов пользователем. Примером такого механизма выступает функция разрешения шины в языке VHDL [10]. В рассмотренном нами примере действия системы моделирования могут быть следующими. Диспетчер памяти сохраняет оба значения, а по окончанию временного цикла (t_m = *const*), передает оба значения "функции разрешения конфликта пользователя", которая и определяет новое значение $dm^{г}_{ИМ_1ИМ_2}$. Активность, реализующую подобную функцию в системе моделирования будем обозначать $A_{конфл}$. Заметим, что подобная активность может быть реализована на трех уровнях: на уровне частных моделей, на уровне имитационной модели или на уровне УПМ.

Журнализация изменения данных модели. Возможность перехода модели из состояния в состояние непосредственно связана с изменением данных модели. Возможность возврата в одно из прошлых состояний модели реализуется путем сохранения данных модели в определенные моменты времени в прошлом, что можно осуществить одним из двух способов:

1. Сохранять все данные модели (дамп памяти модели) в моменты продвижения модельного времени. Данный способ дает надежный механизм возврата в любой момент в прошлом, для которого существует дамп памяти. Однако, этот 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.

2. Запоминание изменения 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. Для того, чтобы вернуть модель в прошлое, достаточно пройтись по цепочке в обратном порядке.

Оба этих способа составляют смысл процесса журнализации состояний имитационной модели.

Введем несколько новых активностей:

$A_{\text{дамп}}(t_k)$ – активность, позволяющая модели запомнить все свои данные в момент времени t_k.

$A_{\text{фикс}}(dm_i,\ t_k)$ – активность, фиксирующая изменение значения элемента данных dm_i в момент времени t_k.

Алгоритм журнализации планируется представить в виде отдельной работы.

Поведение частной модели. Любая частная модель должна содержать в себе активность, отражающую поведение объекта в текущий момент времени. В различных концепциях построения программной модели таких активностей (или функций программы) может быть несколько, каждая из которых вызывается УПМ в зависимости от состояния модели и входящих данных. Однако, в любом случае возможна реализация поведенческой активности в виде одного программного модуля (функции), который может в дальнейшем вызывать другие функции программы. Поясним это утверждение примером.

Пусть, программная модель содержит две активности A_1 и A_2, выполнение которых зависит от входного сигнала x, $x \in \{x_1, x_2\}$:

$$\textbf{if } x = x_1 \textbf{ then } A_1 \textbf{ else if } x = x_2 \textbf{ then } A_2.$$

В этом случае всегда можно реализовать активность A_3, которая и реализует приведенное условие:

$$A_3 \to \{x_1\, A_1 \mid x_2\, A_2\}.$$

Если вспомнить, что активность – это всего лишь функция программы, то реализация такой активности очевидна. Таким же образом можно доказать возможность объединения в одну активность нескольких, инициализирующихся в зависимости от состояния (внутренних данных частной модели), а также цепочки нескольких активностей одной модели.

Будем считать, что в частной модели присутствует одна поведенческая активность $A_{\text{пов}}$, которая моделирует поведение объекта:

$$dm \xrightarrow{A_{\text{пов}}} dm'.$$

Активность $A_{\text{пов}}$ изменяет данные модели в зафиксированный момент модельного времени.

Выводы. В данной работе была проанализирована одна из основных составляющих программных распределенных имитационных моделей – данные. Показано, что данные являются тем элементом имитационной модели, который полностью отражает ее состояние. Рассмотрены основные виды данных модели, множество операции над ними. Проведен анализ активностей, которые должна содержать программная имитационная модель для управления данными.

В дальнейшем планируется создать функционально полный набор активностей для работы с данными модели, активностей УПМ и представить в терминах процессной алгебры проведение самого имитационного моделирования.

Список литературы: **1.** *Максимей И.В.* Имитационное моделирование на ЭВМ. – М.: Радио и связь, 1988. – 222 с. **2**. *Хоар Ч.* Взаимодействующие последовательные процессы. – М.: Мир, 1989.– 264 с. **3.** A Calculus for Communicating Systems, LNCS92, 1980. – 171 p. **4.** The Space and Motion of Communicating Agents, to appear, Cambridge University Press, 2009. – 200 p. **5.** IEEE STD 1278.1-1995. IEEE Standard for Distributed Interactive Simulation – Application Protocols. N.Y.: Institute of Electrical and Electronics Engineers, Inc., 1995. **6.** *Allen R., Garlan D., Ivers J.* Formal modeling and analysis of the HLA component integration standard / Proc. of the 6th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering, 1998. – P. 70-79. **7.** *Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, Jim Conallen, Kelli A. Houston*. Object-Oriented Analysis and Design with Applications (3rd Edition). – Addison-Wesley Professional, 2007. – 720 p. **8.** *Шрайбер Т. Дж.* Моделювання на GPSS. – М.: Машинобудування, 1980. – 592 с. **9.** *William Stallings*. Operating Systems: Internals and Design Principles, 8th Editions. – Prentice Hall, 2008. – 800 p. **10.** *Суворова Е.А., Шейнин Ю.Е.* Проектирование цифровых систем на VHDL. – СПб.: БХВ, 2003. – 576 с.

УДК 681.324

Представлення станів розподілених імітаційних моделей у вигляді процесів з урахуванням специфіки програмної реалізації / Волк М.О. // Вісник НТУ "ХПІ". Тематичний випуск: Інформатика і моделювання. – Харків: НТУ "ХПІ". – 2009. – № 13. – С. 23 – 32.

Розглянуто формальний апарат опису розподілених імітаційних моделей на основі процесної алгебри і структурної реалізації програмного представлення моделей. Приводиться структура розподіленої імітаційної моделі, у якій виділяються програмні дані моделі як засіб представлення її станів; уводиться множина активностей, керуючих даними імітаційної моделі. Іл.: 5. Бібліогр.: 10 назв.

Ключові слова: розподілені імітаційні моделі, процесна алгебра, дані моделі, активність.

UDC 681.324

Process algebra for state description of distributed simulation model with program realization / Volk M.O. // Herald of the National Techical University "KhPI". Subject issue: Information Science and Modelling. – Kharkov: NTU "KhPI". – 2009. – №. 13. – P. 23 – 32.

The formal description of the distributed simulation models on the basis of process algebra and structural implementation of program representation of models is considered. The structure of the distributed simulation model in which the program given models as a resource of representation of its states are selected is resulted; the set activities of manage the simulation model data is entered. Figs: 5. Refs: 10 titles.

Key words: distributed simulation model, process algebra, model data, activity.

Поступила в редакцию 21.04.2009