ПРЕДСТАВЛЕНИЕ ВРЕМЕНИ В ИМИТАЦИОННОМ МОДЕЛИРОВАНИИ

В. В. Окольнишников

Институт вычислительной математики и математической геофизики СО РАН, Новосибирск, Россия

e-mail: okoln@kti.nsc.ru

This paper describes time management mechanisms for various types of imitational modeling. Conservative and optimistic time management algorithms are presented for distributed imitational modeling. Time management in the High Level Architecture is considered.

Введение

Имитационное моделирование (ИМ) является наиболее мощным, а иногда единственным методом исследования динамического поведения сложных систем. Имитационная модель воспроизводит поведение моделируемой системы во времени.

В ИМ имеются три понятия времени: физическое, модельное и процессорное. Физическое время относится к моделируемой системе. Модельное время — воспроизведение физического времени в модели. Процессорным временем в этой статье называется время выполнения имитационной модели на компьютере.

Соотношение физического и модельного времени определяется спецификой моделируемой системы и задается коэффициентом K. K — это диапазон физического времени, принимаемого за единицу модельного времени. При таком соглашении значению физического времени $T_{\text{ф}}$ соответствует значение модельного времени $T_{\text{м}}$, вычисляемое по простой формуле:

$$T_{\text{м}} = (T_{\text{ф}} - T_{\text{фн}})/K,$$

где $T_{\text{фн}} \le T_{\text{ф}} \le T_{\text{фк}}$, $T_{\text{фн}}$ — начальное, а $T_{\text{фк}}$ — конечное время моделирования.

Сущностью ИМ является продвижение модельного времени при выполнении модели от 0 до $(T_{\text{фк}} - T_{\text{фн}})/K$ единиц модельного времени и выполнение событий, связанных с определенными значениями модельного времени. Событие в модели — это программный модельный образ значимого, с точки зрения разработчика модели, изменения в моделируемой системе.

Основной задачей ИМ является правильное отображение порядка и временных отношений между изменениями в моделируемой системе на порядок выполнения событий в модели. Если не рассматривать "одновременные" события, т. е. выполняющиеся в один и тот же момент модельного времени, то это требование правильного отображения порядка

изменений в моделируемой системе означает, что события в модели должны выполняться в хронологическом порядке в модельном времени.

Процессорное время выполнения модели зависит от частоты изменений в моделируемой системе за моделируемый период физического времени, а также от объема вычислений, связанных с выполнением событий. Процессорное время явно не связано с модельным временем, кроме специальных приложений, рассматриваемых ниже.

Моделирование сложных систем может потребовать значительных затрат процессорного времени. Поэтому другой задачей ИМ является уменьшение процессорного времени. Это может быть достигнуто за счет параллельного исполнения событий на мультипроцессорной технике.

В некоторых приложениях требуется, наоборот, искусственное увеличение процессорного времени, например, для моделей-тренажеров. Это требуется для того, чтобы приблизить виртуальную среду, генерируемую компьютером, к реальной.

Таким образом, управление временем в системах ИМ является наиболее интересной, сложной, наукоемкой проблемой. В статье приводится обзор методов решения этой проблемы для последовательного ИМ (разд. 1) и распределенного (параллельного) ИМ (разд. 2). В разд. 3 описывается управление временем в серии стандартов IEEE P1516, более известном как HLA (High Level Architecture).

1. Последовательное имитационное моделирование

Имитационное моделирование как ветвь программирования прошло более чем 40-летний путь развития. Парадигмы ИМ, основные подходы, терминология, наиболее известные языки моделирования сложились в первые 20 лет. Со всем разнообразием ИМ можно познакомиться в классических трудах по ИМ [1 – 6]. За это время по ИМ было издано множество книг и опубликовано множество работ. Одной из последних изданных книг является [7]. Далее будет рассматриваться одно из основных направлений имитационного моделирования — дискретное ИМ.

При этом подходе модель состоит из множества объектов (процессов). Каждый объект моделирует какую-нибудь функцию или элемент декомпозиции моделируемой системы. Каждый объект имеет набор атрибутов и методов (событий).

Отличие ИМ от объектно-ориентированного программирования заключается в том, что объект может не только выполнить некоторое событие в момент своей активности, но и запланировать выполнение своего события или события другого объекта “в будущем”, т. е. на момент модельного времени, больший или равный текущему значению модельного времени.

Для реализации выполнения будущих событий требуется дополнительная программа, которая выполняет функцию планировщика для организации выполнения событий различных объектов в хронологическом порядке. Такая управляющая программа является необходимой частью системы исполнения (run time system) любой системы моделирования. Типовая структура управляющей программы и взаимосвязь управляющей программы с объектами представлены на рис. 1.

Управляющая программа взаимодействует с объектами в соответствии с клиент/серверным подходом. Алгоритм работы управляющей программы состоит из следующих действий:

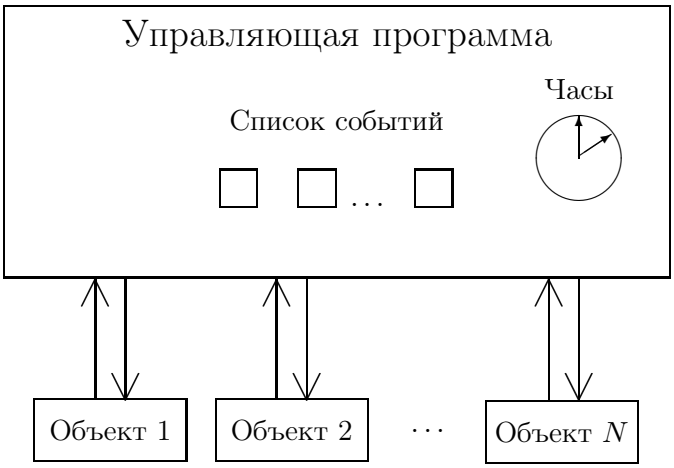


Рис. 1. Схема выполнения последовательной модели.

1) активизация объектов для выполнения событий, запланированных на текущее значение часов модельного времени. Удаление выполненных событий из списка;

2) включение в список событий новых событий, запланированных активными объектами. Событие включается в список событий вместе со значением модельного времени, в которое это событие должно быть выполнено в будущем;

3) увеличение значения часов модельного времени, если на текущее значение часов модельного времени не осталось невыполненных событий. Переход на п. 1.

При такой организации выполнения модели события в модели выполняются последовательно, что и нашло отражение в термине “последовательное ИМ”. Последовательное ИМ характерно для выполнения модели на автономном однопроцессорном компьютере.

Вместе с тем несколько событий в разных объектах могут выполняться в один и тот же момент модельного времени, поэтому говорят, что объекты (процессы) выполняются квазипараллельно.

При выполнении модели модельное время изменяется скачкообразно. На оси модельного времени моменты исполнения событий составляют дискретное множество. Поэтому такое моделирование называется дискретным ИМ или ИМ с дискретными событиями.

Имеется несколько алгоритмов продвижения модельного времени в дискретном ИМ. Эти алгоритмы в разного рода модификациях и сочетаниях реализованы в системах дискретного ИМ. Каждый алгоритм наиболее эффективен для определенного класса приложений. Основными алгоритмами управления временем являются:

1. Моделирование, управляемое событиями (Event driven). Наиболее распространенная реализация, при которой в качестве следующего значения часов модельного времени выбирается минимальное время событий из списка событий. Для птимизации этого алгоритма обычно список событий упорядочивается в порядке возрастания значений модельного времени, в которые события должны быть выполнены.

2. Моделирование с фиксированным шагом (Time stepped). При таком моделировании значение часов модельного времени каждый раз увеличивается на фиксированную величину. Такой подход удобен при наличии условных событий, т. е. событий, для выполнения которых требуется истинность некоторого логического условия. На каждом шаге можно вычислять логические условия и выполнять события. При управляемом событиями

моделировании можно “перескочить” момент модельного времени, при котором условие истинно. Эта реализация более проста, но уступает предыдущей в эффективности.

3. Моделирование, управляемое часами реального времени (Wallclock time driven). При таком моделировании значение часов модельного времени определяется некоторой неубывающей функцией от значений аппаратно или программно реализованных часов реального времени. Такие имитационные модели обычно связаны либо с аппаратурой, либо с людьми. Примером последних являются тренажеры, модельное время для которых определяется линейной функцией от реального времени.

Характерным признаком последовательного ИМ является наличие централизованного списка событий и глобальных часов модельного времени.

2. Распределенное имитационное моделирование

2.1. Понятие распределенного имитационного моделирования

Распределенное ИМ (Distributed simulation) имеет три источника своего развития: моделирование, требующее для своего выполнения большого количества вычислительных ресурсов, военные приложения и компьютерные игры с использованием Интернет. Под распределенным ИМ, которое в дальнейшем будет называться просто распределенным моделированием, понимается распределенное выполнение единой программы имитационной модели на мультипроцессорной или мультикомпьютерной системе.

Вычисления, требующие для своего выполнения большого количества вычислительных ресурсов. Если критическим ресурсом является процессорное время, то при использовании мультипроцессорной системы или суперкомпьютера с N процессорами можно теоретически ожидать убыстрение выполнения модели в N раз по сравнению с выполнением модели на одном процессоре.

При увеличении суммарной мощности таких систем становятся доступными решения новых задач, которые, в свою очередь, порождают новые потребности в вычислительных ресурсах. Примером масштабов таких задач является заявленная США потребность в машинном времени в количестве 2.7 млн часов для моделирования взрыва сверхновой звезды [8].

Другим примером использования распределенного моделирования являются имитационные модели большого масштаба (Large-Scale). Такие модели не могут быть исполнены на автономном компьютере из-за ограничений памяти, но могут быть исполнены на вычислительной системе с кластерной архитектурой.

Военные приложения. Целью использования распределенного моделирования для военных приложений является интеграция отдельно разработанных моделей в единое модельное окружение. Примером таких окружений могут быть тренажеры для обучения и имитации гипотетических сценариев военных действий.

Другим примером являются модели инфраструктур, объединяющие экономические, экологические, транспортные и другие подмодели. Такие подмодели могут исполняться на географически распределенных гетерогенных мультикомпьютерных системах. Централизация этих подмоделей может привести к недопустимо большому объему передачи данных из географически распределенных баз данных.

Компьютерные игры с использованием Интернет. Методология распределенного моделирования широко используется при реализации распределенных многопользова-

тельских компьютерных игр. Эти работы берут начало в использовании ИМ для создания анимационных фильмов. Успехи в машинной графике и создании реалистических виртуальных окружений в таких играх сделали более значимым и распределенное моделирование.

Исторически понятие распределенного моделирования относилось к моделированию на компьютерах с MIMD (Multipli Instruction Stream / Multipli Data Stream) архитектурой. Для моделирования на компьютерах с SIMD (Single Instruction Stream / Multipli Data Stream) архитектурой использовался термин "параллельное моделирование".

Поскольку при параллельном и распределенном моделировании используется одна и та же техника, с появлением кластерных вычислительных систем и Grid-вычислений грань между этими двумя видами ИМ начала стираться. В настоящее время многие авторы [9] используют термин "распределенное моделирование" для обозначения как параллельного, так и собственно распределенного моделирования.

2.2. Выполнение модели при распределенном моделировании

Последовательная имитационная модель может быть выполнена на параллельной вычислительной технике. Выигрыш по времени исполнения может быть достигнут за счет параллельного выполнения событий, запланированных на один и тот же момент модельного времени. Распределенное моделирование использует другую более общую форму параллелизма, а именно параллельное выполнение событий, запланированных в различных отрезках модельного времени.

При распределенном моделировании в отличие от последовательного моделирования первичной единицей является не объект, а так называемый логический процесс. Логический процесс — это последовательная подмодель, структура которой представлена на рис. 1. Каждый логический процесс имеет собственный набор объектов и собственную управляющую программу. Логический процесс имеет собственный локальный список событий и собственные часы локального модельного времени. Логические процессы взаимодействуют исключительно с помощью передачи сообщений.

Распределенную модель, состоящую из N логических процессов, можно определить следующим образом:

$$\begin{aligned}
\mathrm{DM} &= U_{k=1}^{N}\mathrm{lp}^k, \\
\mathrm{lp}^k &= \{\mathrm{sm}^k, T^k, S^k\}, \\
S^k &= \left\{ \left(Q_1^k, t_1^k\right), \left(Q_2^k, t_2^k\right), \ldots, \left(Q_m^k, t_m^k\right), \,\middle|\, t_1^k \le t_2^k \le \ldots \le t_m^k \right\}, \\
T_k &= t_1^k,
\end{aligned}$$

где DM — распределенная модель; lp — логический процесс; sm — подмодель логического процесса; T — значение локальных часов модельного времени логического процесса; S — локальный список событий логического процесса; Q — событие в списке событий; t — модельное время наступления события Q (временная метка события Q).

Глобальных (общих для всей распределенной модели) часов модельного времени и глобального списка событий явно не существует, так как наличие общей управляющей программы, работающей с этими глобальными структурами, было бы "бутылочным горлышком" для параллельного исполнения.

Текущее модельное время всей модели TIME в каждый момент равно:

$$\text{TIME} = \min_{k=1}^{N} T^k.$$

Каждый логический процесс выполняется в собственном модельном времени как автономная последовательная модель. Для этого он снабжен экземпляром управляющей программы. Схема выполнения распределенной модели представлена на рис. 2.

Логический процесс общается с другими процессами, передавая им сообщения. Сообщения могут быть разных видов. Рассмотрим простейший вид сообщения — асинхронное сообщение, никак не влияющее на выполнение логического процесса — отправителя.

Сообщение M_{ij}, передаваемое от логического процесса lp^i логическому процессу lp^j, в простейшем случае имеет следующую структуру:

$$M_{ij} = (Q^j, T^i).$$

Логический процесс lp^j, получив сообщение, которое имеет форму события, вставляет это событие в упорядоченный список своих локальных событий в соответствии со значением T^i. Управляющая программа начинает выполнять модифицированный список локальных событий. Таким образом, полученное сообщение может изменить логику выполнения логического процесса — получателя.

Передача сообщений может осуществляться логическими процессами непосредственно с помощью средств операционной системы. Но более перспективна схема, приведенная на рис. 2. Она включает коммуникационную подсистему в качестве отдельного компонента. Логические процессы взаимодействуют с коммуникационной подсистемой с помощью определенного интерфейса.

Использование коммуникационной подсистемы имеет следующие преимущества:

1. Вся системная часть реализации передачи сообщений сосредоточена в одной программе (коммуникационной подсистеме).

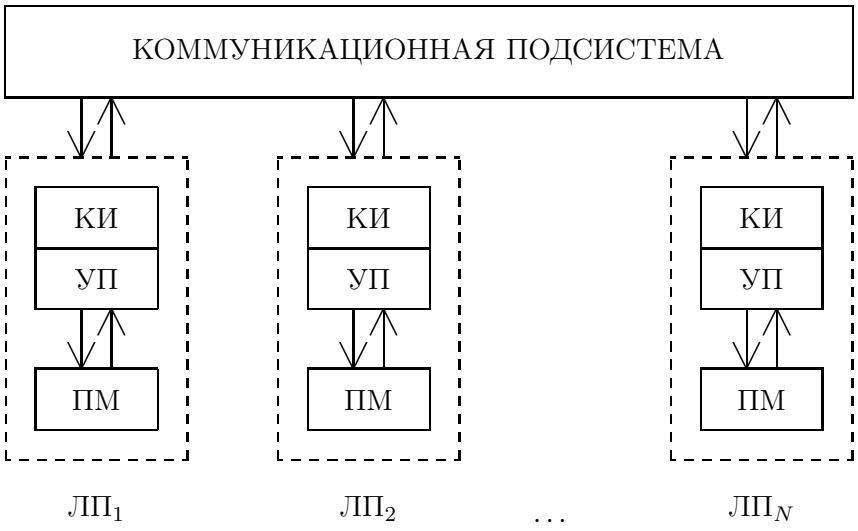


Рис. 2. Схема выполнения распределенной модели: ЛП — логический процесс, ПМ — подмодель, УП — управляющая программа, КИ — коммуникационный интерфейс.

2. Логические процессы могут быть реализованы на C++. Для переноса моделей и системы моделирования на другую вычислительную систему достаточно перенести коммуникационную подсистему. Перенос коммуникационной системы упрощается, если при ее реализации использованы стандартные средства, например передача сообщений реализована с помощью протокола MPI (Message passing interface) [10], который имеет реализации как для Windows, так и для параллельных компьютеров и гетерогенных кластеров.

3. Коммуникационная подсистема может синхронизировать выполнение логических процессов в модельном времени.

Необходимость синхронизации следует из примера. Рассмотрим фрагмент модели счастливого сюжета романа А.С. Пушкина "Дубровский" — сюжета, в котором переданное Машей кольцо попадает Дубровскому вовремя.

Логический процесс **Т** (Троекуров) посылает сообщение логическому процессу **М** (Маша). Логический процесс **М** должен послать ответное сообщение через ΔT единиц модельного времени. Логический процесс **М** посылает сообщение логическому процессу **Д** (Дубровский), возможный ответ которого влияет на ответ **М**.

Временная диаграмма исполнения модели представлена на рис. 3, *а*. Диаграмма отражает поведение логических процессов в модельном времени. Передача сообщений в распределенном моделировании происходит мгновенно в модельном времени. Если по логике моделируемой системы передача сообщений происходит с задержкой, то эта задержка явно имитируется в программе модели. Поэтому передача сообщений во временной диаграмме изображается вертикальными стрелками.

Последовательная реализация рассматриваемой модели приводит к временной диаграмме исполнения модели, изображенной на рис. 3, *а*, которая корректно моделирует счастливый сюжет.

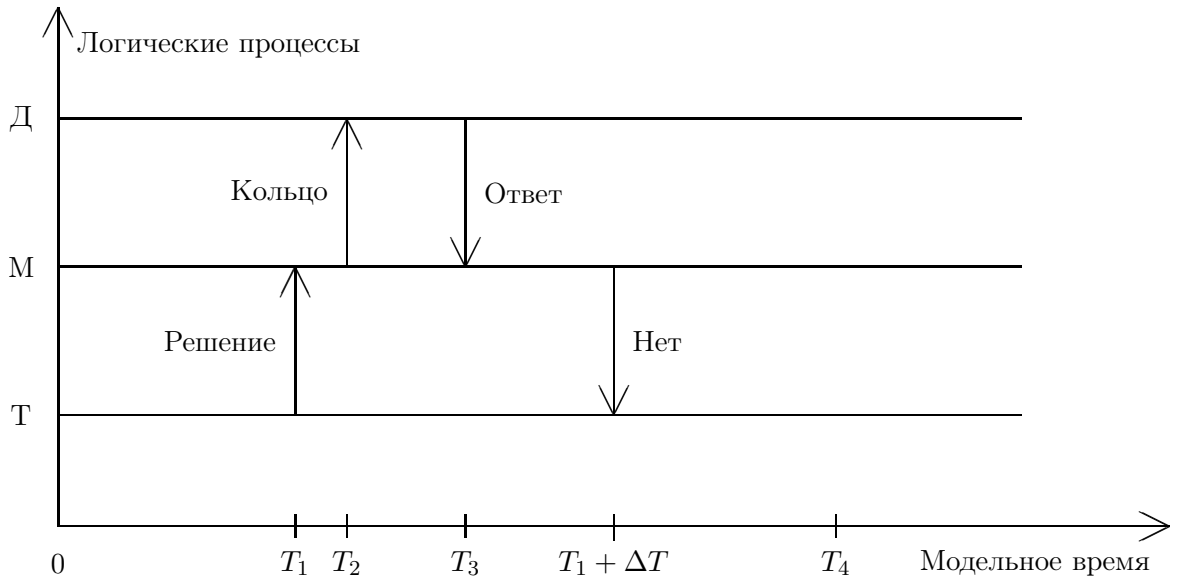
При распределенной реализации модельное время в разных логических процессах "движется" с разными скоростями. Выше уже отмечалось, что процессорное время не связано с модельным временем явно. Точнее, процессорное время является неубывающей функцией от модельного времени. Но для разных логических процессов эти функции разные. Поэтому через некоторый интервал астрономического времени от начала исполнения модели модельные времена разных логических процессов могут оказаться разными.

Сообщение, посланное логическим процессом **Д**, может быть получено логическим процессом **М**, как показано на рис. 3, *а* или как на рис. 3, *б*, если модельное время логического процесса **М** "убежало вперед".

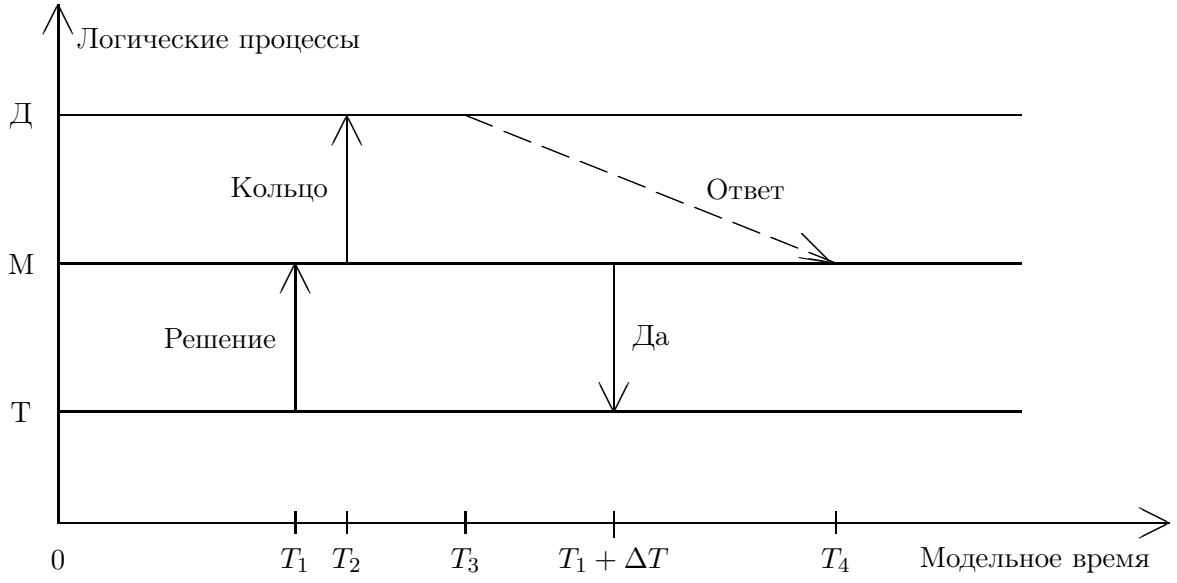
Во втором случае "опоздание" сообщения вызвано не физическими, а "техническими" причинами, например, процессор, на котором исполняется логический процесс **Д**, перегружен или для определения ответа требуется слишком большой объем вычислений.

Для логического процесса **М** полученное от **Д** сообщение в момент модельного времени T_4 означает изменение прошлого в момент T_3. Такая коллизия не может случиться в реальном мире, но часто встречается у писателей-фантастов, которые легко справляются с "парадоксом времени", заменяя существующую реальность на новую, возникшую в результате изменения прошлого.

Результат моделирования, представленный на рис. 3, *б*, означает нарушение главного критерия, сформулированного в начале статьи, чтобы модель правильно воспроизводила последовательность событий в моделируемой системе. Для распределенного моделирования этот критерий можно переформулировать следующим образом: результат исполнения распределенной модели должен совпадать с результатом исполнения этой модели на однопроцессорном компьютере.



а



б

Рис. 3. Временная диаграмма исполнения модели: *а* — корректная имитация; *б* — некорректная имитация.

Парадокс времени возникает при получении логическим процессом j от логического процесса i сообщения типа

$$M_{ij} = (Q^j, T^i) \;\Big|\; T^i < T^j.$$

Для борьбы с парадоксами времени в системе распределенного моделирования должна быть реализована специальная программа синхронизации. Эта программа входит в состав коммуникационной подсистемы. Программа синхронизации реализует некоторые абстрактные алгоритмы, которые получили название “алгоритмы синхронизации модельного времени”.

2.3. Алгоритмы синхронизации модельного времени

Целью механизма синхронизации модельного времени является выполнение каждым логическим процессом событий в порядке неубывания их временных меток. Это требование известно как локальное ограничение причинной связи (local causality constraint), так как оно обеспечивает имитацию естественного порядка "от причины — к следствию" [11].

Если не принимать во внимание порядок событий, имеющих одинаковые временные метки (одновременные события), то выполнение этого требования гарантирует, что выполнение программы модели на параллельном компьютере приведет точно к таким же результатам, что и выполнение модели на последовательном компьютере, где все события выполняются строго в порядке возрастания временных меток. Это также гарантирует, что повторное выполнение модели приведет к тем же результатам, если все входные данные совпадают и повторные вычисления, связанные с каждым событием, дают одинаковые результаты.

Алгоритмы синхронизации модельного времени делятся на два основных класса: консервативные и оптимистические. С более подробной классификацией алгоритмов синхронизации модельного времени можно познакомиться в [12 – 15].

2.3.1. Консервативные алгоритмы

Задачей консервативных алгоритмов является предотвращение парадоксов времени. Распределенное моделирование не накладывает никаких требований на систему передачи сообщений. Предполагается лишь, что получатель должен получать сообщения в том же порядке, в котором их посылает отправитель, и что отправитель выполняет свой список событий в порядке неубывания их временных меток. Это означает, что если логический процесс j получил от логического процесса i сообщения типа

$$M_{ij} = (Q^j, T^i) \;\Big|\; T^i > T^j,$$

то логический процесс j в интервале модельного времени $\Omega^j = (T^j, T^i)$ не получит другого сообщения от логического процесса i и может параллельно выполняться в этом интервале модельного времени, не "опасаясь" возникновения "парадокса времени" со стороны логического процесса i.

На этом факте базируются все консервативные алгоритмы синхронизации модельного времени. При этом следует учитывать все логические процессы распределенной модели. Например, одновременное с сообщением от i сообщение от логического процесса k

$$M_{kj} = (Q^j, T^k) \mid T^k > T^j$$

может привести к уменьшению интервала Ω^j до $\Omega^j_1 = (T^j, \min(T^i, T^k))$.

Если от некоторого логического процесса m вообще нет сообщения логическому процессу j, то для предотвращения "парадокса времени" интервал Ω^j следует уменьшить до пустого интервала $\Omega^j_2 = (T^j, T^j)$. Это означает, что логический процесс j во избежание возможного "парадокса времени" со стороны логического процесса m следует искусственно приостановить до получения сообщения от логического процесса m.

Приостановка выполнения логических процессов снижает эффективность использования параллелизма. Кроме снижения эффективности параллельного исполнения приостановка логического процесса до получения сообщения от другого логического процесса

может вызвать тупиковую ситуацию, когда ждущие логические процессы образуют замкнутый контур и останавливается исполнение всей распределенной модели.

Для уменьшения негативного влияния задержки выполнения логических процессов консервативные алгоритмы используют дополнительную информацию. Например, дополнительной информацией может быть информация, что логический процесс m завершился и, следовательно, не может вызвать "парадокс времени".

Другой дополнительной информацией может быть информация о связях логических процессов. Обозначим через C^j множество логических процессов распределенной модели, которые могут посылать сообщения логическому процессу j. В этом случае при вычислении интервала Ω^j логические процессы, не принадлежащие множеству C^j, можно не принимать во внимание, так как они не могут вызвать "парадокс времени" для логического процесса j.

Первые консервативные алгоритмы [16, 17] использовали механизм посылки "пустых" сообщений. Суть этого механизма сводится к следующему. Если логический процесс m, принадлежащий множеству C^j, при выполнении события (Q^m, T^m) не породил сообщение $M_{mj} = (Q^j, T^m)$ для логического процесса j, то посылается пустое сообщение $M_{mj} = (\emptyset, T^m)$.

При получении пустого сообщения логический процесс j не выполняет никаких действий. Цель посылки пустого сообщения заключается в передаче логическому процессу j дополнительной информации о состоянии логического процесса m для вычисления интервала Ω^j, безопасного для параллельного исполнения логического процесса j. Интервал Ω^j вычисляется следующим образом:

$$\Omega^j = (T^j, H^j),$$

где $H^j = \min\limits_{i \in C^j}(T^i)$.

Величина H^j является ключевым понятием консервативных алгоритмов. Далее она будет называться "временным горизонтом" (Logical Virtual Time Horizon) [12]. В [9] она имеет другое название — "нижняя граница временных меток" (LBTS — Lower Bound on the Time Stamp). События из списка событий логического процесса j с временной меткой, меньшей, чем H^j, безопасны для исполнения. Выполнение событий с временной меткой, большей, чем H^j, может привести к парадоксу времени. Безопасность выполнения событий с временной меткой, равной H^j, зависит от особенностей конкретного алгоритма. Эта проблема подробно исследуется в [18].

Консервативные алгоритмы, использующие механизм посылки пустых сообщений, основаны на вычислении для каждого логического процесса временного горизонта и параллельного исполнения логических процессов в пределах их временных горизонтов.

Временной горизонт вычисляется на основе временных меток поступающих сообщений, как содержательных, так и пустых. Внутренние события, запланированные логическим процессом самому себе, для общности оформляются как посылка процессом сообщений самому себе с временными метками, которые не используются при вычислении временного горизонта.

Если в пределах временного горизонта не имеется ни внешнего, ни внутреннего события, то логический процесс приостанавливается до поступления внешнего сообщения. Использование пустых сообщений не гарантирует отсутствия тупиковых ситуаций. Тупиковая ситуация возникает, если приостановленные логические процессы образуют замкнутый контур, а циркулирующие по нему пустые сообщения не вызывают "расширения" (увеличения) временного горизонта ни у одного логического процесса в контуре.

В работе [19] был предложен алгоритм распознавания тупиковых ситуаций. После обнаружения замкнутого контура, вызвавшего тупиковую ситуацию, логическому процессу с наименьшим временным горизонтом посылается служебное сообщение, расширяющее его горизонт. Это дает возможность выполнения им события с минимальной временной меткой и выхода из тупиковой ситуации. Выполнение события с минимальной временной меткой в модели всегда безопасно, так как не может вызвать парадокс времени.

Основным недостатком таких алгоритмов является большое количество пустых сообщений, так как при получении пустого сообщения логический процесс должен послать вторичные пустые сообщения с той же временной меткой другим логическим процессам.

Для уменьшения количества пустых сообщений в [20] был предложен алгоритм, в котором пустые сообщения выдаются не всегда, а только по специальному запросу. Приостановленный логический процесс посылает запрос всем логическим процессам, принадлежащим C^j, может ли он выполнить событие с минимальной временной меткой, находящейся за пределами временного горизонта.

Многие алгоритмы для определения безопасности выполнения следующих событий используют понятие "расстояние" (Distance) между логическими процессами. Расстояние — это минимальное количество модельного времени, которое должно пройти, прежде чем событие в одном логическом процессе вызовет прямо или косвенно выполнение события в другом логическом процессе.

Расстояния используются логическим процессом для определения границ временных меток возможных будущих сообщений от других логических процессов. Для этого требуется информация о том, какие логические процессы могут посылать сообщения другим логическим процессам. Алгоритмы, использующие расстояния, приведены в [21 – 23].

В рассмотренных алгоритмах использовалась дополнительная информация, полученная из анализа структуры модели. Наиболее перспективной является дополнительная информация, которую может сообщить определенным способом разработчик модели.

Если при выполнении события с временной меткой T логический процесс может гарантировать, что в промежутке $[T, T + L)$, где $L > 0$, не будет послано ни одно сообщение, то он может послать пустое сообщение не с временной меткой T, как обычно, а "досрочно" с временной меткой $T + L$. Величина L называется "предсказанием" (Lookahead). Наличие предсказания расширяет временные горизонты других логических процессов, что увеличивает эффективность параллельного исполнения.

Предположим, что логический процесс моделирует некоторое транспортное средство. Если по логике модели перемещение этого транспортного средства из пункта А в пункт Б является неделимым действием, то разработчик модели может оценить модельное время, требующееся для моделирования перемещения, и "сообщить" его системе моделирования. Эта величина является одним из примеров предсказания.

Другим примером может быть точность моделирования. Если известно, что все изменения в модели происходят с шагом, превышающим некоторый квант модельного времени, то этот квант может быть использован в качестве предсказания. Использование предсказаний исследуется, например, в [24, 25].

Выше подробно рассматривалось представление модельного времени в виде положительного вещественного числа. Для упорядочивания выполнения событий в логических процессах в консервативных алгоритмах используются и другие представления времени выполнения событий.

Например, в алгоритме Лампорта [11] упорядочивание одновременных событий осуществляется по паре (LT, N), где LT — локальное время процесса, а N — номер процесса.

Если более важен правильный порядок выполнения событий, чем абсолютные значения модельного времени событий, то в качестве значения локального модельного времени логического процесса lp^i может быть использован вектор $(m_1^i, m_2^i, \ldots, m_N^i)$, где N — число логических процессов. Здесь m_i^i — число событий, которые произошли с логическим процессом lp^i в текущий момент, а m_j^i $(j \neq i)$ — число событий, которые произошли с логическим процессом lp^j на момент посылки последнего сообщения (прямого или косвенного) от логического процесса lp^j логическому процессу lp^i.

Такой вектор называется векторной меткой времени (Vector Time). При посылке сообщения в качестве временной метки передается не скалярное значение, а вектор [26]. Если векторная временная метка одного события меньше (покомпонентное сравнение) векторной временной метки другого события, то первое событие является **причинно предшествующим** второму событию и должно быть выполнено раньше [27]. В [27] определяются также матричные метки времени (Matrix Time), которые являются развитием этого подхода.

Более полный обзор использования векторных меток времени приводится в [28]. Замена упорядочивания выполнения событий по временным меткам на упорядочивание выполнения событий по причинно-следственным отношениям рассматривается также в [29].

Другим направлением представления модельного времени является приближенное время (Approximate Time). Оно основано на временной нечеткости и использует временные диапазоны вместо точных отметок модельного времени. Диапазон представляет собой отрезок вещественных чисел $[FT, LT] = \{t | FT \leq t \leq LT\}$, где t — модельное время события; FT, LT — нижняя и верхняя границы диапазона соответственно [30]. Описания механизмов использования часов приближенного времени и упорядочивания выполнения событий логических процессов в приближенном времени можно найти в [31 – 33].

Еще одним подходом к синхронизации модельного времени в логических процессах является синхронизация модельного времени с реальным временем — часами реального времени в вычислительных средствах, на которых эти логические процессы выполняются. При этом возникает проблема синхронизации самих часов реального времени. Имеется множество алгоритмов синхронизации часов реального времени в распределенных системах [34, 35].

Для увеличения производительности модельное время в логическом процессе lp_i “убыстряется” с коэффициентом K_i по сравнению с реальным временем. Но скорости логических процессов, между которыми происходит передача сообщений, должны быть согласованы на момент передачи сообщения с тем, чтобы учесть реальное время прохождения сообщения по сети. Алгоритмы, основанные на динамическом изменении (адаптации) множества K_i, получили название адаптационных алгоритмов (Adaptive Time) [36, 37].

Наконец, имеется подход, при котором ослабляются требования строгой временной упорядоченности выполнения событий в логических процессах (Relaxing Time) [38]. При таком подходе парадоксы времени просто игнорируются, а за счет параллельного исполнения достигается высокая производительность.

При этом нарушается требование повторяемости и корректности результатов моделирования. Но для некоторых моделей возникновение единичных некорректностей “сглаживается” статистической обработкой результатов моделирования. Например, в [39] показано, что для определенного класса моделей (модели с очередями) игнорирование парадоксов времени уменьшает суммарное время моделирования в 3 раза, а точность моделирования всего на 2 % по сравнению с использованием моделирования со строгой синхронизацией. Такой подход актуален для сверхбольших (Ultra-Large-Scale) моделей, состоящих

из более чем миллиона компонентов, например моделей коммуникационных сетей [40].

Хотя рассмотренные методы распределенного моделирования применимы для произвольных гетерогенных вычислительных систем, использование Интернета для распределенного моделирования имеет свои специфику и преимущества [41].

Распределенная модель является частным случаем распределенной программной системы. Поэтому проблемы, имеющиеся в распределенном моделировании, могут быть решены с использованием методов, разработанных для распределенных систем. Обзор этих методов, а также принципы, концепции и технологии распределенных систем приведены в фундаментальной книге [42].

К проблемам распределенного моделирования относятся: запуск модели, синхронизация и окончание моделирования. В этой статье подробно обсуждаются проблемы синхронизации.

Корректный запуск модели заключается в синхронном начале работы всех логических процессов. Но поскольку для достижения состояния готовности разных логических процессов требуется разное процессорное время, возможны "фальстарты". Эта проблема может быть решена, например, с помощью барьерной синхронизации (Barrier Synchronization), используемой в распределенных системах. Многие операционные системы и программные средства поддерживают барьерную синхронизацию, например пакет MPI.

Моделирование заканчивается, если все логические процессы синхронно достигнут некоторого заданного значения модельного времени или в модели не останется будущих событий. Второе условие можно проверить, анализируя локальные состояния логических процессов. Проблема заключается в том, что некоторые сообщения посланы, но еще не получены.

В распределенных системах для решения этой проблемы используется понятие *глобальное состояние* (Global State). Глобальное состояние включает в себя локальные состояния каждого процесса вместе с находящимися в пути сообщениями [43]. Анализируя глобальное состояние, можно заключить, что либо модель зашла в тупик (Deadlock), либо модель корректно завершилась [44].

Методы, используемые в распределенном моделировании для определения глобального состояния, базируются на понятии *распределенного снимка состояния* (Distributed Snapshot) [45]. Обзор различных методов приведен в [46, 47].

К настоящему времени известно большое количество консервативных алгоритмов и их модификаций, используемых при моделировании определенных классов реальных систем. Критерием использования того или иного консервативного алгоритма для моделирования конкретных систем является отношение накладных расходов этого алгоритма к выигрышу, получаемому при параллельном исполнении.

2.3.2. Оптимистические алгоритмы

Если консервативные алгоритмы исключают даже потенциальную возможность возникновения парадокса времени, то оптимистические алгоритмы "надеются", что при параллельном исполнении логических процессов потенциальная возможность возникновения парадокса времени не станет реальностью.

В случае же возникновения парадокса времени оптимистические алгоритмы реализуют "откат" (rollback) логического процесса до значения модельного времени, в который ему было послано сообщение, вызвавшее парадокс времени. Откат включает в себя ликвида-

цию последствий некорректного исполнения логического процесса и повторное исполнение этого процесса с учетом сообщения, вызвавшего парадокс времени. Этот механизм получил название “деформации времени” (Time Warp Mechanism) [48].

Проиллюстрируем вышесказанное на примере, изображенном на рис. 3. Если исполнение модели происходит с использованием консервативного алгоритма синхронизации, то исполнение логического процесса **М** будет приостановлено в точке T_2, так как для вычисления временного горизонта логического процесса **М** требуется получение либо содержательного, либо пустого сообщения от логического процесса **Д**. Таким образом, будет реализован сценарий, изображенный на рис. 3, *а*.

Если исполнение модели происходит с использованием оптимистического алгоритма синхронизации, то логический процесс **М** не будет приостановлен в точке T_2, а в точке T_4 получит сообщение с временной меткой T_3, причем $T_3 < T_4$. При этом произойдут откат логического процесса **М** до момента модельного времени T_3 и повторное исполнение с этого момента модельного времени, которое пойдет уже по сценарию, изображенному на рис. 3, *а*.

Для реализации отката требуется решить две технические проблемы. Первая — это восстановление состояния логического процесса **М** в момент модельного времени T_3. Вторая — это ликвидация последствий некорректного выполнения логического процесса **М**, в частности ликвидация последствий сообщения логическому процессу **Т** в момент модельного времени $T_1 + \Delta T$. Первая проблема решается с помощью известной в программировании техники контрольных точек. Вторая — с помощью посылки специальных служебных сообщений — “антисообщений”. Содержанием антисообщения является информация о том, что такое-то посланное сообщение является недействительным.

Рассмотрим теперь действия логического процесса **Т**, получившего антисообщение, отменяющее сообщение “Да”, посланное в момент модельного времени $_1 + \Delta T$. Поскольку логический процесс **Т** исполняется параллельно, возможны следующие три ситуации:

1. Локальное время логического процесса **Т** меньше $T_1 + \Delta T$. Это самый простой вариант. Сообщение “Да” еще не вызвало никаких действий и может быть просто “аннигилировано” антисообщением.

2. Локальное время логического процесса **Т** больше $T_1 + \Delta T$, и с момента $T_1 + \Delta T$ логический процесс **Т** не послал ни одного сообщения. В этом случае происходит откат логического процесса **Т** до момента модельного времени $T_1 + \Delta T$.

3. Локальное время логического процесса **Т** больше $T_1 + \Delta T$, и с момента $T_1 + \Delta T$ логический процесс **Т** послал сообщения другим процессам. В этом случае происходят откат логического процесса **Т** до момента модельного времени $T_1 + \Delta T$ и посылка новых антисообщений этим другим процессам.

Таким образом, одно антисообщение может вызвать поток антисообщений и цепную реакцию откатов логических процессов модели. Случай, когда локальное время логического процесса **Т** равно $T_1 + \Delta T$, отдельно не рассматривается, так как он может быть реализован одним из перечисленных выше способов, но для выбора этого способа требуется более детальный анализ.

Теоретически откат логического процесса может произойти до любого момента модельного времени, в который произошло какое-нибудь событие. Поэтому требуется сохранять состояние логического процесса для каждого такого момента модельного времени, что приводит к недопустимо большому расходу памяти. К настоящему времени разработано множество оптимистических алгоритмов, которые оптимизируют рассмотренную выше схему.

Одна группа алгоритмов основана на вычислении нижней границы временных меток любого будущего отката GVT (Global Virtual Time). Для вычисления GVT используется техника, аналогичная той, которая используется для вычисления временного горизонта в консервативных алгоритмах [47]. Если вычислено текущее значение GVT, то память, используемая для хранения состояний всех логических процессов (контрольные точки) с временными метками, меньшими, чем GVT, может быть освобождена. Значение GVT в распределенных моделях, использующих оптимистические алгоритмы синхронизации, является аналогом глобального модельного времени в последовательных моделях.

Другие алгоритмы, называемые "оптимистическими временными окнами" [38], ограничивают "степень оптимизма" оптимистических алгоритмов. Оптимистическое временное окно определено как [GVT, GVT+W], где W — параметр, задаваемый пользователем. Логические процессы исполняются параллельно только в пределах этого окна модельного времени. Это ограничивает использование параллелизма, но и ограничивает "глубину" откатов, а следовательно, размер памяти, необходимой для хранения контрольных точек.

К третьей группе оптимистических алгоритмов следует отнести алгоритмы, минимизирующие число антисообщений. При этом подходе отправка сообщений задерживается до тех пор, пока не будет гарантии, что посланное сообщение не будет позже возвращено назад, т. е. до тех пор, пока GVT не продвинется к модельному времени передачи сообщения. Такой подход устраняет антисообщения и возможную цепную реакцию откатов назад [49, 50].

Еще один подход, позволяющий ограничить число антисообщений, использует концепцию, называемую "оглядывание" — lookback (аналогичную предсказанию в консервативных алгоритмах синхронизации) [51, 52]. Технология, называемая "прямой отменой" (Direct Cancellation), используется для быстрой отмены некорректных сообщений, что также ограничивает число антисообщений [53, 54].

В другом подходе оптимистические алгоритмы "откладывают" (lazy cancellation) посылку антисообщений при откате. Например, в рассмотренном примере (см. рис. 3) логический процесс **M**, сделав откат до момента модельного времени T_3, не сразу посылает антисообщение логическому процессу **T**, а только в момент $T_1 + \Delta T$, если повторное вычисление породило сообщение "Нет", которое не совпадает с сообщением, посланным ранее ("Да"). Такое откладывание посылки антисообщения может себя оправдать, если при более сложной логике модели логический процесс **M**, даже получив сообщение "Ответ", все равно пошлет собщение "Да".

Несколько оптимистических алгоритмов были разработаны для решения проблемы расхода памяти при сохранении состояния логического процесса. Например, можно использовать откат вычислений вместо сохранения состояния логического процесса [55, 56]. Сохранение состояния может производиться не после каждого события, а после выборочных событий [57, 58]. Память, используемая для хранения состояний, может быть освобождена, даже если временная метка соответствующего события больше GVT [59].

Ранние оптимистические алгоритмы использовали определенные пользователем контрольные параметры, которые должны настраиваться пользователем для оптимизации выполнения. Впоследствии были разработаны адаптивные алгоритмы, автоматически регулирующие контрольные параметры, для оптимизации выполнения [60, 61].

Некоторые алгоритмы вместо откатов реализуют непосредственную отмену действий, вызванных сообщением. Например, если программа события, вызванного сообщением, увеличивает некоторую переменную на единицу, то программа "антисобытия", вызванного антисообщением должна, соответственно, уменьшить эту переменную на единицу и т. п. [62].

Априорно нельзя сказать, что какой-то алгоритм, консервативный или оптимистический, работает быстрее. Все зависит от динамики исполнения модели. В рассмотренном примере консервативный алгоритм обеспечивает более быстрое выполнение модели. Но в случае, когда логический процесс **Д** либо не послал ответ, либо послал "неудовлетворительный" ответ, либо послал ответ после момента модельного времени $T_1 + \Delta T$, парадокс времени бы не возник. В этом случае более быстрое выполнение модели обеспечил бы оптимистический алгоритм.

Хотя оптимистические алгоритмы требуют дополнительных вычислений для реализации антисобытий и дополнительной памяти для реализации контрольных точек, для некоторого класса моделей, в которых откаты происходят, но сравнительно редки, оптимистические алгоритмы могут дать выигрыш во времени исполнения модели за счет более эффективного использования параллелизма.

Перспективным подходом была бы разработка системы распределенного моделирования с библиотекой различных консервативных и оптимистических алгоритмов. На основе экспертной оценки разработчика модели и/или на основе тестового исполнения модели на ограниченном отрезке модельного времени со снятием статистики можно было бы определить, какой из классов алгоритмов, консервативный или оптимистический, более подходит для данной модели.

Оптимистический алгоритм может быть выбран сознательно для определенных моделей решения задач оптимизации. В этом случае откат в модельном времени, как в логическом программировании, может быть инструментом выбора оптимального варианта.

3. Имитационное моделирование высокого уровня

Следующим этапом развития ИМ является имитационное моделирование высокого уровня. "Высокий уровень" означает, что компонентами такого моделирования являются не объекты, как в последовательном ИМ, и не логические процессы, как в распределенном ИМ, а сами имитационные модели. Целью создания таких проектов является разработка интегрированных окружений, включающих в себя ранее реализованные имитационные модели.

Министерство обороны США выступило с инициативой создания инструмента, поддерживающего переиспользование и взаимодействие **разнородных** имитационных моделей для уменьшения времени и стоимости разработок [63]. Эта работа, которая получила название "Архитектура высокого уровня" (HLA — High Level Architecture), началась в 1995 году и привела к принятию серии стандартов IEEE (Institute of Electrical and Electronics Engineers) в 2000 году.

Стандарт HLA имеет два источника своего развития. Первым источником является проект SIMNET (SIMulator NETworking). Целью проекта являлась разработка с помощью распределенного моделирования тренажеров для обучения и тренировки персонала при имитации военных действий [64]. Эти работы привели к созданию стандарта DIS (Distributed Interactive Simulation) [65] для взаимосвязи (interconnection) распределенных тренажеров. Вторым источником является протокол ALSP (Aggregate Level Simulation Protocol) для взаимодействия (interoperability) и переиспользования тренажеров в военных играх [66].

С 1996 года было разработано несколько версий HLA. Развитию HLA способствовали как практические реализации, так и работы по формальному моделированию и анализу

описаний интерфейсов, предлагаемых HLA. Примером такого исследования является модель HLA [67], разработанная на языке описания структуры программного обеспечения (Architectural Description Language) Wright [68]. С помощью этой модели были проверены и улучшены семантика и описание HLA.

3.1. Введение в HLA

Ключевыми понятиями HLA являются Federation и Runtime Infrastructure (RTI). Federation — это объединение (федерация) компонентов имитационного моделирования, называемых "федератами" (Federates).

В отличие от объектов в последовательном ИМ и логических процессов в распределенном ИМ федераты в одной федерации могут быть разнородными. Федерат может быть имитационной моделью, тренажером, управляемым человеком, программой, осуществляющей сбор данных, интерфейсом с реальным объектом, например летательным аппаратом. Взаимодействие федератов осуществляется посредством обмена данными. Обмен данными и исполнение федератов в едином модельном времени осуществляются с помощью программной оболочки (инфраструктуры) RTI. Инфраструктура RTI является, в сущности, распределенной операционной системой для федерации. Взаимодействие федератов с RTI происходит посредством вызова федератами сервисов (servises) RTI.

Стандарт HLA не предъявляет требований к реализации федератов и RTI, а только определяет правила оформления федератов и федерации и интерфейс между федератами и RTI. Стандарт состоит из трех частей:

— правила HLA [69], определяющие базисные принципы, лежащие в основе HLA;

— шаблоны федерата и федерации [70], обеспечивающие федерату стандартный формат декларирования общих (доступных другим федератам) данных;

— спецификация интерфейса [71] федератов с сервисами RTI.

Сервисы RTI делятся на следующие группы:

— управление федерацией (Federation management services). Эти сервисы используются для создания и функционирования федерации в целом;

— управление декларациями (Declaration management services). Эти сервисы используются федератами для объявления экспортируемых и импортируемых данных;

— управление объектами (Object management services). Эти сервисы используются для работы с объектами и их атрибутами;

— управление правом доступа (Ownership management services). Эти сервисы используются для передачи прав между федератами на модификацию значений атрибутов объектов;

— управление распределением данных (Data distribution management services). Эти сервисы позволяют уменьшать объем данных, пересылаемых между федератами, за счет более эффективного распределения данных;

— управление временем (Time management services). Эти сервисы синхронизируют продвижение локального модельного времени федератов при исполнении федерации.

3.2. Управление временем в HLA

Поскольку темой данной статьи является модельное время, рассмотрим подробнее управление временем в HLA. Схема выполнения федерации в модельном времени в HLA представлена на рис. 4.

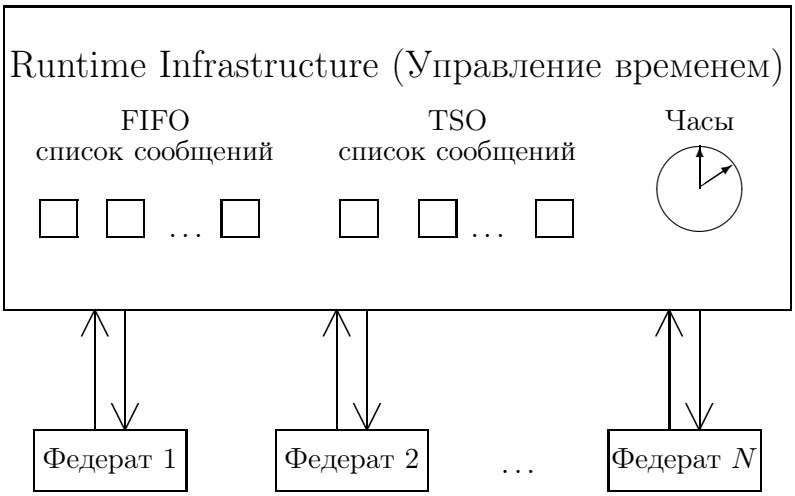


Рис. 4. Схема выполнения модели высокого уровня.

Каждый федерат имеет часы локального времени. Для одной категории федератов локальное время совпадает с реальным временем. Это федераты, представляющие собой виртуальные окружения для тренировки (тренажеры), или федераты, связанные с реальным оборудованием (hardware-in-the-loop). Выполнение событий для них синхронизовано с реальным временем и не требует механизмов синхронизации. Вторая категория федератов — это имитационные модели, локальное время которых является виртуальным и изменяется либо с постоянным шагом, либо с помощью механизма дискретных событий, которые упорядочены в порядке неубывания их временных меток.

Для упорядочивания событий используется либо консервативный, либо оптимистический алгоритм синхронизации. Федерат, который использует консервативный алгоритм синхронизации, будем называть консервативным федератом, а федерат, который использует оптимистический алгоритм синхронизации, — оптимистическим федератом. В одной федерации могут находиться как консервативные, так и оптимистические федераты. Причем каждый федерат, взаимодействуя с другими федератами, “не знает”, каким способом (пошаговым, консервативным или оптимистическим) они реализованы.

Инфраструктура RTI реализует передачу сообщений между федератами и синхронизирует их выполнение в едином модельном времени. На рис. 4 представлены три основных компонента RTI: часы модельного времени, очередь сообщений, упорядоченных в порядке поступления сообщений (FIFO — First In First Out), и очередь сообщений, упорядоченных в порядке неубывания временных меток сообщений (TSO — Time Stamp Order).

Федераты, посылая сообщения, указывают федерата-получателя и в какую очередь (FIFO или TSO) это сообщение должно быть помещено. В случае TSO-сообщения обязательно должна быть указана временная метка сообщения. Временная метка в HLA — это абстрактная структура данных, определенная в [70].

При поступлении сообщения в FIFO-очередь оно передается федерату-получателю без задержки. При поступлении сообщения в TSO-очередь оно задерживается и передается федерату-получателю в соответствии с тем, каким (консервативным или оптимистическим) является федерат.

Взаимодействие федерата с RTI осуществляется с помощью вызова интерфейсных

функций (сервисов). Состав и синтаксис этих функций определены в [71]. Рассмотрим реализацию механизма продвижения локального модельного времени федерата. В зависимости от того, каким (консервативным или оптимистическим) является федерат, используются разные группы интерфейсных функций.

3.2.1. Консервативные федераты

Для работы с консервативными федератами RTI имеет интерфейсную функцию

NextEventRequest (T).

Консервативный федерат имеет интерфейсные функции:

— *ReflectAttributeValues* ();

— *TimeAdvanceGrant* (T).

Когда консервативный федерат завершил все действия, связанные с текущим значением своего локального модельного времени, и готов перейти к выполнению следующего события с временной меткой T_1, он вызывает интерфейсную функцию *NextEventRequest* (Запрос следующего события)

NextEventRequest (T_1)

и ожидает разрешения на посланный запрос.

Инфраструктура RTI, получив вызов *NextEventRequest*, анализирует состояние TSO-очереди. Если TSO-очередь не содержит сообщений с временными метками, меньшими или равными T_1, и RTI гарантирует, что и в дальнейшем не будет таких сообщений, то она вызывает интерфейсную функцию федерата *TimeAdvanceGrant* (Разрешение продвижения времени)

TimeAdvanceGrant (T_1).

Если TSO-очередь содержит сообщения с временными метками, меньшими или равными T_1, то RTI посылает федерату первое сообщение в очереди. Пусть временная метка этого сообщения T_2, тогда RTI вызывает интерфейсную функцию федерата

TimeAdvanceGrant (T_2).

Посылка сообщения в стандартном формате осуществляется с помощью вызова интерфейсной функции федерата *ReflectAttributeValues* (Передать значения атрибутов).

Получив *TimeAdvanceGrant*, консервативный федерат устанавливает свое локальное модельное время равным T_1 или T_2 и выполняет соответствующее событие.

Таким образом, RTI поддерживает консервативную реализацию федерата. Для этого в RTI должен быть реализован один из известных консервативных алгоритмов, например вычисление LBTS — нижней границы временных меток сообщений. Некоторые реализации RTI требуют, чтобы федераты использовали технику “предсказаний” при посылке сообщений.

Следует отметить, что HLA стандартизует только интерфейсы федератов с RTI, а не сами реализации. Поэтому корректность исполнения распределенной федерации, т. е. гарантия, что консервативному федерату не будет послано сообщение с временной меткой, меньшей, чем его локальное модельное время, обеспечивается только при условии корректной реализации самого федерата, ответственность за которую несет разработчик федерата.

Условием корректной реализации федерата является задержка исполнения федерата между вызовом функции *NextEventRequest* и получением разрешения на продвижение модельного времени *TimeAdvanceGrant*. Другим условием корректности исполнения федерата является гарантия, что после исполнения интерфейсной функции

NextEventRequest (T)
федерат не пошлет сообщений с временной меткой, равной T. Если это ограничение является слишком жестким, то используются другие интерфейсные функции RTI.

3.2.2. Оптимистические федераты

Для работы с оптимистическими федератами RTI имеет интерфейсные функции:
— *FlushQueuetRequest* (T);
— *Retract* ().
Оптимистический федерат имеет интерфейсные функции:
— *ReflectAttributeValues* ();
— *FlushQueueGrant* (T).

Когда оптимистический федерат завершил все действия, связанные с текущим значением своего локального модельного времени, и готов перейти к выполнению следующего события с временной меткой T_1, он вызывает интерфейсную функцию *FlushQueuetRequest* (Запрос считывания очереди)

FlushQueuetRequest (T_1).

Инфраструктура RTI, получив вызов *FlushQueuetRequest*, посылает оптимистическому федерату все содержимое TSO-очереди. Сообщения посылаются с помощью вызова интерфейсной функции федерата *ReflectAttributeValues*. После этого федерат устанавливает новое значение своего локального модельного времени и выполняет соответствующие события.

При поступлении сообщения с временной меткой, меньшей, чем локальное модельное время, оптимистический федерат должен самостоятельно осуществить откат и послать, если требуется, антисообщения. Антисообщения посылаются с помощью интерфейсной функции *Retract* (Отменить).

Инфраструктура RTI вычисляет на основе информации обо всех федератах значение GVT — нижней границы временных меток любого будущего отката и вызывает интерфейсную функцию *FlushQueueGrant* (Подтверждение считывания очереди)

FlushQueueGrant (GVT).

Получив *FlushQueueGrant*, оптимистический федерат может освободить память, занимаемую контрольными точками с временными метками, меньшими, чем GVT, так как отката к этим контрольным точкам не будет. Таким образом, RTI поддерживает и оптимистическую реализацию федерата.

Заключение

Все рассмотренные выше типы ИМ (последовательное, распределенное и высокого уровня) в настоящее время широко используются и бурно развиваются. Каждый тип используется для своего класса приложений и уровня вычислительной техники.

Новые задачи и общее развитие программирования вызывают развитие и основных направлений ИМ. К таким задачам можно отнести:

— разработку глобальных и крупномасштабных моделей (Large-Scale Simulation, Ultra-Large-Scale Simulation) типа моделей глобальных коммуникационных сетей, моделей изменения климата и т. п.;

— разработку моделей, использующих Интернет (Web-based Simulation);

— использование имитационных моделей в оперативных контурах систем управления для выбора оптимального варианта или оценки действий управляющего персонала;

— использование в имитационных моделях Grid- и Мета-технологий и др.

Список литературы

[1] Дал О.И., Нигард К. Симула — язык для программирования и описания систем с дискретными событиями // Алгоритмы и алгоритмические языки. Вып. 2. М.: ВЦ АН СССР, 1967.

[2] Нейлор Т. Машинные имитационные эксперименты с моделями экономических систем. М.: Мир, 1975.

[3] Шеннон Р. Имитационное моделирование систем — искусство и наука. М.: Мир, 1978.

[4] Шрайбер Т.Дж. Моделирование на GPSS. М.: Машиностроение, 1984.

[5] Киндлер Е. Языки моделирования. М.: Энергоатомиздат, 1985.

[6] Прицкер А. Введение в имитационное моделирование и язык СЛАМ-2. М.: Мир, 1987.

[7] Лоу А.М., Кельтон А.Д. Имитационное моделирование. СПб., 2004.

[8] http://www. universetoday.com/am/publish/computer_simulation_exploding_star.html

[9] Fujimoto R.M. Distributed simulation systems // Proc. of the Winter Simulation Conf. 2003. P. 124–134.

[10] Корнеев В.Д. Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000.

[11] Lamport L. Time, clocks, and the ordering of events in a distributed systems // Commun. ACM. 1978. Vol. 21(7). P. 558–565.

[12] Ferscha A. Parallel and distributed simulation of discrete event systems // Parallel and Distributed Computing Handbook. McGraw-Hill, 1996. P. 1003–1041.

[13] Казаков Ю.П., Смелянский Р.Л. Об организации распределенного имитационного моделирования // Программирование. 1994. № 2. С. 45–63.

[14] Fujimoto R.M. Parallel and Distributed Simulation Systems. Wiley Interscience, 2000.

[15] Fujimoto R.M. Parallel and distributed simulation systems // Proc. of the Winter Simulation Conf. 2001. P. 147–157.

[16] Bryant R.E. Simulation of Packet Communications Architecture Computer Systems. MIT-LCS-TR-188. 1977.

[17] Chandy K.M., Misra J. Distributed simulation: a case study in design and verification of distributed programs // IEEE Transactions on Software Engineering. 1978. Vol. SE–5(5). P. 440–452.

[18] Jha V., Bagrodia R. Simultaneous events and lookahead in simulation protocols // ACM Transactions on Modeling and Computer Simulation. 2000. Vol. 10(3). P. 241–267.

[19] Chandy K.M., Misra J. Asynchronous distributed simulation via a sequence of parallel computations // Communications of the ACM. 1981. Vol. 24(4). P. 198–205.

[20] Misra J. Distributed discrete-event simulation // ACM Computing Surveys. 1986. Vol. 18(1). P. 39–65.

[21] Ayani R. A parallel simulation scheme based on the distance between objects // Proc. of the SCS Multiconference on Distributed Simulation, Society for Computer Simulation. 1989. P. 113–118.

[22] Lubachevsky B.D. Efficient distributed event-driven simulations of multiple-loop networks // Communications of the ACM. 1989. Vol. 32(1). P. 111–123.

[23] Cai W., Turner S.J. An algorithm for distributed discrete-event simulation — "carrier null message" approach // Proc. of the SCS Multiconference on Distributed Simulation, SCS Simulation Series. 1990. P. 3–8.

[24] Meyer R.A., Bagrodia R.L. Path lookahead: a data flow view of pdes models // Proc. of the 13th Workshop on Parallel and Distributed Simulation (PADS99). 1999. P. 12–19.

[25] Xiao Z., Unger B. Scheduling critical channels in conservative parallel simulation // Proc. of the 13th Workshop on Parallel and Distributed Simulation (PADS99). 1999. P. 20–28.

[26] Lamport L. Concurrent reading and writing of clocks // ACM Trans. Comp. Syst. 1990. Vol. 8(4). P. 305–310.

[27] Raynal M., Singhal M. Logical time: capturing causality in distributed systems // IEEE Computer. 1996. Vol. 29(2). P. 49–56.

[28] Baldony R., Raynal M. Fundamentals of distributed computing: a practical tour of vector clock systems // IEEE Distributed Systems Online. 2002.
(http://csdl.computer.org/comp/mags/ds/2002/02/o2001.pdf).

[29] Lee B., Cai W. A causality based time management mechanism for federated simulations // Proc. of the 15th Workshop on Parallel and Distributed Simulation. 2001. P. 83–90.

[30] Moore R.E. Methods and Applications of Interval Analyses. SIAM. Philadelphia, 1979.

[31] Fujimoto R.M. Exploiting temporal uncertainty in parallel and distributed simulations // Proc. of the 13th Workshop on Parallel and Distributed Simulation. 1999. P. 46–53.

[32] Beraldi R., Nigro L. Exploiting temporal uncertainty in time warp simulations // Proc. of the 4th Workshop on Distributed Simulation and Real-Time Applications. 2000. P. 39–46.

[33] Loper M., Fujimoto R.M. Exploiting temporal uncertainty in process-oriented distributed simulations // Proc. of the Winter Simulation Conf. 2004. P. 395–400.

[34] Ramanathan P., Shin K., Butler R. Fault-tolerant clock synchronization in distributed systems // IEEE Computer. 1990. Vol. 23, N 10. P. 33–42.

[35] Liskov B. Practical uses of synchronized clocks in distributed systems // Distributed Computing. Vol. 6. 1993. P. 211–219.

[36] Das S.R. Adaptive protocols for parallel discrete event simulation // J. of the Operational Research Society. 2000. Vol. 51. P. 385–394.

[37] Altuntas B., Wysk R.A. A framework for adaptive synchronization of distributed simulations // Proc. of the Winter Simulation Conf. 2004. P. 371–377.

[38] Sokol L.M., Stucky B.K. MTW: experimental results for a constrained optimistic scheduling paradigm // Proc. of the SCS Multiconference on Distributed Simulation. 1990. P. 169–173.

[39] Rao D.M., Thondugulam N.V., Wilsey P.A. Unsynchronized parallel discrete event simulation // Proc. of the Winter Simulation Conference. 1998. P. 1563–1570.

[40] Rao D.M., Wilsey P.A. An ultra-large-scale simulation framework // J. of Parallel Distrib. Comput. 2002. Vol. 62(11). P. 1670–1693.

[41] Lorenz P.H., Dorwarth K.C. Towards a Web-based simulation environment // Proc. of the Winter Simulation Conf. 1997. P. 1338–1344.

[42] Таненбаум Э., Стеен М. Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003.

[43] Helary J. Observing global states of asynchronous distributed applications // Proc. Intern. Workshop on Distributed Algorithms. Lecture Notes Computer Science. Berlin: Springer-Verlag, 1989. Vol. 392. P. 124–135.

[44] Bracha G., Toueg S. Distributed deadlock detection // Distributed Computing. 1987. Vol. 2. P. 127–138.

[45] Chandy K., Lamport L. Distributed snapshots: determining global states of distributed systems // ACM Trans. Comp. Syst. 1985. Vol. 3, N 1. P. 63–75.

[46] Mattern F. Algorithms for distributed termination detection // Distributed Computing. 1987. Vol. 2. P. 161–175.

[47] Mattern F. Efficient algorithms for distributed snapshots and global virtual time approximation // J. of Parallel and Distributed Computing. 1993. Vol. 18, N 4. P. 423–434.

[48] Jefferson D. Virtual time // ACM Transactions on Programming Languages and Systems. 1985. Vol. 7, N 3. P. 404–425.

[49] Dickens P.M., Reynolds J. SRADS with local rollback // Proc. of the SCS Multiconference on Distributed Simulation. 1990. P. 161–164.

[50] Steinman J.S. SPEEDES: a multiple-synchronization environment for parallel discrete event simulation // Intern. J. on Computer Simulation. 1992. P. 251–286.

[51] Chen G., Szymanski B.K. Lookback: a new way of exploiting parallelism in discrete event simulation // Proc. of the 16th Workshop on Parallel and Distributed Simulation. 2002. P. 153–162.

[52] Chen G., Szymanski B.K. Four types of lookback // Proc. of the 17th Workshop on Parallel and Distributed Simulation. 2003. P. 3–10.

[53] Fujimoto R.M. Time warp on a shared memory multiprocessor // Transactions of the Society for Computer Simulation. 1989. Vol. 6, N 3. P. 211–239.

[54] Zhang J.L., Tropper C. The dependence list in time warp // Proc. of the 15th Workshop on Parallel and Distributed Simulation. 2001. P. 35–45.

[55] Jefferson D.R. Virtual time II: storage management in distributed simulation // Proc. of the Ninth Annual ACM Symposium on Principles of Distributed Computing. 1990. P. 75–89.

[56] Lin Y.B., Preiss B.R. Optimal memory management for time warp parallel simulation // ACM Transactions on Modeling and Computer Simulation. 1991. Vol. 1, N 4. P. 283–307.

[57] Lin Y.B., Preiss B.R. Selecting the checkpoint interval in time warp simulations // Proc. of the 7th Workshop on Parallel and Distributed Simulation. 1993. P. 3–10.

[58] Palaniswamy A.C., Wilsey P.A. An analytical comparison of periodic checkpointing and incremental state saving // Proc. of the 7th Workshop on Parallel and Distributed Simulation. 1993. P. 127–134.

[59] Preiss B.R., Loucks W.M. Memory management techniques for time warp on a distributed memory machine // Proc. of the 9th Workshop on Parallel and Distributed Simulation. 1995. P. 30–39.

[60] Ferscha A. Probabilistic adaptive direct optimism control in time warp // Proc. of the 9th Workshop on Parallel and Distributed Simulation. 1995. P. 120–129.

[61] Das S.R., Fujimoto R.M. Adaptive memory management and optimism control in time warp // ACM Transactions on Modeling and Computer Simulation. 1997. Vol. 7, N 2. P. 239–271.

[62] Carothers C.D., Perumalla K. Efficient optimistic parallel simulation using reverse computation // ACM Transactions on Modeling and Computer Simulation. 1999. Vol. 9, N 3. P. 224–253.

[63] Fujimoto R.M. Time management in the high level architecture // Simulation. 1998. Vol. 71, N 6. P. 388–400.

[64] Mille D.C., Thorpe J.A. SIMNET: the advent of simulator networking // Proc. of the IEEE. 1995. Vol. 83, N 8. P. 1114–1123.

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

[66] Wilson A.L., Weatherly R.M. The aggregate level simulation protocol: an evolving system // Proc. of the Winter Simulation Conf. 1994. P. 781–787.

[67] 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.

[68] Allen R., Garlan D. A formal basis for architectural connection // ACM Transactions on software Engineering and Methodology. 1997. P. 213–249.

[69] IEEE Std P1516. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — Framework and Rules. N.Y.: Institute of Electrical and Electronics Engineers, Inc., 2000.

[70] IEEE Std P1516.2. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — Object Model Template (OMT) Specification. N.Y.: Institute of Electrical and Electronics Engineers, Inc., 2000.

[71] IEEE Std P1516.3. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — Federation Development and Execution Process. N.Y.: Institute of Electrical and Electronics Engineers, Inc., 2000.

Поступила в редакцию 30 ноября 2004 г.,
в переработанном виде — 18 апреля 2005 г.