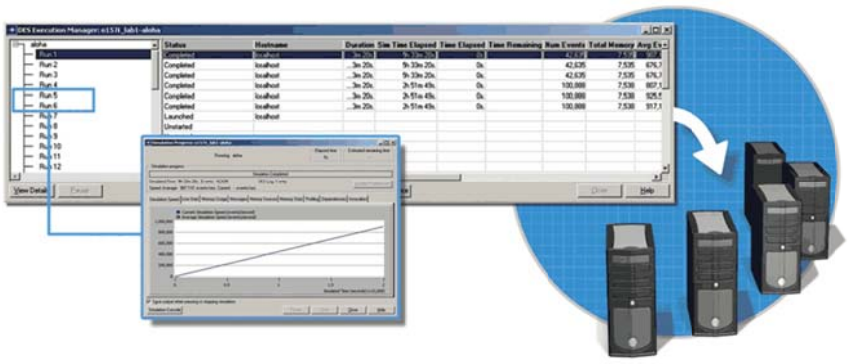
Л.А. Муравьева-Витковская

ОСНОВЫ РАСПРЕДЕЛЕННОГО МОДЕЛИРОВАНИЯ



Санкт-Петербург

2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

Л.А. Муравьева-Витковская

ОСНОВЫ РАСПРЕДЕЛЕННОГО МОДЕЛИРОВАНИЯ

Учебное пособие

Санкт-Петербург

2013

Муравьева-Витковская Л.А. Основы распределенного моделирования. – СПб: НИУ ИТМО, 2013. – 150 с.

В пособии, содержащем три раздела, излагаются современные подходы к распределенному моделированию сложных систем. В первом разделе формулируются основные понятия, определения, цели и задачи распределенного моделирования, рассматриваются методы и инструментальные средства распределенного моделирования. Во втором разделе излагаются общие принципы распределенного моделирования, декомпозиция модели сложной системы на компоненты и организация их взаимодействия на основе агентной технологии. Последний раздел посвящен описанию стандартов и основных направлений развития систем распределенного моделирования.

Пособие предназначено для магистрантов, обучающихся по направлениям 230100 «Информатика и вычислительная техника» и 231000 «Программная инженерия». Пособие может быть полезным для аспирантов и специалистов в области телекоммуникационных и компьютерных сетей, а также студентам при подготовке выпускных квалификационных работ.

Рекомендовано к печати Ученым советом факультета компьютерных технологий и управления 09 апреля 2013 г., протокол № 4.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время одним из перспективных направлений развития математического моделирования является распределенное моделирование сложных систем, которое постоянно совершенствуется благодаря появлению новых информационных технологий, телекоммуникационных систем, массовому применению компьютеров и компьютерных сетей.

Распределенное моделирование является мощным и эффективным инструментальным средством исследования самых разнообразных сложных систем из различных областей человеческой деятельности. Многообразие процессов, протекающих в исследуемых сложных системах, обусловливает и многообразие математических методов и средств, используемых в распределенном моделировании.

Распределенное моделирование, как и моделирование в целом, представляет собой сложнейший многоэтапный процесс исследования сложных систем, направленный на выявление свойств и закономерностей, присущих исследуемым системам, с целью их проектирования или модернизации. В процессе распределенного моделирования решается множество взаимосвязанных задач, основными среди которых являются разработка модели, анализ свойств и выработка рекомендаций по модернизации существующей или проектированию новой системы [1].

Большинство сложных технических систем, в том числе вычислительные системы и компьютерные сети, описываются в терминах дискретных случайных процессов с использованием вероятностных методов [1, 3, 9, 11, 12]. При этом широкое применение находят математические модели, отражающие структурно-функциональную организацию исследуемых систем, построенные на основе моделей теории массового обслуживания, анализ которых может проводиться методами распределенного моделирования.

Низкая стоимость, быстрота развертывания, широкие функциональные возможности компьютерных сетей обеспечивают быстрые темпы развития и внедрения систем распределенного моделирования. Развитие систем распределенного моделирования сопровождается непрерывной сменой технологий, в основе которых лежат стандарты распределенного моделирования. Учитывая преимущества распределенного моделирования при исследовании сложных систем, можно прогнозировать, что в ближайшие годы системы распределенного моделирования останутся наиболее актуальными и будут постоянно совершенствоваться.

В пособии для более эффективного усвоения материала фрагменты, представляющие наибольший интерес, выделяются разными шрифтами, что позволяет акцентировать внимание на тех или иных аспектах, которые являются важными для понимания описанных моделей и методов распределенного моделирования.

Полужирный курсив выделяет наиболее важные и часто используемые термины и понятия, для которых дается чёткое определение или подробное описание.

Полужирным шрифтом выделяются прочие общепринятые термины и понятия, часто встречающиеся в литературе, но не имеющие чёткого определения, а также вспомогательные заголовки, названия и т.д.

Курсив выделяет в тексте ключевые слова и фразы, раскрывающие смысл излагаемого материала, на которые следует обратить внимание, а также выделяет термины и понятия, которые определены в других разделах учебного пособия.

Целью данного учебного пособия является формирование у неподготовленного читателя начального представления о принципах моделирования сложных систем, рассмотренных на примере математических моделей и методов их исследования с применением распределенного моделирования.

Для достижения указанной цели в учебном пособии:

- вводится четкая однозначная терминология, используемая в процессе изложения материала;
- формируется представление о моделях сложных систем, их многообразии, а также о величинах, описывающих эти модели;
- формулируются задачи распределенного моделирования как инструментального средства исследования сложных систем, в том числе технических систем, таких как вычислительные машины, комплексы, системы и сети;
- описываются этапы распределенного моделирования сложных систем;
- излагаются общие принципы распределенного моделирования сложных систем;
- рассматриваются примеры наиболее известных и широко применяемых на практике систем распределенного моделирования.

Структура учебного пособия. Пособие содержит *введение*, три *основных раздела* и *список литературы*.

В основных разделах учебного пособия излагаются современные подходы к распределенному моделированию сложных систем. Материал каждого раздела разбит на параграфы, которые имеют двойную нумерацию. Некоторые параграфы разбиты на пункты с тройной нумерацией.

В **первом разделе** формулируются основные понятия и определения, используемые при изложении материала, цели и задачи распределенного моделирования, требования к моделям сложных систем, приводится классификация моделей, излагаются основы технологии моделирования сложных систем, рассматриваются методы и инструментальные средства распределенного моделирования, детально описываются этапы распределенного моделирования сложных систем.

Во **втором разделе** излагаются общие принципы распределенного моделирования, декомпозиция модели сложной системы на компоненты, проблемы распределенного моделирования дискретных, непрерывных и гибридных сложных систем, методы представления непрерывно меняющегося значения переменной для удаленного компонента, организация взаимодействия распределенных компонентов на основе агентной технологии.

Третий раздел посвящен описанию стандартов распределенного моделирования, рассмотрению существующих систем распределенного моделирования, их свойств, классификации и основных направлений развития, примеров наиболее известных и широко применяемых на практике систем распределенного моделирования.

Каждый раздел заканчивается *резюме*, которое содержит краткое изложение представленного в разделе материала.

Представленный *список литературы* не претендует на полноту и содержит ограниченный перечень литературных источников, которые в той или иной мере использовались при написании пособия.

Учебное пособие предназначено, прежде всего, для студентов, обучающихся в магистратуре по направлению подготовки 231000 «Программная инженерия», профилю подготовки «Информационно-вычислительные системы», изучающих дисциплину «Распределенное моделирование» и связанные с ней дисциплины. Пособие может быть полезным в качестве введения в проблематику распределенного моделирования для выпускников (бакалавров, магистрантов и специалистов), подготавливающих выпускные квалификационные работы, в которых требуется выполнить исследование некоторой сложной системы, например, компьютерной сети, а также для магистрантов, аспирантов и специалистов, занимающихся исследованием реальных сложных систем с применением методов распределенного моделирования.

Раздел 1. Технология моделирования сложных систем

1.1. Основные понятия и терминология

1.1.1. Словарь терминов

Система (от греч. systema – целое, составленное из частей; соединение) – совокупность взаимосвязанных элементов, объединенных в одно целое для достижения некоторой цели, определяемой назначением системы.

Элемент – минимальный неделимый объект, рассматриваемый как единое целое.

Сложная система – это крупномасштабная система, характеризующаяся большим числом входящих в ее состав элементов и сложных связей между ними, наличием множества протекающих в ней процессов, многообразием причинно-следственных взаимосвязей между ними, а также стохастической природой функционирования.

Комплекс – совокупность взаимосвязанных систем.

Элемент, система и комплекс – понятия относительные. Любой элемент может рассматриваться как система, если его расчленить на более мелкие составляющие – элементы. И наоборот, любой комплекс может рассматриваться как система, если входящие в его состав системы трактовать как элементы. В связи с этим, понятия «система» и «комплекс» часто трактуют как эквивалентные понятия [1]. Например, компьютерную сеть можно рассматривать как систему, элементами которой являются узлы обработки и передачи данных, каналы связи. В то же время, узел обработки данных компьютерной сети можно рассматривать как систему, состоящую из таких элементов, как центральный процессор, оперативная память, накопители на магнитных дисках, устройства ввода-вывода и т.д.

Для описания системы необходимо определить ее *структуру* и *функцию* и, соответственно, *структурную и функциональную организацию* [1].

Структура системы задается перечнем элементов, входящих в состав системы, и связей между ними.

Способы описания структуры системы:

- ***графический*** – в форме *графа*, в котором вершины соответствуют элементам системы, а дуги – связям между ними; в форме *схем*, широко используемых в инженерных приложениях, в которых элементы обозначаются в виде специальных символов;
- ***аналитический*** – путем задания количества типов элементов, числа элементов каждого типа и матрицы связей (инцидентности), определяющей взаимосвязь элементов.

Функция системы – правило достижения поставленной цели, описывающее поведение системы и направленное на получение результатов, предписанных назначением системы.

Способы описания функции системы:

- ***алгоритмический*** – словесное описание в виде последовательностей шагов, которые должна выполнять система для достижения поставленной цели;
- ***аналитический*** – в виде математических зависимостей в терминах некоторого математического аппарата: теории множеств, теории случайных процессов, теории дифференциального или интегрального исчисления и т.п.;
- ***графический*** – в виде временных диаграмм или графических зависимостей;
- ***табличный*** – в виде различных таблиц, отражающих основные функциональные зависимости, например, в виде таблиц булевых функций, автоматных таблиц функций переходов и выходов и т.п.

Организация системы – способ достижения поставленной цели за счет выбора определенной структуры и функции системы. В соответствии с этим различают *структурную и функциональную организацию* системы.

Функциональная организация определяется способом порождения функций системы, достаточных для достижения поставленной цели.

Структурная организация определяется набором элементов и способом их соединения в структуру, обеспечивающую возможность реализации возлагаемых на систему функций.

Функциональная организация реализуется безотносительно к необходимым для этого средствам (элементам), в то время как *структурная организация* определяется функцией, возлагаемой на систему.

1.1.2. Свойства сложных систем

Любым сложным системам присущи фундаментальные свойства, требующие применения системного подхода при их исследовании методами математического моделирования. Такими свойствами являются:

- ***целостность***, означающая, что система рассматривается как единое целое, состоящее из *взаимодействующих* элементов, возможно неоднородных, но одновременно *совместимых*;
- ***связность*** – наличие *существенных* устойчивых связей между элементами и/или их свойствами, причем с системных позиций значение имеют не любые, а лишь существенные связи, которые определяют *интегративные* свойства системы;
- ***организованность*** – наличие определенной структурной и функциональной организации, обеспечивающей снижение энтропии (степени неопределенности) системы по сравнению с энтропией системообразующих факторов, определяющих возможность создания системы, к которым относятся: число элементов системы, число существенных связей, которыми может обладать каждый элемент, и т.п.;

• ***интегративность*** – наличие качеств, присущих системе в целом, но не свойственных ни одному из ее элементов в отдельности; другими словами, интегративность означает, что свойства системы хотя и зависят от свойств элементов, но не определяются ими полностью.

Таким образом, можно сделать следующие *важные выводы*:

• система не есть простая совокупности элементов;

• расчленяя систему на отдельные части и изучая каждую из них в отдельности, нельзя познать все свойства системы в целом.

В общем случае моделирование направлено на решение задач:

• *анализа*, связанных с оценкой эффективности систем, задаваемой в виде совокупности *показателей эффективности*;

• *синтеза*, направленных на построение оптимальных систем в соответствии с выбранным *критерием эффективности*.

Эффективность – степень соответствия системы своему назначению.

Эффективность систем обычно оценивается набором показателей эффективности [1].

Показатель эффективности (качества) – мера одного свойства системы. Показатель эффективности всегда имеет количественный смысл.

Количество показателей эффективности технических систем во многих случаях, может оказаться достаточно большим. Обычно показатели эффективности являются противоречивыми. Это означает, что изменение структурной или функциональной организации системы приводит к улучшению одних показателей и, в то же время, к ухудшению других показателей эффективности, что существенно осложняет выбор наилучшего варианта (способа) структурно-функциональной организации проектируемой системы. Очевидно, что желательно иметь один показатель эффективности. Таким показателем является критерий эффективности.

Критерий эффективности – мера эффективности системы, обобщающая все свойства системы в одной оценке – значении критерия эффективности. Если при увеличении эффективности значение критерия возрастает, то критерий называется ***прямым***, если же значение критерия уменьшается, то критерий называется ***инверсным***.

Критерий эффективности служит для выбора из всех возможных вариантов структурно-функциональной организации системы наилучшего (оптимального) варианта.

Оптимальная система – система, которой соответствует максимальное (минимальное) значение прямого (инверсного) критерия эффективности из всех возможных вариантов построения системы, удовлетворяющих заданным требованиям.

Анализ (от греч. análysis — разложение, расчленение) – процесс определения свойств, присущих системе. В процессе анализа на основе сведений о функциях и параметрах элементов, входящих в состав системы,

и сведений о структуре системы определяются характеристики, описывающие свойства, присущие системе в целом.

Синтез (от греч. synthesis - соединение, сочетание, составление) – процесс порождения функций и структур, удовлетворяющих требованиям, предъявляемым к эффективности системы.

Таким образом, с понятием «эффективность» связаны следующие понятия: показатель эффективности; критерий эффективности; оптимальная система; анализ системы; синтез (создание) системы.

Количественно любая сложная система описывается совокупностью величин, которые могут быть разбиты на два класса: ***параметры*** и ***характеристики***.

1.1.3. Параметры сложных систем

Параметры – величины, описывающие первичные свойства сложной системы и являющиеся исходными данными при решении задач анализа [1].

Множество ***параметров*** *сложных технических систем* можно разделить на следующие классы (рис. 1.1):

- ***внутренние***, описывающие структурно-функциональную организацию системы, к которым относятся:
  - ***структурные параметры***, описывающие состав и структуру системы;
  - ***функциональные параметры***, описывающие функциональную организацию (режим функционирования) системы;
- ***внешние***, описывающие взаимодействие системы с внешней по отношению к ней средой, к которым относятся:
  - ***нагрузочные параметры***, описывающие входное воздействие на систему, например частоту и объем используемых ресурсов системы;
  - ***параметры внешней (окружающей) среды***, описывающие обычно неуправляемое воздействие внешней среды на систему, например помехи и т.п.

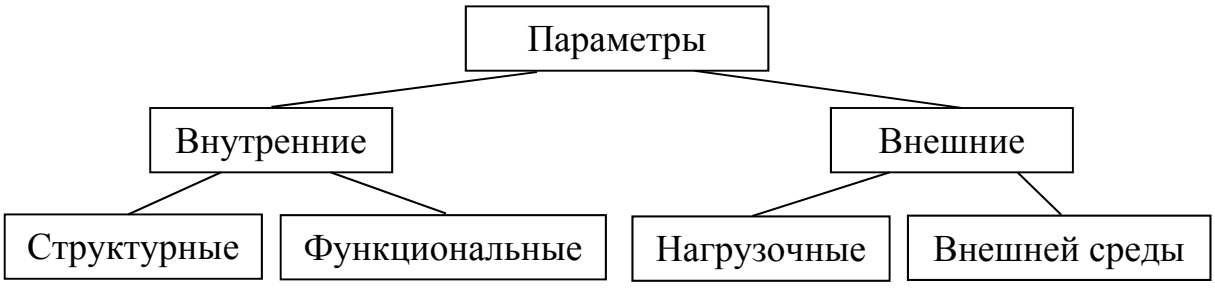


Рис. 1.1. Параметры сложных систем

Параметры сложной системы могут быть:

- детерминированными или случайными;
- управляемыми или неуправляемыми.

Детерминированные параметры – параметры, описываемые детерминированными (неслучайными, фиксированными, известными заранее) величинами.

Случайные параметры – параметры, описываемые случайными дискретными или непрерывными величинами.

Управляемые параметры – структурные, функциональные и нагрузочные параметры, удовлетворяющие требованиям, предъявляемым к эффективности системы, и определяемые на соответствующих этапах синтеза (структурном, функциональном, нагрузочном).

Неуправляемые параметры – параметры внешней (окружающей) среды, описывающие неуправляемое воздействие (возмущающий фактор) внешней среды на систему, остающиеся неизменными в процессе синтеза системы.

Рассмотрим параметры сложной системы на примере компьютерной сети [2].

В качестве структурных параметров компьютерной сети используются:

- количество узлов, входящих в состав сети, и их взаимосвязь (топология сети);
- типы узлов и состав оборудования (ЭВМ и сетевых устройств);
- технические данные устройств (производительность вычислительных систем (ВС) и сетевых устройств – маршрутизаторов и коммутаторов, емкости буферов узлов связи, пропускные способности каналов связи и т.п.);

К функциональным параметрам компьютерной сети относятся:

- способ коммутации;
- метод доступа к каналу связи;
- алгоритм выбора маршрута передачи данных в сети:
- распределение прикладных задач по узлам сети;
- режим функционирования ВС;
- последовательность выполнения прикладных задач в ВС;
- приоритеты задач и т.п.

В качестве нагрузочных параметров компьютерной сети могут использоваться:

- число типов потоков данных (аудио, видео, компьютерные);
- интенсивности поступления сообщений (пакетов, кадров) разных типов в сеть или к отдельным ресурсам (узлам и каналам связи);
- длина передаваемых по сети блоков данных;
- число типов прикладных задач;
- ресурсоемкость каждой прикладной задачи;
- объем занимаемой памяти и т.п.

1.1.4. Характеристики сложных систем

Характеристики – величины, описывающие вторичные свойства сложной системы и определяемые в процессе решения задач анализа как функция параметров, то есть эти величины являются вторичными по отношению к параметрам [1].

Характеристики сложной системы делятся на:

- глобальные, описывающие эффективность системы в целом;
- локальные, описывающие качество функционирования отдельных элементов или частей (подсистем) системы.

К глобальным характеристикам сложных технических систем относятся:

- мощностные (характеристики производительности), описывающие скоростные качества системы, измеряемые, например, количеством задач, выполняемых вычислительной системой за единицу времени;
- временные (характеристики оперативности), описывающие временные аспекты функционирования системы, например, время решения задач в вычислительной системе;
- надежностные (характеристики надежности), описывающие надежность функционирования системы;
- экономические (стоимостные) в виде стоимостных показателей, например, стоимость технических и программных средств вычислительной системы, затраты на эксплуатацию системы и т.п.;
- прочие: масса-габаритные, энергопотребления, тепловые и т.п.

Таким образом, параметры системы можно интерпретировать как некоторые входные величины, а характеристики – выходные величины, зависящие от параметров и определяемые в процессе анализа системы (рис.1.2).

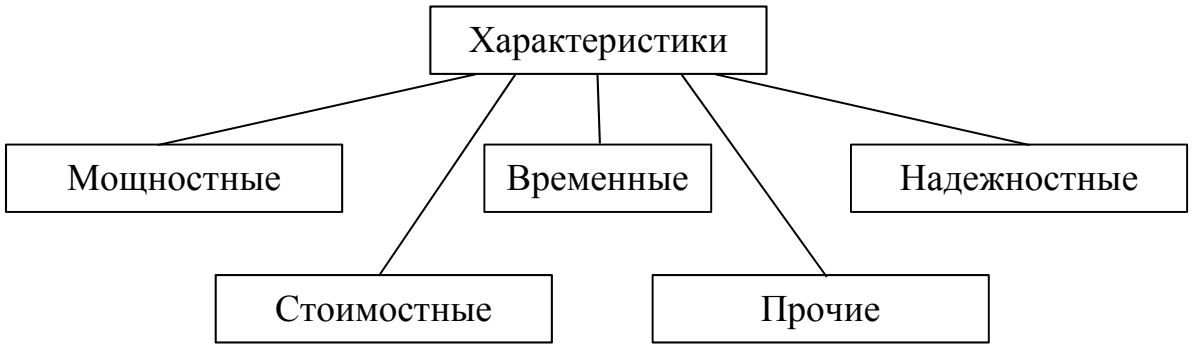


Рис. 1.2. Характеристики сложных систем

Тогда закон функционирования системы можно представить в следующем виде:

$$\vec{H}(t) = f_c(\vec{S}, \vec{F}, \vec{Y}, \vec{X}, t),$$

где f_c – функция, функционал, логические условия, алгоритм, таблица или словесное описание, определяющие правило (закон) преобразования входных величин (параметров) в выходные величины (характеристики);

$\vec{S}$ – структурные параметры;

$\vec{F}$ – функциональные параметры;

$\vec{Y}$ – нагрузочные параметры;

$\vec{X}$ – параметры внешней (окружающей) среды;

$\vec{H}(t)$ – вектор характеристик, зависящий от текущего момента времени t ($t \geq 0$):

$$\vec{H} = \{\vec{V}, \vec{T}, \vec{N}, \vec{C}, \vec{Z}\},$$

где $\vec{V}$ – мощностные характеристики;

$\vec{T}$ – временные характеристики;

$\vec{N}$ – характеристики надежности;

$\vec{C}$ – экономические характеристики;

$\vec{Z}$ – прочие характеристики.

1.2. Цели и задачи распределенного моделирования сложных систем

Распределенное моделирование – замещение исходного объекта (сложной системы) другим объектом, называемым ***моделью***, и проведение экспериментов с моделью с целью получения информации о сложной системе путем исследования свойств модели.

Объектами распределенного моделирования в технике являются сложные технические системы и протекающие в них процессы. В частности, в вычислительной технике объектами моделирования являются вычислительные машины, комплексы, системы и компьютерные сети. При этом, наибольший интерес представляют конструктивные модели, допускающие не только фиксацию свойств (как в произведениях искусств), но и исследование свойств систем (процессов), а также решение задач проектирования или модернизации сложных систем с заданными свойствами [1].

Распределенное моделирование предоставляет возможность исследования таких объектов, прямой эксперимент с которыми:

- трудно выполним;
- экономически невыгоден;
- вообще невозможен.

Распределенное моделирование является важнейшей сферой применения вычислительных систем и компьютерных сетей в различных областях науки и техники: в математике и физике, в авиа- и

автомобилестроении, в приборо- и машиностроении, в оптике, в электронике и т.д. Все более широкое распространение распределенное моделирование находит в таких областях как экономика, социология, искусство, биология, медицина и т.п. В то же время, вычислительные системы и компьютерные сети сами являются объектами моделирования на этапах проектирования новых и модернизации существующих систем, анализа эффективности использования систем в различных условиях (например, в экстремальных ситуациях, в условиях повышенных требований к надежности и живучести). Применение распределенного моделирования на этапе проектирования позволяет выполнить анализ различных вариантов предлагаемых проектных решений, определить работоспособность и оценить надежность сложной системы, выявить узкие места и мало загруженные ресурсы, а также сформулировать рекомендации по рациональному изменению (модернизации) состава и структуры или способа функциональной организации сложной системы.

Распределенное моделирование, как процесс исследования сложных систем, в общем случае предполагает решение следующих взаимосвязанных задач:

- разработка модели сложной системы;
- системный анализ характеристик функционирования сложной системы;
- системное проектирование (синтез) или модернизация сложной системы с заданными свойствами;
- детальный анализ сложной системы, полученной в результате проектирования (синтеза) или модернизации.

1.2.1. Системный подход к распределенному моделированию сложных систем

Разработка модели сложной системы состоит в выборе конкретного математического аппарата, в терминах которого формулируется модель, и построении модели или совокупности моделей исследуемой системы, отображающих возможные варианты структурно-функциональной организации системы.

В процессе разработки модели необходимо определить состав и перечень параметров и характеристик модели в терминах выбранного математического аппарата, и установить их взаимосвязь с параметрами и характеристиками исследуемой сложной системы, то есть выполнить параметризацию модели.

В основу исследования сложных систем с использованием распределенного моделирования положен ***системный подход***, конечной целью которого является системное проектирование, направленное на построение системы с заданным качеством.

Системный подход включает в себя два понятия:

- системный анализ;

• системное проектирование.

В процессе ***системного проектирования*** необходимо исходя из сведений о назначении сложной системы и требований, предъявляемых к качеству ее функционирования, определить структурную и/или функциональную организацию, обеспечивающую реализацию заданных функций при затратах средств и времени, лимитируемых заданными ограничениями и критерием эффективности. Чтобы решать такого рода задачи, необходимо располагать знаниями о том, как влияют различные способы структурной и функциональной организации на характеристики функционирования сложной системы, т.е. решать задачи ***системного анализа***.

Системное проектирование сложных технических систем, примерами которых могут служить вычислительные комплексы, системы и компьютерные сети, называется ***системотехническим проектированием***.

1.2.2. Анализ характеристик функционирования сложной системы

Анализ характеристик сложной системы с использованием разработанной модели заключается в выявлении свойств и закономерностей, присущих процессам, протекающим в сложных системах с различной организацией, и выработке рекомендаций для решения основной задачи системного проектирования – задачи синтеза.

1.2.3. Проектирование или модернизация сложной системы с заданными свойствами

В общем виде проблема системного проектирования формулируется следующим образом.

Задано назначение сложной системы, определяемое:

• перечнем функций, возлагаемых на систему;

• перечнем и значениями нагрузочных параметров, описывающих взаимодействие системы с внешней средой и потребности в ресурсах системы для реализации заданных функций;

• требованиями к характеристикам системы – мощностным, временным, надежностным, экономическим, устанавливающим предельно допустимые значения характеристик.

Требуется определить:

• структурную организацию системы, т.е. номенклатуру и состав элементов, а также конфигурацию связей между ними;

• функциональную организацию системы, т.е. режим функционирования системы,

удовлетворяющие заданным требованиям и максимизирующие (минимизирующие) прямой (инверсный) критерий эффективности.

Требования к качеству функционирования сложной системы в виде ограничений на характеристики и критерий эффективности

формулируются на основании анализа результатов модельных экспериментов.

Формирование критерия эффективности предполагает построение обобщенного показателя эффективности на основе множества частных показателей на основе одного из следующих подходов:

- построение составного критерия эффективности в виде аддитивного F_1 или мультипликативного F_2 функционала;

$$F_1 = \sum_{i=1}^{K} \alpha_i x_i \; ; \qquad F_2 = \frac{\prod_{i=1}^{k} x_i}{\prod_{i=k+1}^{K} x_i},$$

где $x_1,...,x_K$ – частные показатели эффективности; α_i - весовой коэффициент показателя x_i (весовые коэффициенты выбираются на основе экспертных оценок);

- выбор в качестве критерия эффективности F одного частного показателя при ограничениях, налагаемых на остальные показатели эффективности:

$F = x_j$ при ограничениях $x_i < x_i^*$ или $x_i > x_i^*$ для всех $x_i \neq x_j$.

Решение задачи проектирования (синтеза) новой или модернизации существующей сложной системы с заданными свойствами заключается в определении параметров структуры и функционирования системы, обеспечивающих заданные ограничения на характеристики системы.

Для упрощения решения задачи процесс проектирования (синтеза) или модернизации сложной системы можно разделить на последовательность этапов, на каждом из которых решаются частные задачи синтеза – определяются параметры, связанные с отдельными аспектами организации системы, с использованием тех или иных моделей:

- определение требований к параметрам отдельных элементов системы;
- выбор структурной организации системы (определение структурных параметров);
- выбор режима функционирования системы (определение функциональных параметров);
- определение требований к параметрам нагрузки, обеспечивающим функционирование системы с заданным качеством.

В зависимости от целей проектирования можно выделить следующие частные задачи (этапы) синтеза [1]:

- ***структурный синтез***, состоящий в выборе способа структурной организации системы, в рамках которой могут быть удовлетворены требования технического задания; структурный синтез включает в себя два этапа:
  - элементный синтез, состоящий в определении требований к параметрам отдельных элементов системы;

- топологический (конфигурационный) синтез, состоящий в определении способа взаимосвязи элементов системы, т.е. топологии (конфигурации) системы;

• ***функциональный синтез***, состоящий в выборе режима (способа) функционирования системы;

• ***нагрузочный синтез***, состоящий в определении требований к параметрам нагрузки, обеспечивающим функционирование системы с заданным качеством.

На каждом из перечисленных этапов синтеза определяются значения соответствующего подмножества параметров, характеризующих структурную, функциональную организацию сложной системы или нагрузку, возлагаемую на систему. При этом значения параметров оптимизируются лишь в отношении факторов, учитываемых на каждом из этапов синтеза, но не в отношении системы в целом. Поэтому многоэтапный синтез позволяет получить лишь приближенные оптимальные решения, качество которых проверяется путем детального анализа синтезированной сложной системы.

1.2.4. Детальный анализ сложной системы, полученной в результате проектирования или модернизации

Детальный анализ сложной системы, полученной в результате проектирования (синтеза) или модернизации, проводится с целью оценки качества решения задачи системного проектирования и полученных в процессе синтеза параметров системы, а также выявления предельных возможностей системы, узких мест в системе и т.д.

Поскольку задача проектирования (синтеза) или модернизации сложной системы обычно решается на моделях, использующих упрощающие решение предположения и допущения, анализ синтезированной системы, выполняемый с целью определения фактической эффективности конкретных значений характеристик, обычно проводится на основе более детальных моделей, в качестве которых чаще всего используются имитационные или комбинированные (например, аналитико-имитационные) модели.

1.3. Модели сложных систем

1.3.1. Требования к моделям сложных систем

Ко всем разрабатываемым моделям сложных систем предъявляются два противоречивых требования [1]:

• простота модели;

• адекватность исследуемой системе.

Требование *простоты* модели обусловлено необходимостью построения модели, которая может быть рассчитана доступными методами. Построение сложной модели может привести к невозможности

получения конечного результата имеющимися средствами в приемлемые сроки и с требуемой точностью.

Степень сложности (простоты) модели определяется уровнем ее детализации, зависящим от принятых предположений и допущений: чем их больше, тем ниже уровень детализации и, следовательно, проще модель и, в то же время, менее адекватна исследуемой системе.

Одним из основных требований, предъявляемых к модели, является ее *адекватность* реальной сложной системе, которая достигается за счет использования моделей с различным уровнем детализации, зависящим от особенностей структурно-функциональной организации системы и целей исследования.

Адекватность (от лат. adaequatus – приравненный, равный) – соответствие модели оригиналу, характеризуемое степенью близости свойств модели свойствам исследуемой системы.

Адекватность математических моделей зависит от:

- степени полноты и достоверности сведений об исследуемой системе;
- уровня детализации модели.

При этом моделирование может проводиться:

- в условиях полной определенности, означающей наличие точной информации обо всех исходных параметрах;
- в условиях неопределенности, обусловленных:
  - неточностью сведений о параметрах;
  - отсутствием сведений о значениях некоторых параметров.

Процессы функционирования сложных технических систем во временном и надежностном аспектах определяются, в общем случае, на основе вероятностного подхода. Это обусловлено случайностью формирования нагрузки, моментов возникновения сбоев и отказов в технических средствах системы, случайностью времени восстановления работоспособности отказавших элементов. В связи с этим модели сложных дискретных систем со стохастическим характером функционирования строятся в терминах теории вероятностей, в частности, теории случайных процессов и теории массового обслуживания.

Математические модели позволяют прогнозировать эффект, достигаемый при изменении структурно-функциональных параметров сложной системы и параметров нагрузки.

Процессы функционирования реальных сложных систем обычно практически невозможно описать полно и детально, что обусловлено существенной сложностью таких систем. Основная проблема при разработке модели состоит в нахождении компромисса между простотой ее описания, что является одной из предпосылок понимания и возможности ее исследования аналитическими методами, и необходимостью учета многочисленных особенностей, присущих реальным системам.

Попытка построить единую универсальную модель сложной системы, несомненно, обречена на неудачу ввиду ее необозримости и невозможности расчета. Поэтому моделирование реальных систем часто реализуется на основе принципа иерархического многоуровневого моделирования, базирующегося на иерархическом описании систем и процессов, протекающих в них.

1.3.2. Принцип иерархического многоуровневого моделирования

Основная идея принципа ***иерархического многоуровневого моделирования*** (ИММ) заключается в том, что система и протекающие в ней процессы представляются семейством моделей, каждая из которых описывает поведение системы с точки зрения различных уровней абстрагирования, отличающихся рядом характерных особенностей и параметров, с помощью которых и описывается поведение системы.

Применительно к моделям сложных дискретных систем со стохастическим характером функционирования можно выделить два направления иерархии (рис. 1.3):

- ***иерархия по вертикали***, в которой деление моделей по уровням осуществляется в зависимости от структурно-функциональных особенностей исследуемых систем;
- ***иерархия по горизонтали***, в которой деление моделей по уровням осуществляется в зависимости от методов их исследования.

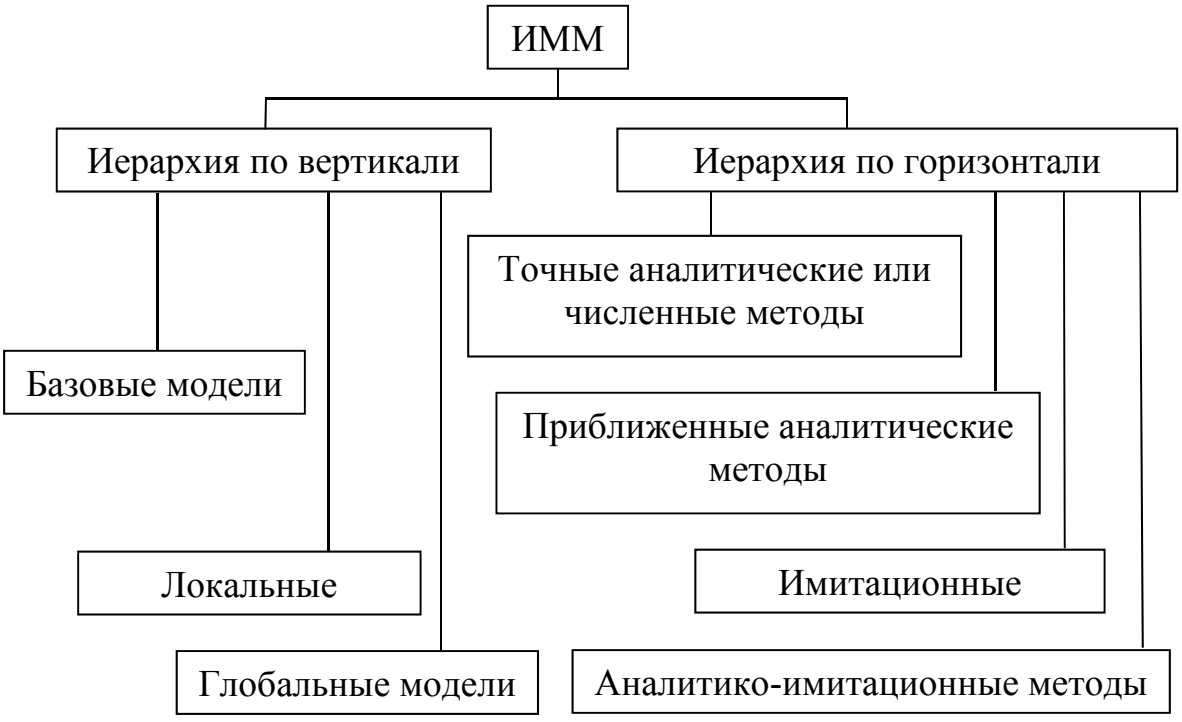


Рис. 1.3. Иерархическое многоуровневое моделирование

В ***иерархии по вертикали***, в общем случае, можно выделить три уровня моделей:

- уровень ***базовых моделей***, содержащий простейшие модели на основе которых строятся и могут быть рассчитаны другие более сложные модели второго и третьего уровней;
- уровень ***локальных моделей***, отображающих отдельные особенности структурно-функциональной организации систем, и позволяющих решать частные задачи анализа и синтеза;
- уровень ***глобальных моделей***, наиболее полно отображающих структурные и функциональные особенности организации исследуемых систем и представляющих собой модели с высокой степенью детализации.

Глобальные модели строятся на основе базовых и локальных моделей.

Иерархия по горизонтали включает следующие четыре уровня моделей в зависимости от методов их исследования:

- модели, поддающиеся *точному расчету*, позволяющему получить результаты либо аналитически в явном виде, либо численно с использованием численных методов анализа, например, для решения системы линейных алгебраических уравнений марковского случайного процесса;
- модели, поддающиеся *приближенному аналитическому расчету* с приемлемой для инженерных применений точностью, причем результаты могут быть получены либо в явном виде, либо в виде границ (верхней и нижней);
- модели, требующие применения *статистических методов расчета*, основанных на имитационном моделировании;
- модели, использующие *аналитико-имитационные расчеты*.

Базовые модели допускают применение точных и приближенных аналитических методов и позволяют получить результат в явном виде, локальные модели, кроме этого, обычно предполагают применение имитационных методов, а глобальные – наряду с перечисленными методами моделирования, могут использовать аналитико-имитационные методы.

На практике наиболее широко применяются модели, поддающиеся точному или приближенному аналитическому расчету. Имитационное моделирование обычно используется для аттестации приближенных методов и детального изучения свойств и закономерностей на моделях большой сложности с целью разработки на основе полученных результатов приближенных и эвристических методов расчета.

Взаимодействие моделей различных уровней иерархии осуществляется путем пересчета характеристик, полученных на одном уровне, в параметры модели, используемой на другом (соседнем) уровне. На каждом уровне может использоваться множество различных моделей. Состав моделей каждого уровня зависит от структурно-функциональной

организации сложной системы и целей исследования. Последнее также определяет степень детализации моделей одного и того же уровня.

1.3.3. Структурно-функциональная декомпозиция сложной системы

Реализация принципа иерархического многоуровневого моделирования базируется на методе структурно-функциональной декомпозиции исследуемой сложной системы, направленном на выделение и исследование наиболее существенных аспектов структурно-функциональной организации.

Структурно-функциональная декомпозиция сложных систем позволяет на разных этапах исследования использовать модели разных уровней: на этапе функционального проектирования – базовые модели, на этапе структурного проектирования – локальные модели и на завершающем этапе структурно-функционального проектирования – глобальные модели.

Такой подход позволяет существенно упростить решение задачи системотехнического проектирования реальных систем, характеризующейся значительной сложностью ввиду большой размерности задачи и громоздкости результатов.

1.3.4. Классификация моделей сложных систем

Многообразие систем, проявляющееся в многообразии их структурно-функциональной организации, определяет использование множества разных моделей, которые могут быть классифицированы в зависимости от:

- характера функционирования исследуемой системы:
  - детерминированные, функционирование которых описывается детерминированными величинами;
  - стохастические или вероятностные, функционирование которых описывается случайными величинами;
- характера протекающих в исследуемой системе процессов:
  - непрерывные, в которых процессы протекают непрерывно во времени;
  - дискретные, в которых процессы меняют свое состояние скачкообразно в дискретные моменты времени;
- степени достоверности исходных данных об исследуемой системе:
  - с априорно известными параметрами;
  - с неизвестными параметрами;
- режима функционирования системы:
  - стационарные, в которых характеристики не меняются со временем;
  - нестационарные, в которых характеристики изменяются со временем;

- назначения:
  - статические или структурные, отображающие состав и структуру системы;
  - динамические или функциональные, отображающие функционирование системы во времени;
  - структурно-функциональные, отображающие структурные и функциональные особенности организации исследуемой системы;
- уровня моделирования:
  - изобразительные (наглядные) – модели, применяемые на качественном уровне моделирования;
  - конструктивные – модели, применяемые на количественном уровне моделирования;
- способа представления (описания) и реализации:
  - концептуальные или содержательные, представляющие собой описание (в простейшем случае словесное) наиболее существенных особенностей структурно-функциональной организации исследуемой системы;
  - физические или материальные – модели, эквивалентные или подобные оригиналу (макеты) или процесс функционирования которых такой же, как у оригинала и имеет ту же или другую физическую природу;
  - математические или абстрактные, представляющие собой формализованное описание системы с помощью абстрактного языка, в частности с помощью математических соотношений, отражающих процесс функционирования системы;
  - программные (алгоритмические, компьютерные) – программы для ЭВМ, позволяющие наглядно представить исследуемый объект посредством имитации или графического отображения математических зависимостей, описывающих искомый объект.

Между классами систем и моделей необязательно должно существовать однозначное соответствие. Например, дискретные системы могут быть представлены в виде непрерывных моделей, а детерминированные системы – в виде вероятностных моделей, и наоборот.

1.4. Методы и инструментальные средства распределенного моделирования сложных систем

1.4.1. Методы распределенного моделирования сложных систем

При исследовании технических систем с дискретным характером функционирования наиболее широкое применение получили следующие методы математического моделирования [1]:

- ***аналитические*** (аппарат теории вероятностей, теории массового обслуживания, теории случайных процессов, методы оптимизации и т.п.);

• ***численные*** (применение методов численного анализа для получения конечных результатов в числовой форме, когда невозможно получить аналитические зависимости характеристик от параметров в явном виде);

• ***статистические*** или ***имитационные*** **(**исследования на ЭВМ, базирующиеся на методе статистических испытаний и предполагающие применение специальных программных средств и языков моделирования, таких как GPSS, SIMULA, SOL, ПЛИС, ИМСС и др.);

• ***комбинированные***.

Аналитические методы состоят в построении математической модели в виде математических символов и отношений, при этом требуемые зависимости выводятся из математической модели последовательным применением математических правил.

Достоинство аналитических методов заключается в возможности получения решения в явной аналитической форме, позволяющей проводить детальный анализ процессов, протекающих в исследуемой системе, в широком диапазоне изменения параметров системы. Результаты в аналитической форме являются основой для выбора оптимальной структурно-функциональной организации системы на этапе синтеза.

Недостаток аналитических методов – использование целого ряда допущений и предположений в процессе построения математической модели и невозможность в некоторых случаях получить решение в явном виде из-за неразрешимости уравнений в аналитической форме, отсутствия первообразных для подынтегральных функций и т.п. В этих случаях широко применяются численные методы.

Аналитические методы делятся на:

• точные;

• приближенные;

• эвристические.

Численные методы основываются на построении конечной последовательности действий над числами. Применение численных методов сводится к замене математических операций и отношений соответствующими операциями над числами, например, к замене интегралов суммами, бесконечных сумм конечными и т.п. Результатом применения численных методов являются таблицы и графики зависимостей, раскрывающих свойства объекта. Численные методы являются продолжением аналитических методов в тех случаях, когда результат не может быть получен в явном виде, и по сравнению с аналитическими методами позволяют решать значительно более широкий круг задач.

В тех случаях, когда анализ математической модели даже численными методами может оказаться нерезультативным из-за чрезмерной трудоемкости или неустойчивости алгоритмов в отношении погрешностей аппроксимации и округления, строится имитационная

модель, в которой процессы, протекающие в ВС, описываются как последовательности операций над числами, представляющими значения входов и выходов соответствующих элементов. Имитационная модель объединяет свойства отдельных элементов в единую систему. Производя вычисления, порождаемые имитационной моделью, можно на основе свойств отдельных элементов определить свойства всей системы.

При построении имитационных моделей широко используется *метод статистических испытаний* (метод Монте-Карло). Процедура построения и анализа имитационных моделей методом статистических испытаний называется ***статистическим моделированием***. Статистическое моделирование представляет собой процесс получения статистических данных о свойствах моделируемой системы.

Достоинством статистического моделирования является **универсальность**, гарантирующая принципиальную возможность построения моделей систем любой сложности с любой степенью детализации и, как следствие этого, возможность получения результатов исследования практически с любой наперед заданной точностью.

Недостаток статистического моделирования – **трудоемкость** процесса моделирования и **частный характер результатов**, не раскрывающий зависимости, а лишь определяющий ее в отдельных точках.

Статистическое моделирование широко используется для оценки погрешностей аналитических и численных методов.

Комбинированные методы представляют собой любую комбинацию выше перечисленных методов, в частности:

- ***численно-аналитические***, в которых часть результатов получается численно, а остальные – с использованием аналитических зависимостей;
- ***аналитико-имитационные***, представляющие собой имитационное моделирование в сочетании с аналитическими методами, позволяющими сократить время моделирования за счет определения значений ряда характеристик на основе аналитических зависимостей по значениям одной или нескольких характеристик, найденных путем статистической обработки результатов имитационного моделирования.

1.4.2. Инструментальные средства распределенного моделирования сложных систем

Инструментальные средства распределенного моделирования можно разделить на два класса:

- технические средства;
- программные средства.

В качестве ***технических средств*** математического моделирования обычно используются средства вычислительной техники: от персональных компьютеров – для исследования простейших моделей до больших вычислительных машин и компьютерных сетей – для исследования сложных моделей с применением распределенного моделирования.

В качестве ***программных средств*** распределенного моделирования могут быть использованы процедурно-ориентированные или проблемно-ориентированные алгоритмические языки, а также специализированные комплексы программ автоматизированного моделирования сложных систем.

1.5. Этапы моделирования сложных систем

Выделим ***основные этапы*** распределенного моделирования:

- формулирование исследуемой проблемы и целей моделирования;
- разработка концептуальной модели;
- разработка математической модели;
- параметризация модели;
- выбор метода моделирования;
- выбор средств моделирования;
- трансляция модели;
- верификация модели;
- валидация модели;
- стратегическое и тактическое планирование;
- проведение экспериментов на модели (экспериментирование);
- анализ результатов моделирования.

Упрощенная схема процесса распределенного моделирования приведена на рис. 1.4.

Рис. 1.4. Упрощенная схема процесса распределенного моделирования

Перечисленные этапы распределенного моделирования редко выполняются в строго заданной последовательности, начиная с определения проблемы и кончая документированием. В ходе распределенного имитационного моделирования могут быть сбои в прогонах модели, ошибочные допущения, от которых в последствие приходится отказываться, переформулировки целей исследования. То есть, на каждом этапе возможно возвращение назад, к предыдущим этапам. Именно такой итеративный процесс даёт возможность получить модель, которая позволяет принимать решения.

1.5.1. Формулирование исследуемой проблемы и целей распределенного моделирования

Существенное влияние на все последующие этапы моделирования оказывает этап формулирования исследуемой проблемы и определения целей моделирования, на котором:

- определяется *объект моделирования*, возможные способы его структурно-функциональной организации, условия функционирования;
- формулируются *задачи анализа и/или проектирования (синтеза)*, которые должны быть решены в процессе моделирования;
- конкретизируются наиболее важные *характеристики*, подлежащие исследованию, и формы представления результатов моделирования;
- формулируются *требования к качеству функционирования* в виде ограничений, налагаемых на характеристики системы, и формулируется *критерий эффективности*;
- определяются *требования к точности* получения результатов моделирования и форма их представления.

1.5.2. Разработка концептуальной модели сложной системы

Концептуальная (содержательная) модель – это абстрактная модель, выявляющая причинно-следственные связи, присущие исследуемой сложной системе и существенные для достижения целей моделирования.

Основное ***назначение концептуальной модели*** – выявление наиболее существенных аспектов структурно-функциональной организации, учет которых необходим для получения требуемых результатов. В концептуальной модели обычно в словесной форме приводятся сведения о природе и параметрах элементарных явлений исследуемой системы, о виде и степени взаимодействия между ними, о месте и значении каждого элементарного явления в общем процессе функционирования системы.

Одна и та же система может представляться различными концептуальными моделями, которые строятся в зависимости от целей исследования, сформулированных на предыдущем этапе. Например, одна

концептуальная модель может отображать временные аспекты функционирования системы, другая – надежностные, третья – масса-габаритные аспекты построения системы.

Построение концептуальной модели и ее формализация предполагает выполнение следующих этапов:

- ***постановка задачи моделирования***, включающая:
  - формулировку целей и обоснование необходимости моделирования;
  - определение номенклатуры показателей эффективности в зависимости от целей моделирования;
  - выбор методики моделирования с учетом имеющихся ресурсов;
  - оценку размерности задачи и определение возможности ее разбиения на подзадачи;
  - формирование требований к составу исходных параметров для проведения моделирования;
  - выдвижение гипотез и принятие предположений и допущений.
- ***определение исходных параметров*** и их описание, включающее:
  - выбор наименований и обозначений параметров;
  - выбор единиц измерения параметров;
  - установка диапазонов изменения параметров;
- ***формирование критерия эффективности***, предполагающее построение обобщенного показателя эффективности на основе множества частных, зачастую противоречивых, показателей с использованием одного из следующих подходов:
  - построение **составного критерия эффективности** в виде **аддитивного** F_1 или **мультипликативного** F_2 функционала:

$$F_1 = \sum_{i=1}^{N} \alpha_i x_i; \qquad F_2 = \frac{x_1 * \ldots * x_n}{x_{n+1} * \ldots * x_N},$$

    где $x_1, \ldots, x_N$ – частные показатели эффективности; α_i – весовой коэффициент показателя x_i;
  - выбор в качестве критерия эффективности F одного частного показателя при ограничениях, налагаемых на остальные показатели эффективности: $F = x_k$ при ограничениях $x_i < x_i^*$ или $x_i > x_i^*$ для всех $i \neq k$;
- ***описание концептуальной модели*** системы, предполагающее:
  - описание концептуальной модели в абстрактных терминах и понятиях;
  - описание модели с применением типовых математических схем;
  - окончательное принятие гипотез и предположений;
  - обоснование выбора процедуры аппроксимации реальных процессов при построении модели.
- ***проверка достоверности концептуальной модели***, включающая:
  - проверку логики построения и функционирования модели;
  - оценку достоверности исходной информации;

- рассмотрение постановки задачи моделирования;
- анализ принятых предположений и допущений;
- исследование гипотез и предположений.

1.5.3. Разработка математической модели сложной системы

Концептуальная модель служит основой для разработки математической модели сложной системы в терминах конкретного математического аппарата.

Создание математической модели преследует две ***основные цели***:

- дать формализованное описание структуры и процесса функционирования системы для однозначности их понимания;
- попытаться представить процесс функционирования системы в виде, допускающем аналитическое исследование системы с использованием методов и приемов, разработанных в рамках данного математического аппарата.

Выбор того или иного математического аппарата обусловлен физической природой исследуемой системы и процессов, протекающих в ней. Так, например, для исследования процессов функционирования дискретных систем применяется аппарат теории случайных процессов и теории массового обслуживания.

Основная проблема при создании модели заключается в нахождении компромисса между простотой модели, что является одной из предпосылок понимания и возможности ее эффективного исследования, и ее адекватностью исследуемой системе.

1.5.4. Процесс параметризации моделей сложных систем

Теоретические исследования сложных систем базируются на использовании моделей, отображающих объект исследования в форме, необходимой и достаточной для получения результатов, составляющих цель исследований [1].

Количественно любая модель, как и соответствующая ей система, описывается совокупностью величин, которые могут быть разбиты на параметры и характеристики. Состав параметров и характеристик модели определяется составом параметров и характеристик исследуемой системы и может в идеальном случае совпадать с ним. В общем случае составы параметров и характеристик модели и системы различаются, т.к. в первом случае они формулируются в терминах того математического аппарата, который используется при построении модели, а параметры и характеристики системы формулируются в терминах соответствующей прикладной области, к которой принадлежит система. Так как, в большинстве случаев, параметры и характеристики системы и модели различаются, их принято называть соответственно ***системными*** и ***модельными***.

В связи с тем, что состав и номенклатура системных и модельных параметров и характеристик, в общем случае, различается, возникает необходимость установления соответствия между значениями системных и модельных параметров и характеристик, которое выполняется на этапе ***параметризации*** модели сложной системы.

1.5.5. Выбор метода распределенного моделирования

Математическое моделирование сложных систем предполагает применение следующих методов, подробно описанных в п. 1.4 настоящего пособия:

- аналитических;
- численных;
- статистических (имитационных);
- комбинированных.

Выбор конкретного метода распределенного моделирования зависит от многих факторов, в том числе от:

- целей моделирования;
- сложности исследуемой системы;
- сложности модели, определяемой выбранным уровнем ее детализации;
- требований к номенклатуре исследуемых характеристик;
- требований к точности получаемых результатов;
- требований к общности получаемых результатов;
- требований к затратам времени на моделирование;
- требований к материальным затратам;
- наличия специальных технических средств для проведения моделирования;
- квалификации специалиста, проводящего моделирование и т.д.

Результаты сравнительного анализа методов распределенного моделирования, выполненного на качественном уровне, представлены в табл. 1.1 (фигурными скобками отмечены наилучшие значения каждого показателя).

Таблица 1.1

Сравнительный анализ методов распределенного моделирования

Метод моделирования	Слож-ность метода	Общ-ность рез-тов	Точ-ность рез-тов	Затра-ты време-ни	Матер. затра-ты	Задачи син-теза
Аналитический	{+}	{++++}	+	{+}	{+}	{+}
Численный	++	+++	++	++	++	++
Имитационный	+++	+	{++++}	++++	++++	++++
Комбинированный	++++	++	+++	+++	+++	+++

На практике обычно используется подход, при котором разрабатываются одновременно одна или несколько аналитических и имитационных моделей, при этом имитационные модели применяются как для оценки погрешностей приближенных аналитических моделей, так и для детального анализа синтезированной на основе приближенных аналитических моделей оптимальной системы.

1.5.6. Выбор инструментальных средств распределенного моделирования и разработка программной модели

На данном этапе выбираются технические и программные средства распределенного моделирования для проведения исследований сложной системы на программной модели.

Технические и программные средства моделирования выбираются с учетом ряда критериев, основными среди которых являются ***достаточность*** и ***полнота*** средств для реализации концептуальной и математической модели. Среди других критериев можно назвать ***доступность*** средств, ***простоту и легкость освоения*** технических и программных средств моделирования, ***скорость и корректность создания*** программной модели, ***существование методики использования средств*** для моделирования сложных систем определенного класса.

После выбора средств моделирования разрабатывается ***программная модель*** сложной системы. Этот процесс включает:

- разработку алгоритма;
- конкретизацию форм представления исходных данных и результатов;
- написание и отладку программы.

1.5.7. Верификация модели

Верификация модели (model verification) – процесс контроля корректности трансляции концептуальной модели в программную модель. На этапе верификации устанавливается правильность программной («машинной») модели и определяется, соответствует ли программная модель сложной системы замыслу разработчика, т.е. концептуальной модели.

Верификация имитационной модели есть проверка соответствия ее поведения предположениям экспериментатора. Когда модель организована в виде вычислительной программы для компьютера, то сначала исправляют ошибки в ее записи на алгоритмическом языке, а затем переходят к верификации.

Это первый этап действительной подготовки к имитационному эксперименту. Подбираются некоторые исходные данные, для которых могут быть предсказаны результаты просчета. Если окажется, что ЭВМ

выдает данные, противоречащие тем, которые ожидались при формировании модели, значит, модель неверна, т.е. она не соответствует заложенным в нее ожиданиям. В обратном случае переходят к следующему этапу проверки работоспособности модели – ее *валидации*.

1.5.8. Валидация модели

Валидация модели (model validation) –это процесс проверки того, является ли модель, допустимым представлением реальной системы, основываясь на целевой направленности модели.

Валидация имитационной модели есть проверка соответствия данных, получаемых в процессе машинной имитации, реальному ходу явлений, для описания которых создана модель. Валидация производится тогда, когда экспериментатор на предшествующей стадии (верификации) убедился в правильности структуры (логики) модели. Валидация модели состоит в том, что выходные данные после расчета на компьютере сопоставляются с имеющимися статистическими сведениями о моделируемой системе.

Существует несколько точек зрения на валидацию:

- валидная модель может быть использована для принятия решений, сходных с теми, которые были бы приняты на реальной и недорогой системе;
- сложность процесса валидации зависит от сложности моделируемой системы, а так же от того, существует ли реальная система, так как обычно для построения и валидации модели можно собирать данные о существующей системе;
- модель сложной системы может быть лишь *аппроксимацией* реальной системы, не зависимо от того, как много времени и средств потрачено на ее создание.
- модель всегда должна разрабатываться для конкретного набора задач: фактически модель, валидная для одной задачи, может не быть валидной для другой задачи;
- валидация *не должна* осуществляться после окончания разработки модели, при условии наличия времени и средств;
- каждый раз, когда модель применяется к другой задаче, необходимо перепроверять валидность данной модели, так как задача может существенно отличаться от первоначальной, либо параметры модели могут измениться.

Валидация модели обычно выполняется на различных уровнях (например, на уровне входных данных, элементов модели, подсистем и их взаимосвязи).

Существуют специальные методы валидации (например, путём оценивания чувствительности выходных данных к изменению значений входных), различные парадигмы, подходы и методики.

Валидация – это процесс, который называется в русской литературе по моделированию, – проверка адекватности модели.

Проверка адекватности модели исследуемой сложной системе заключается в анализе ее соответствия исследуемой системе, проявляющегося в близости значений модельных и системных характеристик.

Отличие модели от исследуемой системы связано с тем, что обычно модель является упрощенным и идеализированным отображением исследуемой системы, которое обусловлено:

- идеализацией внешних условий и режимов функционирования;
- не учетом в модели несущественных по мнению исследователя факторов и параметров;
- отсутствием точных сведений о внешних воздействиях и о некоторых конкретных нюансах организации системы;
- введением ряда упрощающих предположений и допущений.

Мерой адекватности модели исследуемой системе может служить абсолютное Δ или относительное δ отклонение модельных характеристик $\mathbf{H}_M$ от системных $\mathbf{H}$: $\Delta = |\mathbf{H}_M - \mathbf{H}|$; $\delta = |\mathbf{H}_M - \mathbf{H}| / \mathbf{H} = \Delta / \mathbf{H}$. Тогда критерием адекватности может служить вероятность того, что отклонение Δ не превышает некоторого предельного значения Δ^*: $\pi = \mathbf{Pr}\ (\Delta<\Delta^*)$.

Однако применение данного критерия на практике затруднено, а во многих случаях и невозможно по следующим причинам:

- для проектируемых или модернизируемых систем обычно заранее неизвестны значения системных характеристик H;
- система в большинстве случаев оценивается по множеству системных характеристик, которые могут иметь разные значения отклонений Δ;

На практике валидация модели обычно проводится путем экспертного анализа разумности результатов моделирования.

Можно выделить следующие ***этапы валидации модели***:

- проверка элементов модели и правильности формирования значений их параметров, особенно задаваемых в виде случайных величин;
- проверка адекватности формирования нагрузки в модели;
- проверка концептуальной модели с целью выявления ошибок постановки задачи;
- проверка математической модели с целью выявления ошибок математического описания структурно-функциональной организации системы и нагрузки;
- оценка точности приближенных аналитических методов расчета характеристик модели;
- проверка программной модели с целью выявления логических ошибок в алгоритме и инструментальных ошибок в программе.

В случае выявления неадекватности модели исследуемой системе необходимо выполнить корректировку или калибровку модели, которая может быть: глобальной и локальной.

Глобальная корректировка заключается в разработке новой модели и необходима при наличии:

- ошибок в постановке задачи моделирования;
- методических ошибок в концептуальной или математической модели.

Локальная корректировка может состоять:

- в уточнении параметров модели;
- в изменении метода расчета характеристик;
- в разработке более детализированной математической модели;
- в изменении программной модели.

В процессе проверки адекватности модели необходимо определить область применения модели, т.е. оценить диапазон изменения параметров, при котором точность результатов моделирования находится в допустимых пределах.

1.5.9. Эксперименты на моделях

Исследование сложных систем на моделях заключается в проведении экспериментов, в процессе которых определяются характеристики системы при разных значениях структурно-функциональных параметров и параметров нагрузки. Большая номенклатура исходных параметров и широкий диапазон их изменения требует предварительного ***планирования*** выполняемых на модели экспериментов (расчетов).

Планирование направлено на уменьшение длительности эксперимента при условии обеспечения достоверности и полноты результатов моделирования.

Особую значимость планирование экспериментов приобретает при использовании методов распределенного имитационного моделирования, характеризующихся большими затратами ресурсов вычислительных систем и компьютерных сетей в процессе моделирования.

Одной из основных проблем распределенного имитационного моделирования является нахождение компромисса между временем моделирования и затратами памяти вычислительной системы, на которой проводится моделирование, связанного с тем, что имитационное моделирование предъявляет повышенные требования как к производительности, так и к памяти вычислительной системы для проведения имитационных экспериментов. Время, затрачиваемое на проведение одного эксперимента с моделью средней сложности, даже на высокопроизводительных вычислительных системах и компьютерных сетях может достигать нескольких десятков минут и, в некоторых случаях, нескольких часов, а потребность в оперативной памяти – до нескольких десятков и сотен мегабайт. Причем с увеличением числа проводимых

имитационных экспериментов соответственно возрастает время моделирования. Все это обусловливает высокую стоимость имитационного моделирования и требует тщательного планирования имитационных экспериментов с целью сокращения затрат на распределенное моделирование.

При проведении распределенного имитационного моделирования используются два способа планирования:

- ***стратегическое планирование***, состоящее в выборе определенных сочетаний параметров и последовательности проведения экспериментов с использованием методов теории планирования экспериментов;
- ***тактическое планирование***, направленное на уменьшение времени выполнения одного эксперимента при обеспечении статистической достоверности результатов имитационного моделирования.

Подробное изложение вопросов планирования имитационных экспериментов можно найти в [5].

1.5.10. Анализ результатов распределенного моделирования

Анализ результатов моделирования направлен на выявление свойств, присущих исследуемой сложной системе, и включает в себя следующие этапы:

- обработка результатов для удобства последующего анализа и использования; на этом этапе выделяются наиболее важные с точки зрения исследователя и с учетом целей исследования результаты, которые представляются в форме, наиболее удобной для изучения свойств исследуемой системы;
- определение зависимостей характеристик от параметров системы путем варьирования исходных параметров структурно-функциональной организации и нагрузки с целью выявления и формулирования свойств исследуемой системы;
- принятие решения о работоспособности исследуемой системы и выработка рекомендаций по наиболее эффективной и рациональной организации проектируемой или модернизируемой системы, которые могут быть использованы в дальнейшем при решении задач синтеза в процессе системотехнического проектирования.

1.6. Резюме

Объектами распределенного моделирования в технике, в общем случае, являются ***сложные системы и комплексы***, обладающие структурной и функциональной организацией. Структура сложной системы может быть задана в графической или аналитической форме. Функция сложной системы может быть задана в алгоритмической, аналитической, графической или табличной форме.

Сложной системе присущи такие свойства как ***целостность***, ***связность***, ***организованность*** и ***интегративность***. Наличие этих свойств означает, что систему нельзя рассматривать как простую совокупность элементов, поскольку, изучая каждый элемент системы в отдельности, нельзя познать все свойства системы в целом.

Распределенное моделирование направлено на решение задач ***анализа***, связанных с оценкой эффективности сложных систем, и проектирования или модернизации (***синтеза***), направленных на построение оптимальных систем в соответствии с выбранным критерием эффективности. ***Эффективность*** сложной системы задается в виде совокупности показателей эффективности, каждый из которых служит мерой одного свойства системы. Мера эффективности, обобщающая все или некоторые, наиболее существенные, свойства сложной системы в одной оценке называется ***критерием эффективности***.

Для количественного описания сложной системы используются ***параметры***, описывающие первичные свойства системы, и ***характеристики***, определяемые в процессе решения задач анализа как функция параметров.

Множество параметров технических систем можно разделить на ***внутренние*** (структурные и функциональные) и ***внешние*** (нагрузочные и параметры внешней среды). Параметры могут быть ***детерминированными*** или ***случайными*** и ***управляемыми*** или ***неуправляемыми***.

Основными характеристиками сложных технических систем являются характеристики ***производительности***, ***оперативности***, ***надежности*** и ***стоимости***.

Изучение сложных систем удобно проводить в терминах процессов, с которыми связаны такие понятия как состояние, переход из одного состояния в другое и событие.

К разрабатываемым моделям обычно предъявляются два противоречивых требования: ***простота*** и ***адекватность*** исследуемой системе.

Модели могут быть классифицированы в зависимости от ***характера функционирования исследуемой системы*** (детерминированные и стохастические или вероятностные), от ***характера протекающих в исследуемой системе процессов*** (непрерывные и дискретные), от ***режима функционирования системы*** (стационарные и нестационарные), от ***способа представления и реализации*** (концептуальные или содержательные; физические или материальные; математические или абстрактные; программные или компьютерные).

Одним из важнейших этапов при разработке модели является ***параметризация***, заключающаяся в установлении соответствия между значениями системных и модельных параметров и характеристик.

Моделирование, как многоэтапный процесс исследования сложных систем, в общем случае предполагает решение следующих

взаимосвязанных задач: разработка модели, анализ характеристик системы, синтез системы, детальный анализ синтезированной системы.

При исследовании сложных технических систем с дискретным характером функционирования наиболее широкое применение получили аналитические, численные, статистические (имитационные) и комбинированные методы распределенного моделирования.

При использовании распределенного имитационного моделирования, необходима ***валидация*** всех имитационных моделей, иначе решения, принятые на основе этих моделей, будут неверными.

Ниже приводятся наиболее важные идеи разработки валидных моделей:

- точная формулировка проблемы;
- проведение интервью с экспертами в данной предметной области;
- постоянное взаимодействие лица, принимающего решения с участниками моделирования, что гарантирует корректность решаемой задачи, а также увеличивает надежность модели;
- разработка и документирование концептуальной модели;
- структурированный просмотр концептуальной модели (если не существует реальной системы, то это может быть единственным методом валидации);
- применение анализа чувствительности для определения наиболее важных (существенных) параметров системы;
- использование теста Тьюринга для сравнения выходных данных модели и системы;
- проверка результатов работы системы и анимации на корректность.

Модель является ***валидной*** для определенной задачи, если корректна логика ее работы и используются соответствующие данные.

Раздел 2. Принципы распределенного моделирования

2.1. Общие принципы распределенного моделирования сложных систем

2.1.1. Причины перехода к распределенному моделированию

В настоящее время наметилась тенденция перехода от последовательного имитационного моделирования к параллельному и распределённому имитационному моделированию [1,2,3,4].

Параллельное имитационное моделирование предполагает, что вычислительный эксперимент с имитационной моделью выполняется на нескольких процессорах многопроцессорной ЭВМ. Это может быть многопроцессорная ЭВМ с MPP архитектурой или многопроцессорная ЭВМ с общей памятью (рис. 2.1).

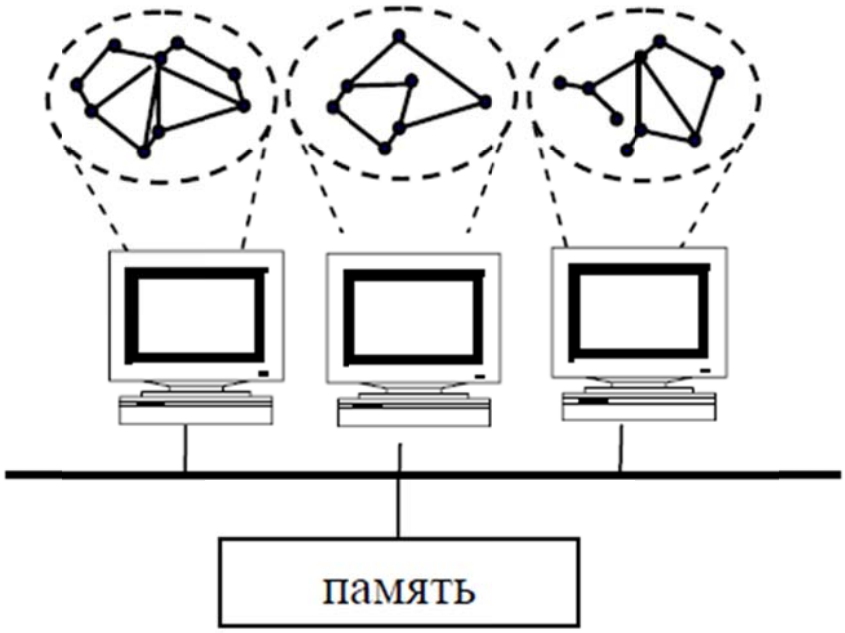


Рис. 2.1. Выполнение имитационной модели на нескольких процессорах многопроцессорной ЭВМ

Выполнение вычислительного эксперимента с имитационной моделью на нескольких компьютерах, объединённых в сеть (локальную или глобальную), можно рассматривать как распределённое имитационное моделирование (рис.2.2).

Причины перехода к параллельному и распределённому имитационному моделированию обусловлены, по мнению R. Fujimoto, следующими факторами [10].

Исследователь, который использует ресурсы N процессоров многопроцессорной ЭВМ (или N компьютеров, объединённых в сеть), в конечном счёте, пытается добиться того, чтобы время выполнения имитационного эксперимента сократилось в N раз.

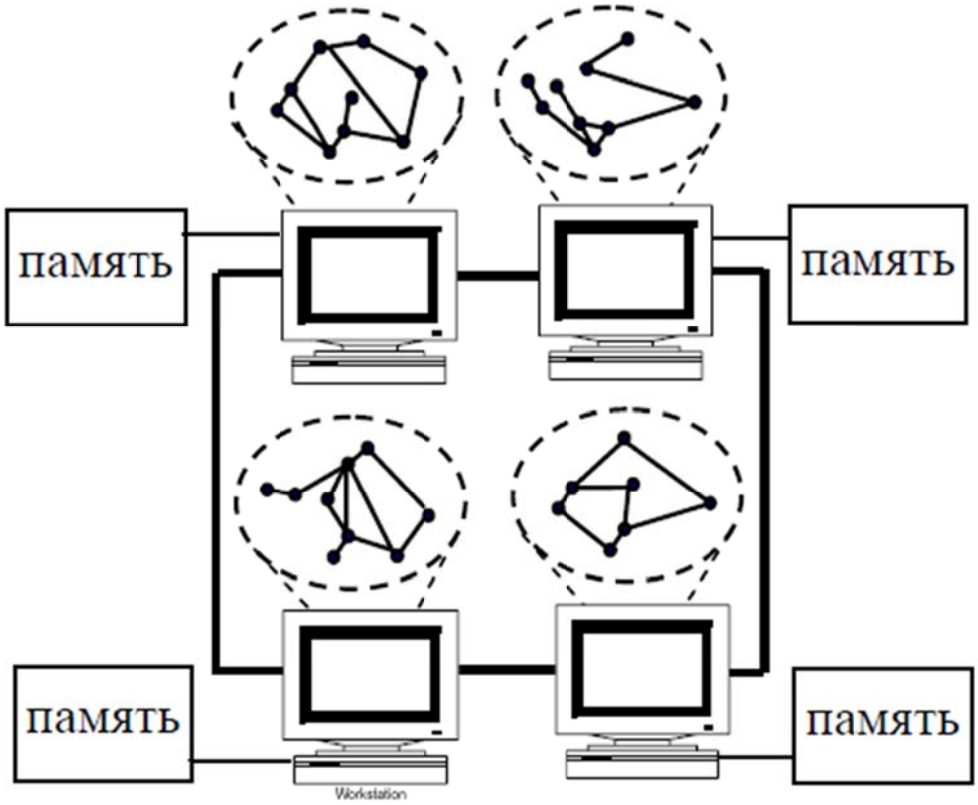


Рис. 2.2. Выполнение имитационной модели на нескольких узлах компьютерной сети

Задачи, которые стоят перед имитационным моделированием, становятся всё более сложными и требуют не только больших временных затрат, но и больших ресурсов памяти. Исследователь, использующий ресурсы нескольких процессоров или компьютеров предполагает, что локальная память этих процессоров или компьютеров также будет использована для решения его задачи.

Другим важным стимулом использования распределённого моделирования является необходимость объединения нескольких имитационных систем в одну распределённую среду имитационного моделирования. Примером может служить объединение нескольких тренажёров, имитирующих, управление танком, самолётом и других имитаторов (военные приложения). Их объединяют с целью создать распределённую виртуальную среду для обучения человека действиям по возможным сценариям в нештатных ситуациях. Другим примером может служить необходимость в объединении нескольких изучаемых подсистем (их моделей) в одну с той целью, чтобы можно было проанализировать их взаимодействие. Так, например, моделирование транспортной инфраструктуры города следует рассматривать неразрывно с подсистемой его электроснабжения и с другими коммуникационными инфраструктурами. В данном случае речь идёт о том, что гораздо более экономичным является подход, когда имитационные модели (в военных

приложениях или при моделировании инфраструктур) объединяются с помощью дополнительных программных средств (HLA, High Level Architecture). В противном случае возникает необходимость в разработке новой имитационной модели.

Ещё одной причиной для перехода к распределённому моделированию является возможность работы нескольких исследователей над одним и тем же имитационным проектом через Интернет. Проект может выполняться на суперкомпьютере или на вычислительной системе с кластерной архитектурой и, таким образом, исследователи используют вычислительные ресурсы суперкомпьютера, наблюдают за ходом имитационного эксперимента и просматривают результаты, находясь в разных городах или странах (рис. 2.3).

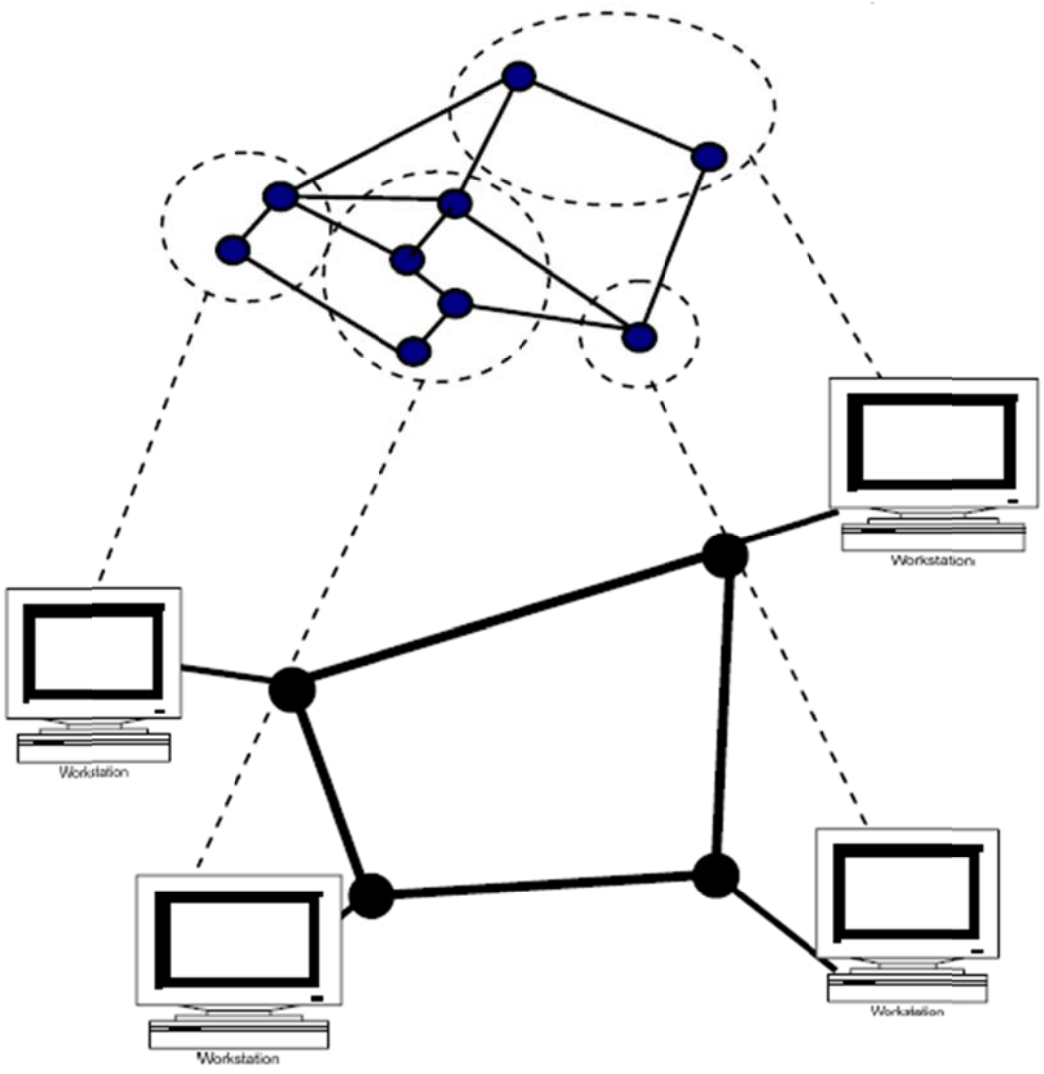


Рис. 2.3. Выполнение имитационной модели в глобальной сети

Кроме того, используя распределённую систему моделирования с удалённым доступом через Интернет, можно сдать в «аренду» неиспользуемые в данный момент времени вычислительные ресурсы [10].

Не следует забывать о таком преимуществе использования распределённых систем, как надёжность: под надёжностью будем подразумевать продолжение выполнения имитационного эксперимента, несмотря на то, что какой либо из процессоров (компьютеров) выходит из строя. В этом случае его работа распределяется на другие процессоры.

2.1.2. История развития распределенного моделирования

Распределённое моделирование развивалось достаточно независимо в трёх различных областях знаний:

- в высокопроизводительных вычислениях;
- в военных приложениях;
- в игровых приложениях через Internet.

Начало развития распределённого моделирования в области высокопроизводительных вычислений можно датировать концом 70-ых, началом 80-ых годов двадцатого века. Научные изыскания сосредоточились вокруг разработки ***алгоритмов синхронизации*** или ***алгоритмов управления временем***, как их называют в настоящее время. Алгоритмы синхронизации были призваны синхронизировать компоненты моделирования, которые распределялись по различным узлам вычислительной системы с целью ускорить последовательный процесс моделирования (то есть получить те же самые результаты, что и в последовательном моделировании, но за более короткий срок). Алгоритмы синхронизации поддерживали ***консервативную парадигму***. Консервативная парадигма основана на сдерживании процессов, которые выполняются параллельно на различных вычислительных узлах компьютерной сети (или мультипроцессорной ЭВМ). Консервативная парадигма использует блокирующий механизм, который не допускает возникновения ошибок синхронизации (обработка событий должна выполняться в строго определённом порядке).

Первые алгоритмы синхронизации были разработаны в конце 70 годов. Они описаны в работах (Chandy и Misra, 1978 [8]) и (Bryant, 1977 [9]). Авторы этих работ впервые сформулировали проблемы, которые возникают в распределённом имитационном моделировании и предложили первые решения этих проблем. Эти алгоритмы относились к классу ***консервативных алгоритмов***.

В начале 80-ых годов 20-го века появились работы Джефферсона (Jefferson) и других авторов. В них излагались принципы работы Time Warp алгоритмов (алгоритмов деформации времени [8]). ***Алгоритмы деформации времени*** (Time Warp алгоритмы) легли в основу оптимистических алгоритмов синхронизации. Многие последующие работы, проводимые впоследствии в области параллельного и распределённого моделирования, основаны на этих фундаментальных исследованиях.

В области военных приложений первые исследования по распределённому моделированию были проведены в связи с разработкой проекта SIMNET (SIMulator NETwork). Если в области высокопроизводительных вычислений исследования были направлены на сокращение времени выполнения имитационного эксперимента, то исследования в области военных приложений были направлены на объединение моделей-тренажёров. Объединение тренажёров позволяло создать новую виртуальную среду для проведения тренинга персонала. Разработанные в этих целях программные средства поддерживали интероперабельность и переиспользование кода.

Исследования в области военных приложений привели к разработке стандартов, на основании которых осуществлялось объединение различных тренажёров. К этим стандартам относится DIS (Discrete Interactive Simulation), стандарт (IEEE Std 1278.1-1995 1995).

В 1990-ых годах появился стандарт Aggregative Level Simulation Protocol (ALSP). В этом стандарте использовались концепции интероперабельности и переиспользования кода из проекта SIMNET.

Оба стандарта (DIS и ALSP) были заменены технологией HLA (High Level Architecture). Первоначально предполагалось применять HLA для реализации систем моделирования в военных приложениях, в настоящее время – для систем моделирования, используемых для обучения, анализа и тестирования персонала.

Третьим направлением применения распределённого моделирования являются сетевые игры (игроки взаимодействую через Internet).

2.1.3. Направления в развитии распределенного моделирования

Развитие распределенного моделирования идет по двум основным направлениям.

Первое направление – широко используемое ***параллельное дискретное событийно-ориентированное моделирование PDES*** (Parallel Discrete Event Simulation). Для создания новых высокоэффективных систем моделирования в этом случае используют спецпроцессоры для параллельного моделирования, сопутствующие языки, библиотеки и инструментальные средства. Примерами этого подхода служат такие системы как TeD/GTW [3], SPEEDES [3], Task-Kit [4] и др. В нашей стране тоже ведутся работы по созданию распределённых систем имитации (Мера (Новосибирск), Диана (МГУ)). Имитационные модели тесно связаны со средой, для которой они были разработаны, и это создает проблемы при смене окружения, например, при переносе системы на другую платформу.

Вторая парадигма, которая появилась в распределенном моделировании – это ***объединение разнородных систем моделирования***. В этом случае компонентами имитационного моделирования являются не объекты (как это делается в последовательном моделировании), не логические процессы (PDES), а сами имитационные модели. Таким

образом, задача этих проектов создать окружение для взаимодействия уже ранее созданных имитационных моделей. Этот подход используется в таких системах как DIS (Distributed Interactive Simulation), ALSP (Aggregate Level Simulation Protocol), HLA (High Level Architecture или «Архитектура высокого уровня»).

В настоящее время широко используется технология HLA. Компонентами имитационного моделирования в этом случае являются федераты, а объединение федератов называют федерацией. Федераты одной федерации могут быть разнородными (это могут быть имитационные модели, тренажёры, программа для сбора данных и т.д.). Обмен данными между федератами и исполнение федератов в едином модельном времени выполняется с помощью программной оболочки RTI (Run Time Infrastructure).

2.2. Декомпозиция модели сложной системы на компоненты

Одним их направлений развития распределенного моделирования является переход от разработки отдельных моделей к разработке их комплексов [10]. Комплекс моделей распределенного моделирования (распределенный модельный комплекс) позволяет объединять имитационные модели, находящиеся на разных компьютерах. Обсудим вопросы выбора архитектуры распределенного модельного комплекса; способы взаимодействия его распределенных частей; концепцию распределенного моделирования с применением распределенного модельного комплекса.

2.2.1. Компонент как основная составляющая распределенной модели сложной системы

Понятия ***комплекса*** и ***компонента*** – это связанные между собой понятия, способные переходить друг в друга. Так, ***комплекс***, внутренняя структура которого сложна и образована множеством компонентов, извне может восприниматься как единый ***компонент***. В то же время, любой из компонентов может обладать внутренней структурой, образующей комплекс.

Известно, что реальные сложные системы, являющиеся предметом моделирования, в зависимости от определенных сочетаний своих характеристик и характеристик окружающего мира, способны порой достаточно резко менять свое поведение. Для отражения этого их свойства введем формализованное определение события. Будем считать, что события могут происходить в самом начале интервала модельного времени Δt , при переходе системы от момента t_i к моменту t_{i+1} . С моделью может быть связано несколько событий. Событие k, k =1,...K состоит в том, что некая скалярная функция $F_k(\vec{x}, \vec{a})$ обращается в нуль на наборе характеристик модели $\overrightarrow{x_l}, \overrightarrow{a_l}$ в начале очередного интервала моделирования: $F_k(\vec{x}, \vec{a}) = 0$. Будем считать, что имеется некоторое

количество способов поведения модели, задающееся различными алгоритмами $\overrightarrow{f_m}(\vec{x}, \vec{a}, \Delta t), m = 1, \dots, M$, и однозначное отображение множества 2^K всевозможных реализаций всех связанных с моделью событий, во множество M различных алгоритмов ее поведения, которое и задает реакцию модели на произошедшее событие.

Заметим, что, начав с системно-динамического описания компонента модели, мы пришли к двойственному, объектно-ориентированному описанию: компонент, определенный выше, задает класс объектов со свойствами $\overrightarrow{x_i}, \overrightarrow{a_i}$, методами $\overrightarrow{f_m}(\vec{x}, \vec{a}, \Delta t), m = 1, \dots, M$, переключения между методами задаются событиями $F_k(\vec{x}, \vec{a}) = 0$, где $k = 1, \dots, K$.

Возникает задача объединения уже существующих и эксплуатируемых моделей (компонентов) с наименьшими затратами как человеческих, так и машинных ресурсов. Ситуация, когда взаимосвязанными из-за необходимости повышать глубину и точность прогноза оказываются только два процесса, является простейшей.

2.2.2. Синхронизация во времени работы компонентов распределенной модели сложной системы

Обсудим теперь вопрос о синхронизации во времени работы компонентов (подмоделей). Предлагается следующий механизм синхронизации модельного времени компонент комплекса.

Из соображений выбора масштаба и точности моделирования определяется «стандартный» интервал моделирования Δt.

На интервале моделирования Δt постулируется механизм аппроксимации значений векторов $\vec{x}, \vec{a}$, позволяющий по известным их значениям $(\vec{x}_i, \vec{a}_i)$ и $(\vec{x}_{i+1}, \vec{a}_{i+1})$ на концах этого интервала вычислить их значения в любой промежуточной точке интервала. Это может быть линейная или какая-либо другая аппроксимация. Выбор того или иного способа аппроксимации зависит от масштаба моделирования и желаемой точности модели.

Для каждого события каждой подмодели ищется решение t уравнения $F_k(\vec{x}(t), \vec{a}(t)) = 0, \ k = 1, \dots, K$ на интервале Δt. Если таких решений нет, в качестве следующего интервала моделирования снова выбирается стандартный интервал моделирования Δt, если же решения есть, то наименьшее из них будет правым концом следующего интервала моделирования. Таким образом, интервал моделирования может уменьшаться с тем, чтобы ни одна из подмоделей не «проскочила» очередное событие, которое может существенно повлиять на ее поведение. Необходимо избегать так называемой «дурной бесконечности» событий, руководствуясь принципом достаточности конечного числа событий на любом конечном интервале времени, сформулированном в [2], т.е. точек накопления событий на конечном интервале времени не должно быть.

В заключение следует отметить, что комплекс, изнутри состоящий из многих компонентов, вовне может проявляться в качестве единого компонента. Введем следующую операцию объединения компонентов комплекса.

1. Внутренними переменными комплекса объявляется объединение внутренних переменных всех его компонентов.

2. Методами комплекса объявляется объединение всех методов его компонентов.

3. Событиями комплекса объявляется объединение всех событий его компонентов.

4. Внешними переменными комплекса объявляется объединение всех внешних переменных его компонентов, из которого исключаются все те переменные a_p^i, для которых строка с номером p одной из матриц коммутации $Q_{ij}, i \geq 1, j \leq N; i \neq j$, содержит единицу ($N$ – число процессов, описываемых компонентами комплекса распределенного моделирования).

Мы видим, что тогда как изнутри комплекс имеет сложный многокомпонентный состав, извне он, в соответствии с данным в начале работы определением, вполне может быть воспринят как единый компонент. Единственное отличие от данного выше определения компонента – возможность параллельного (т.е. одновременного) выполнения нескольких методов. Это отличие можно снять, например, разрешив компоненту одновременно выполнять несколько потоков (процессов), состоящих в чередовании некоторого числа выполняемых последовательно методов, как это было в MISS [2]. Так или иначе, определяющим остается то что, задавая для комплекса, рассматриваемого как единый компонент, начальные данные и значения внешних переменных, можно однозначно определить его внутренние переменные на любом шаге моделирования.

Таким образом, любая модель может быть представлена, с одной стороны, агрегировано, как единый компонент, а с другой стороны, подробно, причем с произвольной степенью подробности, как комплекс компонентов. В комплексе каждый из компонентов в свою очередь допускает представление в виде комплекса, тем самым позволяя реализовывать концепцию иерархического моделирования, т.е. построения семейства моделей разной степени детализации для изучения одного и того же явления.

2.3. Проблемы распределенного моделирования непрерывных и гибридных сложных систем

С ростом объема и сложности моделей систем, а также появлением специфических требований к самим моделям и процессу моделирования, возникла необходимость в распределенном моделировании, когда исполнение различных компонентов системы проводится на многопроцессорной системе или на различных компьютерах сети.

Распределенное моделирование позволяет добиться повышения производительности, достичь инкапсуляции внутренней логики работы компонентов, а также создает предпосылки для повторного использования уже созданных компонентов в моделях новых систем и возможности совместной работы компонентов, созданных различными производителями.

Рассмотрим проблемы распределенного моделирования ***непрерывных*** и ***гибридных*** систем, т.е. систем, в которых существенную роль играют как дискретные, так и непрерывные переменные (скорость, координата, стоимость и т.п.).

2.3.1. Дискретизация значений непрерывной переменной по времени

Большинство стандартов распределенного моделирования разработано для систем с дискретным поведением (дискретных систем). В случае необходимости какому-либо объекту демонстрировать непрерывное поведение, наиболее простым и очевидным решением является дискретизация значений непрерывной переменной по времени. Однако такое решение проблемы не является удовлетворительным в случае наличия сложных зависимостей между распределенными объектами по непрерывным переменным, что может привести к демонстрации моделью неадекватного поведения (неустойчивость изначально стабильных систем, запаздывание реакции и т.п.).

Особенно сильно эти проблемы проявляются при распределенной симуляции гибридных систем, где дискретное и непрерывное поведение зависят друг от друга. Например, система из трех компонентов, уравнения динамики которых (линейные дифференциальные уравнения второго порядка) связаны по непрерывным переменным, при «локальном» (не распределенном) моделировании демонстрирует устойчивое поведение (рис. 2.4). В то же время при распределенном моделировании при равномерной по времени дискретизации значений непрерывных переменных начиная с определенных значений шага дискретизации, та же система начинает демонстрировать неустойчивое поведение, расходится (рис. 2.5).

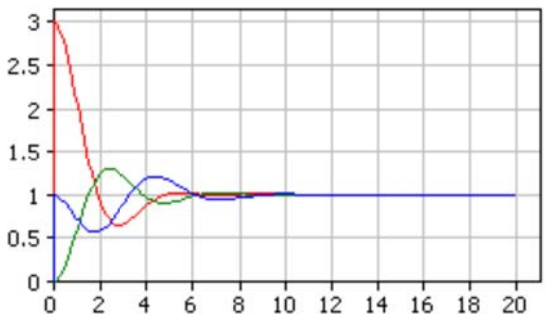


Рис. 2.4. Локальное моделирование системы

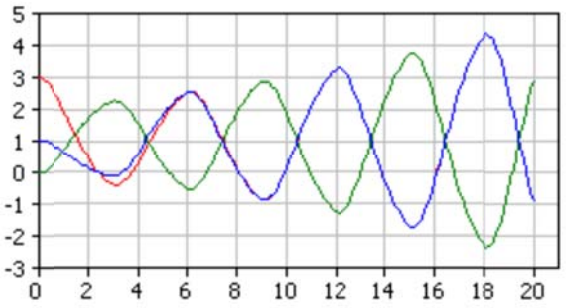


Рис. 2.5. Распределенное моделирование системы

Причиной получения такого результата послужила принципиальная невозможность совместного решения системы дифференциальных уравнений для всех компонентов системы при распределенном моделировании, когда каждое из уравнений решается независимо от других, а значения переменных обновляются в дискретные моменты времени.

2.3.2. Разрыв связи компонентов по непрерывным переменным

В процессе построения модели системы происходит определение ее структуры (декомпозиция), выделение компонентов и их описание на выбранном языке представления. Для обмена информацией между компонентами устанавливаются связи.

В случае моделирования гибридной системы часть связей может соединять компоненты по непрерывным переменным (например, координатам в пространстве), когда один компонент экспонирует свою непрерывную переменную, а другой – отслеживает ее изменения.

Отслеживающий компонент может использовать значение переменной для описания собственного непрерывного поведения (например, переменная будет присутствовать в правой части дифференциальных уравнений, описывающих поведение отслеживающего компонента); для принятия решения (отслеживание момента выполнения некоторого условия с участием этой переменной) или других целей.

При моделировании всей системы на одной моделирующей машине (локальное моделирование) для получения корректных результатов в первом случае уравнения, задающие поведение компонентов, должны быть объединены в единую систему уравнений и решаться совместно.

Для отслеживания с необходимой точностью условия (предиката), заданного над непрерывной переменной, требуется встраивание механизмов проверки таких предикатов в алгоритмы вычисления значений непрерывной переменной (например, интеграция проверки условий с алгоритмами численных методов решения дифференциальных уравнений). Примеры связей компонентов по непрерывным переменным для первого и второго случаев приведены на рис. 2.6 и рис. 2.7 соответственно.

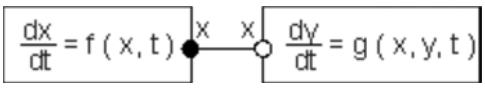


Рис. 2.6. Связь по непрерывной переменной первого типа

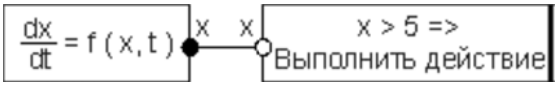


Рис. 2.7. Связь по непрерывной переменной второго типа

В процессе разбиения модели на распределенные компоненты происходит нарушение «непосредственных» связей между компонентами (рис. 2.8). Взаимодействие осуществляется посредством инфраструктуры поддержки распределенного исполнения моделей, которая может работать поверх относительно медленных каналов передачи данных.

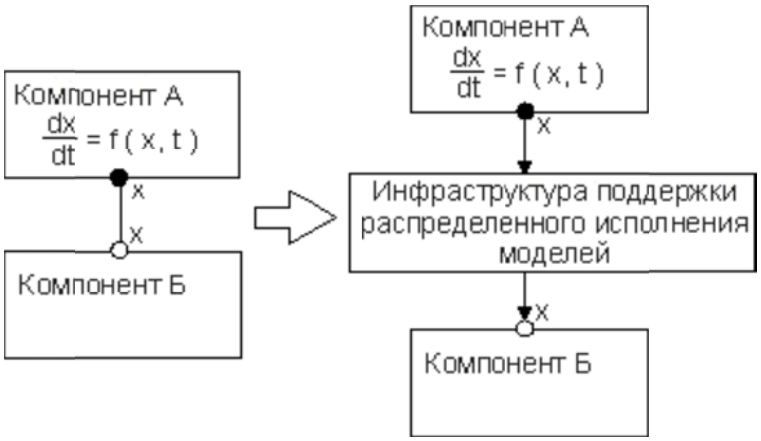


Рис. 2.8. Нарушение связи по непрерывным переменным

Связи по дискретным переменным состояния естественным образом представляются в виде сообщений, посылаемых лишь в моменты изменения значений переменных. В этом случае не происходит потери информации, так как наблюдающий компонент имеет всю информацию о дискретном состоянии наблюдаемого компонента. При разрыве связи по непрерывным переменным возникает проблема выбора метода представления непрерывно меняющегося значения переменной для удаленного компонента.

Могут быть предложены следующие решения.

Обновление значений с заданной периодичностью.

Достоинство – простота реализации.

Недостатки – существенные потери информации, точности, увеличение вероятности проявления проблемы чувствительности (поведение компонента между двумя последовательными обновлениями неизвестно для наблюдателя, который может пропустить интересующее

его событие). Достижение заданной точности может потребовать большой частоты обновлений, передаваемых по сети, что быстро сведет скорость моделирования к нулю.

Обновление значений с заданной точностью. Моделирующий компонент посылает обновление наблюдающему компоненту только в случае, если отклонение фактического значения переменной от предыдущего значения, которое было передано ранее, превысит некоторый порог.

Достоинство – это значительно более гибкий подход, позволяющий гарантировать заранее заданную точность.

Недостатки – достижение заданной точности может потребовать большой частоты обновлений.

Предсказание поведения. Компонент делает предположение о дальнейшем поведении переменной и передает наблюдающему компоненту параметры аппроксимирующей функции.

Обновление параметров происходит при достижении максимально допустимого отклонения предсказания от фактического значения переменной.

Достоинства – более точное представление поведения.

Недостатки – частота обновлений зависит от адекватности выбора аппроксимирующей функции, дополнительная вычислительная нагрузка на компоненты зависит от вида критерия допустимого отклонения и сложности вычисления аппроксимирующей функции, ошибка определения момента выполнения условия над наблюдаемой переменной может быть существенна.

Однако рассмотренные выше методы, к сожалению, в ряде случаев не могут обеспечить необходимую точность при моделировании гибридных систем при приемлемом количестве обновлений.

Метод, который полностью решает проблему для ограниченного, но важного класса распределенных гибридных моделей: удаленная обработка предикатов.

Удаленная обработка предикатов. В гибридных системах непрерывное и дискретное поведение являются взаимосвязанными и могут влиять друг на друга. При распределенном моделировании представление дискретных событий и взаимодействий не вызывает особых затруднений.

Однако, отслеживание влияния непрерывного поведения на дискретное поведение (рис. 2.7) требует достаточно точного представления непрерывного поведения для отслеживающего компонента.

В распределенном случае применение периодического обновления либо предсказания с помощью аппроксимирующей функции может привести к неприемлемому запаздыванию или пропуску события при определении момента выполнения условия, заданного над непрерывной переменной.

Как правило, форма таких условий известна на этапе построения модели (например, $h > C$, где h – непрерывная переменная, C – константа).

Идея удаленной обработки предикатов заключается в перенесении процесса вычисления условий из наблюдающего компонента в компонент, моделирующий соответствующую переменную. Наблюдающий компонент устанавливает значение параметров. Моделирующий компонент вычисляет значения переменной, проверяет выполнение условия и сообщает о факте его выполнения наблюдающему компоненту.

Определение момента времени наступления события в этом случае происходит с максимально возможной точностью, что может быть немаловажным при моделировании процессов управления и принятия решений. При этом решается задача уменьшения объема передаваемой в процессе моделирования информации между распределенными компонентами.

Рассмотренный метод удаленной обработки предикатов при распределенном моделировании гибридной системы имеет свои ограничения.

В случае необходимости проверки сложного условия вида $P(x, y)$, где x и y – непрерывные переменные, моделируемые различными распределенными компонентами, организация удаленной обработки такого условия возможна лишь в случае, когда $P(x, y)$ может быть представлена в виде $F(P(x), P(y))$, то есть когда условия на x и y являются разделяемыми.

В противном случае может быть рекомендовано, пересмотреть способ разбиения модели системы на распределенные компоненты таким образом, чтобы компоненты, жестко связанные по непрерывным переменным исполнялись в рамках одного распределенного компонента (с использованием одной моделирующей машины и ее внутренних методов реализации связи компонентов легли в основу ***оптимистических алгоритмов*** синхронизации по непрерывным переменным).

Альтернативой может служить применение методов периодического обновления значений либо предсказания поведения непрерывной переменной с обязательным контролем приемлемости результирующей потери точности.

2.4. Методы управления временем в распределенном моделировании

Управление временем в распределённом моделировании должно обеспечивать выполнение событий в правильном хронологическом порядке. Более того, на алгоритмы синхронизации возлагается обязанность корректно выполнять повторные имитационные прогоны.

При повторном моделировании пользователь должен быть уверен, что он получит те же результаты, что и в первый раз, если входные данные останутся неизменными. Как отмечает Fujimoto [10], обеспечение следования событий в правильном порядке и проблемы повторного имитационного прогона для моделирования тренажёров несущественно. Поэтому речь о дискретно-событийном моделировании.

Сделаем предположение, что имитационная модель представляет собой совокупность логических процессов (LP_i), которые взаимодействуют друг с другом, посылая друг другу сообщения с временной меткой (time stamped) или события. События логического процесса должны выполняться в хронологическом порядке. Это требование называют ограничением локальной каузальности (local causality constraint).

Можно показать, что если игнорировать события с одинаковыми временными метками и если процесс поддерживает ограничение локальной каузальности, то выполнение имитационной программы на параллельном компьютере даст такие же результаты, как и выполнение на последовательном, где все события выполняются в соответствии с их временными метками. Кроме того, это свойство позволяет убедиться в том, что выполнение имитационного прогона повторяемо (даёт те же результаты при одном и том же наборе исходных данных).

Будем рассматривать каждый логический процесс как последовательный симулятор, поддерживающий дискретное событийное моделирование. А это означает, что каждый логический процесс содержит локальную информацию о состоянии объектов моделирования и список событий, которые были запланированы для этого процесса, но ещё не были выполнены (pending event list). Этот список необработанных событий включает локальные события (т.е. запланированные самим процессом LP_i) и события, запланированные для него другими процессами.

Работа симулятора заключается в том, чтобы выбрать из списка необработанных событий событие с минимальной временной меткой и обработать его. Так выполнение процесса можно рассматривать как выполнение последовательности событий.

Выполнение событие сопровождается изменением переменных, которые определяют состояние моделируемого объекта. Кроме того, логический процесс при выполнении очередного события может запланировать выполнение нового события ej для самого себя или для другого логического процесса.

Каждый логический процесс имеет локальные часы. Локальные часы указывают на время выполнения самого последнего обработанного симулятором события. Время, на которое запланировано логическим процессом любое событие, должно быть больше, чем значение его локальных часов.

Алгоритм управления временем должен следить за тем, чтобы события выполнялись в хронологическом порядке. Эта задача не является тривиальной. Действительно, логический процесс заранее не может знать о том, на какое время будет запланировано событие, которое он получает от другого логического процесса. Fujimoto [10] приводит такой пример (пример 1): предположим, что в списке необработанных событий хранится событие с временной меткой 10. Может ли симулятор логического процесса выбрать его для обработки. Это можно было бы сделать, если бы логический процесс каким-нибудь образом знал о том, что другой

логический процесс не запланировал для него события, со временем меньшим, чем 10.

Рассмотрим пример 2 (приведён в [7]).

Пусть модель представляет собой совокупность трёх процессов. Один процесс отображает поведение покупателя, второй – магазина, в котором покупатель совершает покупки, а третий процесс – деятельность банка, со счёта которого покупатель снимает деньги.

Предположим, что покупатель приобрёл товары в магазине на определённую сумму N (в кредит) (событие e_1, произошло в момент времени t_1=9). Магазин уведомил об этом банк (e_2, t_2=10) Сумма на счёте уменьшается S=S-N. Покупатель посещает банк с целью снять деньги со счёта (e_3, t_3=11). Если денег на счёте достаточно, то банк выдаёт клиенту (которым является покупатель) запрошенную им сумму. Если счёт меньше запрошенной суммы, то покупателю будет отказано.

Рассмотрим описанную выше ситуацию: если уведомление в банк из магазина поступит позже того, как покупатель снимет сумму с вклада в банке (которой уже нет на счёте), то банк понесёт убытки. Ситуация, которая обрисована выше, возникла вследствие того, что хронология событий была нарушена.

Нарушение хронологического порядка могло произойти по той причине, что в распределённом моделировании время для разных логических процессов движется с разной скоростью.

Например, процесс, реализующий работу магазина, выполняется на загруженном процессоре. Уведомление банку поступает, следовательно, позже, поскольку процесс «банк» «убежал» вперёд, так как он выполняется на менее загруженном процессоре. Распределённый алгоритм должен уметь бороться с такими парадоксами времени.

В этом и заключается проблема синхронизации компонентов распределённого моделирования. Было предпринято множество попыток решить эту проблему [10]. В настоящее время все алгоритмы синхронизации делятся на ***консервативные*** и ***оптимистические алгоритмы***.

Если вернуться к рассмотренному примеру 1, то ***консервативный алгоритм*** не позволит логическому процессу обрабатывать событие с временной отметкой 10, пока не убедится, что другой логический процесс не запланировал для него события с временной меткой, меньшей 10.

Оптимистический алгоритм позволяет выбирать из списка необработанных событий очередное событие и обрабатывать его, исключив проверку событий, планируемых другими логическими процессами. Однако отдельный программный механизм реализует обнаружение ошибок и восстановление от ошибок выполнения событий, которые происходят не в хронологическом порядке.

Рассмотрим более подробно каждый из алгоритмов управления временем.

2.4.1. Консервативное управление временем

Первые алгоритмы синхронизации использовали консервативный подход.

Принципиальная задача консервативного протокола – определить время, когда обработка очередного события из списка необработанных событий является «безопасным». Иными словами, событие является безопасным, если можно гарантировать, что процесс в дальнейшем не получит от других процессов событие с меньшей временной меткой.

Консервативный подход не позволяет выполнять событие до тех пор, пока нет гарантии, что оно является безопасным.

Большинство консервативных алгоритмов основано на вычислении LBTS (Lower Bound on the Time Stamp – нижняя граница временных меток) будущих сообщений, которые могут быть получены логическим процессом. Этот механизм позволяет определить, является ли очередное событие из списка необработанных событий безопасным. Действительно, если консервативный алгоритм определит, что LBTS = 11, то все события из списка необработанных событий с временной отметкой, меньшей 11, являются безопасными и могут быть выполнены. Соответственно, события с временной меткой, большей 11 не являются безопасными и не могут быть выполнены. Что касается событий, которые имеют временную отметку, равную 12 (LBTS = 12), то их обработка зависит от реализации конкретного консервативного алгоритма и правил, по которому должны обрабатываться события, запланированные на один и тот же момент времени (одновременные события – simultaneous events).

Будем считать, что логические процессы не содержат событий, запланированных на одно и то же модельное время.

2.4.2. Алгоритм с нулевыми сообщениями

Первыми консервативными алгоритмами считаются алгоритмы, разработанные Bryant [7], Chandy [8], Misra [9].

Алгоритм предполагает следующее:

- топология процессов, которые посылают сообщения друг другу, известна и фиксирована;
- каждый логический процесс посылает сообщения с неубывающими временными метками;
- коммуникационная среда обеспечивает доставку сообщений в том порядке, в котором они были посланы.

Исходя из этих предположений, можно сделать следующее заключение:

- временная метка сообщения, которое было получено последним на линии связи, является нижней границей временных меток (LBTS) всех будущих сообщений, передаваемых по этой линии связи;

• нижняя временная метка (LBTS) логического процесса определяется как минимальная из всех нижних временных меток, полученных процессом по всем входным линиям связи.

Сообщения каждой линии связи находятся в очереди, которая обрабатывается по дисциплине FIFO (first-in-first-out). Очередь упорядочена также в соответствии с временными метками.

Каждая линия связи имеет своё локальное время, которое равно временной отметке первого сообщения в очереди (если таковые имеются) или временной отметке последнего принятого сообщения. Все события, которые планирует сам процесс для себя, находятся в другой очереди. Алгоритм периодически выбирает линию связи с наименьшим временем и, если в ней есть события, то он обрабатывает это событие. Если очередь пуста, то процесс блокируется. Процесс никогда не блокируется при проверке состояния очереди сообщений, которые он планирует для себя. Итак, этот алгоритм гарантирует, что каждый логический процесс будет обрабатывать события в хронологическом порядке.

Цель: необходимо, чтобы события выполнялись в хронологическом порядке: WHILE (не конец моделирования). Ждать, пока каждая из очередей содержит хотя бы одно сообщение. Удалить сообщение с наименьшей меткой, просмотрев каждую из очередей. Обработать это событие. END-LOOP.

Вернёмся к примеру. В очередях последовательно были обработаны события, запланированные на время t1=10,t2=11. Далее процесс ожидает появления сообщения от покупателя. Иначе он не может продолжить работу. Процесс блокируется.

Несмотря на то, что алгоритм обеспечивает локальную каузальность, при его выполнении могут возникнуть тупики. Действительно, если нижняя граница временных отметок на всех линиях связи будет меньше, чем необработанные события, то это приведет к ожиданию выполнения другого цикла.

Пример. Итак, у нас три процесса LP_1, LP_2, LP_3. Процесс LP_1 блокирован, т.к. ожидает прихода сообщения от LP_3 и не может отослать сообщения процессу LP_2. Процесс LP_2 тоже блокирован, он ожидает сообщений от LP_1. Процесс LP_3 в свою очередь блокирован, поскольку в его входной очереди нет сообщений от первого процесса LP_1. Теперь мы видим, что происходит циклическое ожидание процессов (тупик).

Для того, чтобы избежать возникновение тупиков, Chandy и Misra предложили использовать нулевые сообщения. Процесс LP_a посылает сообщение с временной меткой Tnull процессу LP_b. Этим процесс LP_a сообщает процессу LP_b, что он не пришлёт процессу сообщение с временной отметкой, меньшей, чем Tnull. Нулевые сообщения не соответствуют каким-либо физическим явлениям моделируемого объекта.

Это искусственный приём, используемый для того, чтобы избежать возникновения тупика.

Процессы отсылают нулевые сообщения по всем выходным дугам после обработки очередного события. Нулевое сообщение служит дополнительной информацией для того, чтобы определить очередное безопасное событие.

Нулевое сообщение обрабатывается как обычное сообщение за исключением того факта, что ни одна переменная процесса и никакое событие не будут запланированы. Локальное время логического процесса продвигается до временной отметки нулевого сообщения.

Выясним, каким образом процесс определит временную метку нулевого сообщения, которое он отсылает.

Значение часов каждой линии связи определяет нижнюю границу временной метки следующего события, которое будет извлечено из буфера этой линии связи. Эта нижняя граница в совокупности со сведениями о выполнении процесса являются исходной информацией для определения нижней границы временных меток для всех посылаемых с выходных дуг сообщений. Fujimoto приводит такой пример: если известно, что некоторое обслуживающее устройство обрабатывает заявку в течении T единиц модельного времени, то временная метка любого отсылаемого сообщения, по крайней мере, на T единиц модельного времени больше, нежели временная метка любого сообщения, которое может появиться в будущем.

Как только процесс завершит обработку нулевого или ненулевого сообщения, он вновь посылает нулевое сообщение по выходным дугам. Процесс, получивший нулевое сообщение, обязан вычислить нижнюю границу временной метки и отослать эту информацию свои соседям и т.д.

Цель алгоритма с нулевыми сообщениями (выполняется каждым логическим процессом LP): обеспечить выполнение событий в хронологическом порядке и избежать тупиков. WHILE (не конец моделирования). Ждать пока не выполнится условие: каждая очередь FIFO содержит хотя бы одно сообщение. Удалить событие с наименьшей временной отметкой из FIFO. Обработать это событие. Послать нулевое сообщение своим соседям с временной меткой, указывающей нижнюю границу будущих событий, посланных этому LP (текущее время + look ahead). END-LOOP.

Итак, приведённый выше алгоритм с использованием нулевых сообщений позволяет избавиться от тупиков.

2.4.3. Использование look ahead

Для консервативных алгоритмов синхронизации характерной особенностью является использование предварительного просмотра (look ahead).

Предположим, что некоторый логический процесс имеет локальное время с отметкой T. Процесс может гарантировать, что любое будущее сообщение, отосланное им, будет иметь временную метку, превосходящую временную метку пришедшего позднее сообщения, на величину, равную

$T+L$. В этом случае говорят, что процесс имеет предварительный просмотр величиной, равной L. Предварительный просмотр используют для того, чтобы вычислить временную метку нулевого сообщения. Величина предварительного просмотра указывает на то, что на протяжении некоторого временного отрезка L исходящих сообщений не будет.

Природа происхождения look ahead («предварительного просмотра», «забегания вперёд», «предсказания поведения») может быть различной.

Физические ограничения, ***связанные с замедленной реакцией физического процессора на внешние события***. Например, минимальное время реакции танка на команду оператора – 0.5 с, таким образом, логический процесс танка может гарантировать отсутствие событий в этот промежуток времени (не следует забывать, что алгоритмы распределённого моделирования разрабатывались под эгидой Министерства обороны США).

Физические ограничения, ***связанные с тем, как быстро один физический процесс может воздействовать на другой***. Например, пусть два танка находятся на расстоянии 5 км друг от друга, и при этом существует максимальная возможная скорость полёта снаряда. Эти ограничения устанавливают нижний предел времени, через которое первый танк может воздействовать на второй.

Допустимость небольших временных неточностей. Пусть логический процесс планирует событие (некоторые действия, назначенные на определённое время) посылки сообщения на время T. Если этому сообщению присвоить значение времени $T+1$, на точность моделирования это не повлияет. Поэтому в качестве значения «предсказания» look ahead может быть выбрана единица.

Дискретное время. При моделировании с дискретным временем любое событие может быть запланировано на время $T + \Delta T$, где ΔT – это шаг дискретизации. В таких системах у любого процесса будет естественный look ahead равный этому шагу.

Инерционное поведение. Пусть танк двигается с некоторой скоростью, и в течение следующих десяти минут ничто не сможет ему помешать двигаться дальше, тогда все его действия за эти десять минут могут быть смоделированы немедленно, невзирая на возможность прихода отстающих сообщений, т.к. они всё равно не повлияют на эти события.

Обсуждаемый алгоритм допускает множество модификаций, таких как, например, подсчёт look ahead по отдельности для различных исходящих дуг, что приводит к значительному уменьшению простоя логических процессов.

К отрицательным характеристикам этого алгоритма можно отнести тот факт, что если процессы сильно связаны, то все логические процессы будут продвигаться лишь на очень малые промежутки времени. Таким образом, моделирование по своей сути будет последовательным. Недостатком алгоритма является и то, что при небольших значениях look ahead возможна ситуация циклического ожидания нескольких LP с

небольшими продвижениями локальных часов. В некоторых системах возможно, а зачастую и необходимо, использование нулевого предварительного просмотра (zero look ahead), однако обсуждаемый выше алгоритм не допускает нулевых предварительных просмотров.

2.4.4. Использование дополнительной информации о временной метке следующего события

Рассмотрим подробнее недостатки алгоритма с нулевыми сообщениями. Итак, одним из недостатков алгоритма является тот факт, что он может сгенерировать слишком большое количество нулевых сообщений.

Допустим, что мы имеем два логических процесса LP. Предположим, что оба они заблокированы. Каждый из них достиг временной отметки, равной 100 ($TlocLP_1$ =100) и значение look ahead =1. Пусть следующее запланированное событие имеет временную метку, равную 200. Алгоритм с нулевыми сообщениями будет действовать следующим образом: сообщения будут посланы процессами в моменты времени, равные 101, 102,103 и т.д. Это будет продолжаться до тех пор, пока процессы не достигнут временной отметки, равной 200. После этого событие с временной меткой 200 (e_i, 200), будет выполнено. Таким образом мы видим, что было послано и обработано 100 нулевых сообщений до того момента, когда пришёл черёд ненулевого необработанного события. Из примера [1] видно, что алгоритм с нулевыми сообщениями в этом случае неэффективен. Ситуация ещё более изменится к худшему, если мы будем работать с большим количеством процессов.

Рассмотрим похожий пример, который рассматривает взаимодействие трех логических процессов. Нулевые сообщения запланированы на 5.0, 5.5, 6.0 и т.д., они отсылаются через каждые 0.5 минут единиц модельного времени. Нулевых сообщений гораздо больше, нежели в предыдущем примере. Итак, если look ahead мало, то это приводит с к большому количеству нулевых сообщений.

Проблема заключается в том, что мы используем только текущее модельное время и значение предварительного просмотра (look ahead) для вычисления минимальной временной отметки будущих событий процесса. Ключом к усовершенствованию алгоритма является информация о значении временной метки следующего необработанного события. Если бы все логические процессы, участвующие в моделировании, владели этой информацией, они могли бы сразу перевести локальные часы на отметку 200 ($TlocLP_1$ =200). Таким образом, мы можем избавиться от описанного выше недостатка алгоритма с нулевыми сообщениями.

Многие усовершенствованные алгоритмы используют изложенную только что идею.

Другая проблема заключается в следующем: пусть каждый процесс представляет собой вершину полного графа (каждый процесс может

послать сообщение каждому из остальных процессов). При обработке каждого нулевого события каждый процесс выполняет широковещательную рассылку сообщений всем другим процессам. Одним словом, количество генерируемых процессами нулевых событий будет чрезвычайно большим.

Chandy и Misra решили эту проблему следующим образом: вычисления продолжаются до возникновения тупика. Далее тупик обнаруживают и устраняют. Тупик может быть устранён по той причине, что сообщение с минимальным временем безопасно и, следовательно, может быть обработано.

Альтернативой этому решению является вычисление нижней границы с целью увеличить множество безопасных сообщений.

Существуют и другие консервативные протоколы. Некоторые протоколы используют синхронное выполнение цикла, включающего:

- определение безопасного для обработки события;
- обработку этого события.

Каждый процесс вычисляет нижнюю временную отметку (LBTS) сообщений, которые он может получить от других процессов.

Следующие данные можно извлечь их состояния объекта в заданный момент времени:

- модельное время следующего события логического процесса, если этот процесс заблокирован;
- текущее модельное время логического процесса, если процесс не заблокирован;
- значение «предварительного просмотра» (look ahead);
- временную метку каждого сообщения, которое было отослано, но ещё не дошло до адресата.

Для получения этой информации можно воспользоваться синхронизацией с «барьером». Алгоритм синхронизации с барьером приведён ниже:

Синхронизирующий алгоритм с барьерами. DO WHILE (остались необработанные события). Синхронизация с барьером; удаление всех сообщений из сети. LBTS = min (N_i + LA_i); N_i = время следующего события из LP_i; LA_i = look ahead LP_i. S = множество событий с временной отметкой ≤ LBTS. Обработать события из S. END DO.

С помощью синхронизации с барьером можно получить снимок состояния (snapshot) модели. Снимок состояния – это состояния всех логических процессов без сообщений, которые находятся в пути, т.е. были посланы, но ещё не дошли до адресата.

Для того, чтобы определить это состояние, каждый процесс подсчитывает количество отосланных сообщений (M out) и количество полученных сообщений (M in). Если суммы всех отосланных и всех принятых сообщений совпали, и все логические процессы достигли

барьера, то можно говорить о том, что в системе нет сообщений, находящихся в пути.

Для определения безопасного события часто используют «расстояние» (distance).

Расстояние – это временной отрезок, который необходим для прямого или косвенного воздействия одного логического процесса на другой. Расстояние может быть использовано логическим процессом для определения временной отметки события, которое ему будет послано [10]. Этот подход возможен, если известна структура модели, т.е., если известны процессы, которые могут переслать сообщения конкретному процессу LP_i.

2.4.5. Оптимистическое управление временем

В отличие от консервативных алгоритмов, не допускающих нарушения ограничения локальной каузальности, оптимистические методы не следят за этим ограничением. Однако этот подход гарантирует выявление нарушения каузальности и его устранение.

Оптимистический метод имеет два важных преимущества по сравнению с консервативным методом.

Во-первых, ему свойственен более высокий уровень параллелизма. Действительно, если два события могут влиять друг на друга, но алгоритм таков, что на самом деле этого нет, то оптимистический программный механизм позволяет обрабатывать события параллельно в отличие от консервативного. Консервативный метод в этом случае всё равно требует последовательного выполнения этих двух событий.

Во-вторых, консервативный механизм обычно требует дополнительной информации, например, расстояние между объектами. Это необходимо для определения безопасного события процесса.

Оптимистические алгоритмы, использующие такую информацию, тоже выполняются быстрее, но влияние этой информации на корректность выполнения гораздо меньше. Оптимистический алгоритм, таким образом, более прозрачен при разработке математического программного обеспечения, чем консервативный, разработка программного обеспечения становится проще. С другой стороны, для оптимистического подхода могут потребоваться дополнительные вычисления, что снижает эффективность его применения.

Для оптимистических алгоритмов характерно:

- логические процессы обмениваются сообщениями с временными метками;
- топология процессов меняется, т.е. возможно появление новых процессов;
- сообщения, передаваемые по одной линии связи, не должны быть упорядочены по времени;

• сеть должна с достаточной надёжностью передавать сообщения, но не отвечает за порядок передачи.

Наиболее известный оптимистический алгоритм – алгоритм, разработанный Джефферсоном (Jefferson, 1985). Алгоритм известен под названием Time Warp. Когда логический процесс получает событие, имеющее временную отметку меньшую, чем уже обработанные события, он выполняет откат и обрабатывает эти события повторно в хронологическом порядке. Откатываясь назад, процесс восстанавливает состояние, которое было до обработки событий (используются контрольные точки) и отказывается от сообщений, отправленных «откаченными» событиями. Для отказа от этих сообщений разработан изящный ***механизм антисообщений***.

Антисообщение – это копия ранее отосланного сообщения. Если антисообщение и соответствующее ему сообщение (позитивное) хранятся в одной и той же очереди, то они взаимно уничтожаются. Чтобы изъять сообщение, процесс должен отправить соответствующее антисообщение. Если соответствующее позитивное сообщение уже обработано, то процесс-получатель откатывается назад. При этом могут появиться дополнительные антисообщения. При использовании этой рекурсивной процедуры все ошибочные сообщения будут уничтожены.

Для того, чтобы оптимистический алгоритм стал надёжным механизмом синхронизации, необходимо решить две проблемы:

• некоторые действия процесса, например, операции ввода-вывода, нельзя «откатить»;

• обсуждаемый алгоритм требует большого количества памяти (для восстановления состояний процессов в контрольных точках, которые создаются независимо от того, произойдёт откат или нет), требуется особый механизм, чтобы освободить эту память.

Обе эти проблемы решаются с помощью глобального виртуального времени (GVT). GVT –это нижняя граница временной отметки любого будущего отката. GVT вычисляется с учётом откатов, вызванных сообщениями, поступивших «в прошлом». Таким образом, наименьшая временная метка среди необработанных и частично обработанных сообщений и есть GVT. После того, как значение GVT вычислено, фиксируются операции ввода вывода, выполненные в модельное время, превышающее GVT, а память восстанавливается (за исключением одного состояния для каждого из логических процессов).

Вычисление GVT очень похоже на вычисление LBTS в консервативных алгоритмах.

Это происходит потому, что откаты вызваны сообщениями или антисообщениями в прошлом логического процесса. Следовательно, GVT – это нижняя граница временной метки будущих сообщений (или антисообщений), которые могут быть получены позже.

Существует несколько алгоритмов для вычисления GVT (LBTS) [10]. В асинхронных алгоритмах вычисление GVT выполняется в фоновом

режиме во время имитационного прогона. При этом возникает трудность, которая заключается в том, что о локальном минимуме процессы должны извещать в разные моменты времени. Вторая проблема связана с учётом сообщений, которые были отосланы, но ещё не получены. Некоторые авторы [10] предлагают использовать последовательные отрезки вычислений и счётчики сообщений, эффективно решающие описанные выше проблемы.

Для чистых Time Warp систем характерно чрезмерное «забегание» вперёд некоторых процессов. Это приводит к весьма серьёзной трате памяти и слишком длинным откатам.

Большинство оптимистических алгоритмов пытаются ограничить это «забегание». С этой целью вводятся «перемещаемые» в модельном времени окна. Окно определяется как [GVT, GVT+W], где W – это параметр, задаваемый пользователем. Процесс обрабатывает только те события, временные метки которых находятся в данном временном интервале.

Существуют более сложные модификации алгоритма синхронизации с перемещаемым окном: для параметра W задаётся алгоритм его пересчёта.

Другой подход заключается в том, что отправка сообщения откладывается до того момента, когда появляется гарантия, что она не состоится позже отката обратно. Другими словами, GVT продвигается до модельного времени, на которое было запланировано событие. Таким образом, исключается необходимость в антисообщениях и исключаются каскадированные откаты (откаты, вызывающие новые дополнительные откаты) [10].

Существует подход, использующий «просмотр назад» (look back). «Просмотр назад» – это нечто похожее на «предварительный просмотр» в консервативных протоколах синхронизации. Он позволяет избавиться от анти-сообщений.

Кроме того, существует техника прямой отмены, который иногда используется для срочной отмены некорректных сообщений (Fujimoto).

Другой проблемой, связанной с реализацией оптимистического алгоритма, является большие затраты памяти на хранение исторической информации.

Рассмотрим некоторые решения проблемы памяти более подробно:

- выполнить откат с той целью, чтобы освободить память;
- выполнять сохранение текущего состояния реже, чем сохранение после каждого события (периоды сохранения можно указывать в начале моделирования или пересчитывать после каждого сохранённого состояния);
- выполнять освобождение памяти, занятой векторами состояний, даже в том случае, если их временная метка больше, чем GVT.

Ранние подходы к управлению выполнением Time Warp алгоритма с целью оптимизации основывались на параметрах, задаваемых пользователем. Позже применялись адаптивные подходы, которые следили

за выполнением алгоритма и выполняли регулировку этого алгоритма с целью повышения его скорости выполнения.

2.4.6. Выбор алгоритма управления временем для реализации системы моделирования

Алгоритмы управления временем (синхронизации) – это достаточно хорошо изученная область дискретного распределённого моделирования. Тем не менее, не существует общего мнения на тот счёт, какой из алгоритмов синхронизации лучше: консервативный или оптимистический.

Действительно, выбор алгоритма зависит от конкретного приложения:

- если приложение имеет хорошие характеристики для использования look ahead (заглядывания вперёд, предварительного просмотра) и вычисление этого look ahead, в конечном счете, незначительно увеличивает накладные расходы на разработку приложения, то рекомендуется использовать ***консервативный алгоритм***;
- с другой стороны, в некоторых приложениях ***оптимистический алгоритм*** будет выполняться скорее, но к его недостаткам можно отнести тот факт, что реализация оптимистического алгоритма сложна и требует большого количества памяти.

2.5. Организация взаимодействия распределенных компонентов на основе агентной технологии

Для реализации методов взаимодействия распределенных компонентов построена специальная ***агентная система***, ядро которой тесно взаимодействует с ядром компонента распределенной модели. Эта агентная система и была включена в систему моделирования AnyLogic, которая подробно рассматривается в п. 3.5.3.

Суть предложенного подхода [7] в следующем. К компоненту распределенной модели подключается специальный программный модуль – ядро агентной системы. Данный модуль осуществляет взаимодействие с ядром модели и имеет доступ ко всем переменным компонента с высокой точностью (определяемой используемым численным методом). К этому модулю по сети через специально разработанный интерфейс подключаются программы-агенты, которые засылаются другими компонентами распределенной модели, если они заинтересованы в мониторинге какой-либо переменной (переменных) данного компонента.

Программа-агент может представлять собой: программу, контролирующую частоту обновлений (дискретизация по значению); аппроксимирующую программу; программу-предикат и др. Будучи засланными, исполняющиеся программы-агенты осуществляют взаимодействие с «родительским» компонентом по сети. Так, например,

агент-предикат в момент срабатывания предиката, высылает уведомление о произошедшем событии. Агент-аппроксиматор периодически высылает вектор параметров аппроксимирующей функции и так далее.

Так как агенты фактически подключаются к ядру системы моделирования, это позволяет им с большой точностью отслеживать изменения переменных, производить аппроксимацию и вычисление моментов срабатываний предикатов.

2.5.1. Предопределенные типы агентов

В общем случае пользователь системы может написать своего агента фактически «с нуля» (единственное условие: агент обязательно должен удовлетворять требованиям интерфейса системы). Однако, в ходе работы были созданы несколько предопределенных типов агентов, которые могут быть использованы в большинстве случаев.

Агент-линейный аппроксиматор осуществляет мониторинг заданной переменной и осуществляет аппроксимацию поведения этой переменной с помощью полиномов первой степени.

Агент-квадратичный аппроксиматор осуществляет мониторинг заданной переменной и осуществляет аппроксимацию поведения этой переменной с помощью полиномов второй степени.

Агент-вычислитель предиката предусматривает вычисление заданного предиката на переменных удаленного компонента. Агент информирует компонент, инициировавший его загрузку, о моменте наступления данного события.

2.5.2. Преимущества агентной технологии

Решение, основанное на применении ***агентной технологии***, позволяет значительно ускорить распределенное моделирование гибридной системы, увеличить точность моделирования, уменьшить сетевой трафик.

Рассмотрим подробнее ***свойства*** этого решения.

Универсальность. Разработанная агентная система поддерживает четыре метода организации взаимодействия распределенных компонентов, которые могут быть использованы в зависимости от требований конкретной модели.

Динамическая загрузка. Код агента может быть подгружен по сети в любой момент, когда в нем возникнет необходимость. Агент так же может отключиться от мониторинга переменных в любой момент.

Поддержка обратной связи. В интерфейс агентов включена возможность обратной связи с «родительским» компонентом. Агент может общаться с компонентом, инициировавшим его загрузку посредством специальных сообщений. Таким образом, возможно постоянное согласование параметров мониторинга и их модификация.

Гибкость. Данное решение позволяет использовать в качестве агента практически любой код, т.е. возможно использование не только предикатов, аппроксимаций, но и других, более сложных конструкций. Например, внутри агента возможно использование некоторой предыстории изменения переменных, возможен анализ сложных переменных-объектов (например, текстовых строк) и так далее.

2.5.3. Пример работы агента-квадратичного аппроксиматора

На компонент, осуществляющий моделирование поведения, маятника был послан аппроксимирующий агент, целью которого был мониторинг ординаты маятника с абсолютной точностью = 0,01.

На рис. 2.9 представлен график координаты аппроксимируемой функции (абсцисса).

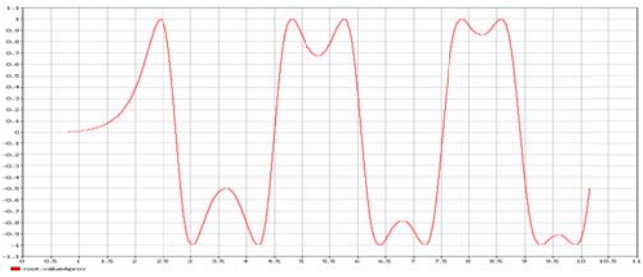
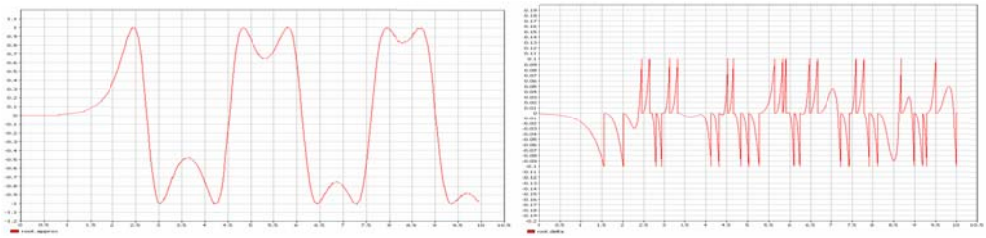


Рис. 2.9. Реальное поведение маятника

На рис. 2.10 представлены: график функции, построенной по результатам работы агента, осуществляющего аппроксимацию данной переменной полиномами второй степени; и график погрешности относительно функции-оригинала.



a) b)

Рис. 2.10. Аппроксимированное поведение (a) и график погрешности аппроксимации (b)

Точность аппроксимации – 0,01. При этом число сообщений (коэффициенты аппроксимирующего полинома), переданных по сети, составило 81.

Для сравнения: достижение той же точности при использовании дискретизации по значению потребовало 1315 обновлений значения переменной, а при использовании дискретизации по времени – более 3000.

2.6. Резюме

В настоящее время наметилась тенденция перехода от последовательного имитационного моделирования к распределенному имитационному моделированию.

Выполнение вычислительного эксперимента с имитационной моделью на нескольких компьютерах, объединённых в сеть (локальную или глобальную), можно рассматривать как ***распределённое имитационное моделирование***.

Причины перехода к распределённому имитационному моделированию обусловлены следующими факторами.

1. Исследователь, который использует ресурсы N процессоров многопроцессорной ЭВМ (или N компьютеров, объединённых в сеть), в конечном счёте, пытается добиться того, чтобы время выполнения имитационного эксперимента сократилось в N раз.

2. Исследователь, использующий ресурсы нескольких процессоров или компьютеров предполагает, что локальная память этих процессоров или компьютеров также будет использована для решения его задачи.

3. Необходимость объединения нескольких имитационных систем в одну распределённую среду имитационного моделирования.

4. Возможность работы нескольких исследователей над одним и тем же имитационным проектом через Интернет.

Распределённое моделирование развивалось достаточно независимо в трёх различных областях знаний:

- в высокопроизводительных вычислениях;
- в военных приложениях;
- в игровых приложениях через Internet.

Развитие распределенного моделирования идет по двум основным направлениям.

Первое широко используемое ***параллельное дискретное событийно-ориентированное моделирование PDES*** (Parallel Discrete Event Simulation).

Вторая парадигма, которая появилась в распределенном моделировании – это ***объединение разнородных систем моделирования***. В этом случае компонентами имитационного моделирования являются не объекты (как это делается в последовательном моделировании), не логические процессы (PDES), а сами имитационные модели.

В настоящее время широко используется ***технология HLA***. Компонентами имитационного моделирования в этом случае являются федераты, а объединение федератов называют ***федерацией***. Федераты одной федерации могут быть ***разнородными*** (это могут быть имитационные модели, тренажёры, программа для сбора данных и т.д.).

Понятия ***комплекса*** и ***компонента*** – это связанные между собой понятия, способные переходить друг в друга. Так, ***комплекс***, внутренняя структура которого сложна и образована множеством компонентов, извне

может восприниматься как единый ***компонент***. В то же время, любой из компонентов может обладать внутренней структурой, образующей комплекс.

Комплекс моделей распределенного моделирования (распределенный модельный комплекс) позволяет объединять имитационные модели, находящиеся на разных компьютерах.

Научные изыскания сосредоточились вокруг разработки ***алгоритмов синхронизации*** или ***алгоритмов управления временем***.

В настоящее время все алгоритмы синхронизации делятся на ***консервативные*** и ***оптимистические алгоритмы***.

Выбор алгоритма управления временем зависит от конкретного приложения:

- если приложение имеет хорошие характеристики для использования look ahead и вычисление этого look ahead, в конечном счете, незначительно увеличивает накладные расходы на разработку приложения, то рекомендуется использовать ***консервативный алгоритм***;
- с другой стороны, в некоторых приложениях ***оптимистический алгоритм*** будет выполняться скорее, но к его недостаткам можно отнести тот факт, что реализация оптимистического алгоритма сложна и требует большого количества памяти.

Для консервативных алгоритмов синхронизации характерной особенностью является использование предварительного просмотра (look ahead).

Для оптимистических алгоритмов характерно:

- логические процессы обмениваются сообщениями с временными метками;
- топология процессов меняется, т.е. возможно появление новых процессов;
- сообщения, передаваемые по одной линии связи, не должны быть упорядочены по времени;
- сеть должна с достаточной надёжностью передавать сообщения, но не отвечает за порядок передачи.

Для реализации методов взаимодействия распределенных компонентов используется специальная ***агентная система***, ядро которой тесно взаимодействует с ядром компонента распределенной модели.

Решение, основанное на применении ***агентной технологии***, позволяет значительно ускорить распределенное моделирование гибридной системы, увеличить точность моделирования, уменьшить сетевой трафик.

Раздел 3. Системы распределенного моделирования

3.1. Стандарты распределенного моделирования

Исследования в области военных приложений привели к разработке стандартов, на основании которых осуществлялось объединение различных тренажёров. К этим стандартам относится DIS (Discrete Interactive Simulation), стандарт (IEEE Std 1278.1-1995 1995).

В 90-ых годах 20-го века появился стандарт Aggregative Level Simulation Protocol (ALSP). В этом стандарте использовались концепции интероперабельности и переиспользования кода из проекта SIMNET.

Оба стандарта (DIS и ALSP) были заменены технологией HLA (High Level Architecture). Первоначально предполагалось применять HLA для реализации систем моделирования в военных приложениях.

Рассмотрим подробно технологию HLA (High Level Architecture, в переводе – архитектура высокого уровня), которая используется для проведения распределённого моделирования и объявлена в США стандартом распределённого моделирования.

3.1.1. История создания стандарта HLA

В середине 90-х годов 20-го столетия при министерстве обороны США было создано специальное подразделение DMSO (Defence Modeling&Simulation Office), которое в 1996 году начало координировать исследования по созданию специальной технологии HLA, определяющей общую архитектуру всех разрабатываемых в США систем моделирования.

С этого момента всем разработчикам средств и систем моделирования предписывалось следовать стандартам HLA. И до настоящего времени DMSO отвечает за распространение и поддержку всех стандартов HLA. В 1998 году HLA была номинирована для стандартизации в NATO. Организация SISO (Simulation Interoperability Standarts Organization) в настоящее время координирует с IEEE и OMG (Object Management Group) завершение работ по стандартам HLA.

3.1.2. Цели и задачи стандарта HLA

Разработчики HLA старались добиться интероперабельности и переиспользования кода в моделировании. Разработчики отталкивались от той идеи, что ни одна система имитации не может полностью удовлетворить всех пользователей Министерства обороны США.

Технология HLA предполагает, что отдельные системы имитации (или несколько систем имитаций), предназначенных для использования в одном приложении, могут быть легко использованы в другом приложении, если их разработчики придерживаются концепции федератов. Федерат – это одна или набор взаимодействующих систем имитации. Природа

федератов может быть весьма разнообразной (программные системы, тренажёры). Федераты объединены в федерации.

Итак, стандарт (технология) HLA – это совокупность методик, соглашений, алгоритмов, которые позволяют использовать уже существующие и разные по своей сути имитационные модели и системы моделирования, тем самым, сокращая время на разработку новой системы моделирования.

Кроме того, технология HLA позволяет создавать распределённые и интерактивные системы имитации.

Под термином «распределенная система» подразумевают:

- территориальную удаленность участников друг от друга;
- логическую обособленность различных участников моделирования.

Под интерактивностью системы понимают способность ее подсистем, модулей и элементов выдавать и принимать сообщения от других подсистем, модулей и элементов.

Обмен сообщениями позволяет организовать совместное функционирование различных участников моделирования.

К дополнительным преимуществам использования технологии HLA относятся:

- ***безопасность***, которая обеспечивается наличием в системе виртуальных тренажёров (поскольку участником моделирования может быть виртуальный тренажёр, а не реальный образец техники, то проблема безопасности значительно снижается);
- ***экологическая безопасность*** обеспечивается опять-таки тем, что используются имитационные модели, а не реальные образцы техники, а потому не идёт речи о шуме и физическом воздействии на окружающую среду движущейся техникой или же эти отрицательно воздействующие на окружающую среду факторы играют незначительную роль, если участниками моделирования являются «живые» объекты управления;
- затраты: ***низкий уровень затрат*** вытекает из того, что обучаемые либо совсем не перевозятся на учебные объекты (например, аэродромы, танкодромы и т.п.), либо перемещается незначительное их число, если это необходимо, и, следовательно, избегаются обычные транспортные и организационные затраты (не будем забывать, что технологию HLA разрабатывали в Департаменте обороны).

К другим преимуществам использования HLA можно отнести также:

- поддержку широкого спектра программно-аппаратных платформ;
- объектно-ориентированную структуру;
- гибкость архитектуры;
- широту функциональных возможностей;
- масштабируемость.

Всё это определяет целесообразность изучения, освоения и применение архитектурных принципов стандарта HLA.

Первоначально стандарт HLA был разработан для Министерства обороны США. Однако, это программное обеспечение имеет более широкое применение и используется во всех областях, где обычно используется имитационное моделирование: образование, тренажёры, инженерная деятельность. Это разнообразное применение учитывалось при выработке требований к HLA.

Итак, HLA можно определить следующим образом: это основные функциональные элементы, интерфейсы, правила проектирования, допустимые для использования во всех приложениях в области имитационного моделирования, и позволяющие создать основу для разработки новых систем имитации с конкретной архитектурой.

3.1.3. Основные функциональные компоненты стандарта HLA

Ключевыми понятиями HLA являются федерация (Federation) и RTI (Run Time Infrastructure).

Федерация – это объединение компонентов имитационного моделирования, называемых ***федератами*** (federates).

В качестве федератов могут рассматриваться:

- имитационные модели и системы имитации (по терминологии стандарта HLA – «созданные» (constructive) участники);
- тренажёры и модели, интерактивно управляемые людьми – по терминологии стандарта HLA – «виртуальные» (virtual) участники;
- реальные образцы техники или системы связи и управления (по терминологии стандарта HLA – «живые» (live) участники);
- специализированное программное обеспечение, предназначенное, например, для визуализации или для сбора и обработки информации.

Стандарт (технология) HLA не накладывает ограничений на внутреннее представление федератов. Федераты могут быть написаны на любом языке, но должны строго соответствовать интерфейсу. Стандарт HLA требует, чтобы федераты обеспечивали обмен информацией между объектами различных имитационных моделей, используя структуры данных, поддерживаемые сервисами RTI (Run Time Infrastructure).

RTI (Run Time Infrastructure) – это второй функциональный компонент технологии HLA. Этот компонент представляет по сути дела операционную систему, обеспечивающую взаимодействие федераций и федератов. Более формально изложим архитектурные особенности HLA.

Итак, стандарт (технология) HLA включает компоненты (рис. 3.1), которые перечислены ниже:

- ***спецификация интерфейса*** (Interface Specification);
- ***шаблон объектных моделей OMT*** (Object Model Template), задающим формат информации, представляющей общий интерес для нескольких участников процесса моделирования;

• ***правила стандарта HLA***, к которым относятся десять базовых правил, определяющих основные принципы разработки программного обеспечения имитационного моделирования в среде HLA;

• специально разработанная для поддержки HLA ***операционная система Run Time Interface***, которая включает шесть базовых групп по управлению интерфейсом.

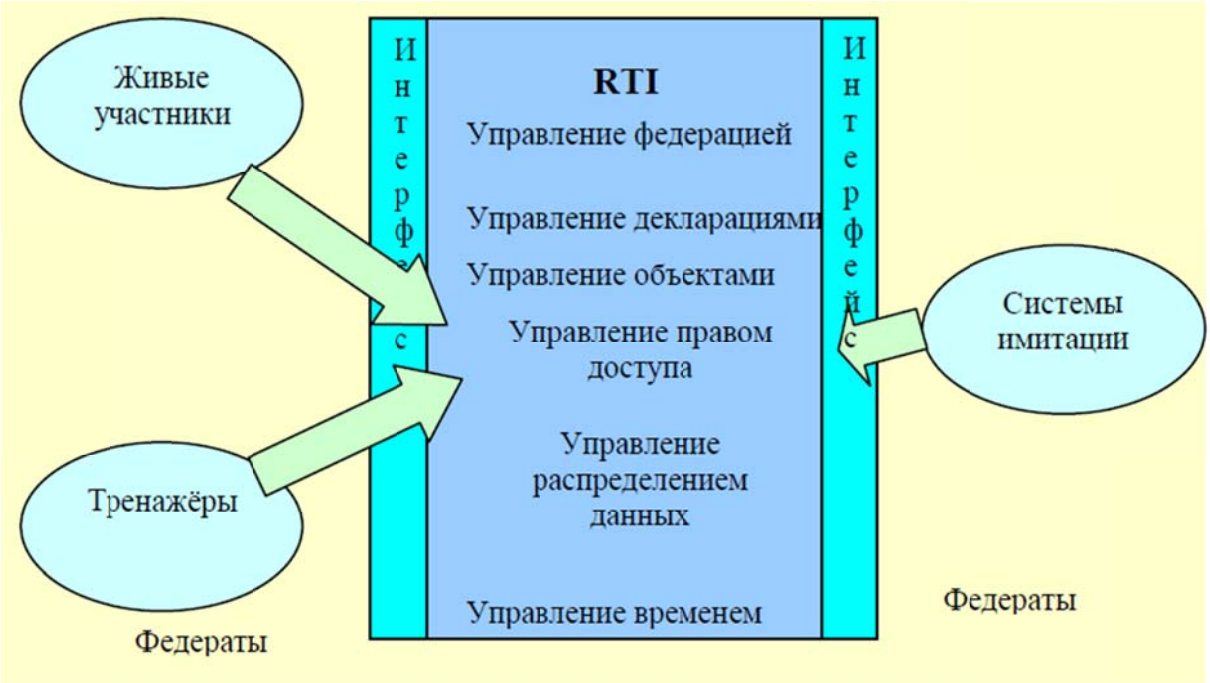


Рис. 3.1. Компоненты стандарта (технологии) HLA

Управление федерацией. Сервисы управления федерацией используют для создания и функционирования федерации в целом.

Управление декларациями. Сервисы управления декларациями используются федератами для объявления экспортируемых и импортируемых данных.

Управление объектами. Сервисы управления объектами используют для работы с объектами и их атрибутами.

Управление правом доступа. Сервисы управления правом доступа используют для передачи прав одного федерата другому. Это право даёт возможность изменить значения атрибутов объектов.

Управление распределением данных. Сервисы распределения данных позволяют уменьшить объем данных, пересылаемых между федератами, за счет более эффективного их распределения.

Управление временем. Эти сервисы синхронизируют продвижение локального модельного времени федератов.

Операционная система RTI выполняет функции симулятора. На неё возлагается обязанность обеспечить взаимодействие федератов (federate-to-federate) и управлять их работой. HLA-симулятор отличается от последовательного симулятора большим набором функциональных

модулей. Это объясняется тем, что объекты, размещённые в разных федератах, зачастую оказываются размещёнными на различных компьютерах в сети, что усложняет специальную программу (симулятор), управляющую их работой.

Теперь рассмотрим более подробно каждый из компонентов архитектуры HLA.

3.1.4. Правила стандарта HLA

В стандарте (технологии) HLA определены 10 правил. Эти правила разделены на две группы:

- правила для HLA федераций;
- правила для HLA федератов.

В начале рассмотрим **правила для федераций**.

Правило 1. Федерации должны документировать FOM (Federation Object Model) в соответствии с OMT: FOM должен документировать соглашение среди федератов о данных, которыми будут обмениваться федераты во время выполнения федерации, и об условиях обмена данными. HLA не предписывает, какие данные должны быть включены в FOM (это ответственность пользователя федерации и разработчика). HLA требует, чтобы FOM был зарегистрирован в формате IEEE P1516.2.

Правило 2. В федерации все представления объектов должны быть в федератах, а не в инфраструктуре RTI во время выполнения. Одна из основных идей HLA состоит в том, чтобы отделить моделирование (определенные функциональные возможности) от универсальной поддержки инфраструктуры RTI. В HLA RTI обеспечивает функциональные возможности, подобные распределенной операционной системе. При моделировании в HLA все атрибуты объектов должны принадлежать федератам, а не RTI. Первичная цель федератов – представление объектов, что позволяет в полной мере удовлетворить потребности пользователя или прикладной области с затратой меньшего объема ресурсов и времени.

Правило 3. Во время выполнения федерации весь обмен FOM данными среди федератов должен происходить через инфраструктуру RTI. Инфраструктура RTI должна обеспечить координацию, синхронизацию, и обмен данными среди федератов для корректного выполнения федерации. Ответственностью федератов является обеспечение того, чтобы верные данные обеспечились в верное время и что данные используются по существу, правильным способом. RTI должен гарантировать, что данные поставлены федератам в соответствии с их объявленными требованиями.

Правило 4. Во время выполнения федерации федераты должны взаимодействовать с RTI в соответствии с техническими требованиями интерфейса HLA. Технические требования интерфейса определяют, как федераты взаимодействуют с инфраструктурой RTI. Требования обмена данными между федератами должны быть определены в FOM. Наряду с

обычными приложениями требуется стандартизированный, общий интерфейс между федератами и RTI, но HLA учитывает независимое развитие и выполнение федератов и RTI.

Правило 5. Во время выполнения федерации только один федерат может быть владельцем атрибута объекта. Стандарт (технология) HLA позволяет в любой момент времени только одному федерату изменять значение атрибута объектного образца. Стандарт (технология) HLA обеспечивает также механизм динамической передачи права монопольного использования атрибута объектного образца от одного федерата к другому. Стандарт (технология) HLA располагает гибким комплектом инструментальных средств, предназначенных для использования различных комбинаций моделирований, чтобы удовлетворить потребности пользователя.

Следующие пять правил – это **правила для федератов**.

Правило 1. Федераты документируют SOM (Simulation Object Model) в соответствии с OMT (Object Model Template):HLA требует, чтобы каждый федерат имел HLA модель объекта (SOM). HLA SOM включает те объектные классы, атрибуты класса, и классы взаимодействия федератов, которые будут обнародованы в федерации. HLA не предписывает, какие данные должны быть включены в SOM; это ответственность разработчика. HLA требует, чтобы SOM был зарегистрирован в предписанном формате. Главная цель HLA состоит в том, чтобы поддержать способность к взаимодействию и многократное использование моделирований.

Правило 2. Федераты способны модифицировать и/или отражать любые атрибуты объектов в их SOM, и посылать и/или принимать взаимодействия, как определено в их SOM:HLA позволяет федератам делать объектные представления и взаимодействия, разработанные для внутреннего доступного использования как часть выполнения федерации для внешнего использования с объектами, представленными в других федератах. Также HLA включает обязательство экспортировать модифицированные значения атрибутов образца и обязательство осуществлять взаимодействия, представленные внешне (то есть, другим федератам в федерации).

Правило 3. Федераты способны передавать и/или принимать монопольное использование атрибутов динамически в течение выполнения федерации, как определено в их SOM: HLA предоставляет возможность передавать и принимать монопольное использование атрибутов образца объекта.

Правило 4. Федераты способны изменить условия (состояния), которые определяют варианты модификации атрибутов объектов, как определено в их SOM. Различные федерации могут определить различные условия, при которых атрибуты образца будут модифицированы (пример условия – некоторая указанная норма, когда значение превышает некоторый указанный порог, то данный атрибут модифицируется).

Правило 5. Федераты способны управлять локальным временем, что позволяет им координировать обмен данными с другими членами федерации. Структура управления временем в HLA предназначена для того, чтобы поддержать способность к взаимодействию среди федератов. В стандарте (технологии) HLA используются различные внутренние механизмы управления временем.

3.1.5. Шаблон объектных моделей (Object Model Template)

Стандартизированная структура (шаблон) моделей объектов HLA ***Object Model Template*** (OMT) необходима по следующим причинам:

- OMT обеспечивает доступный для понимания механизм обмена данными и общей координации среди членов федерации;
- OMT обеспечивает общий стандартизированный механизм для описания возможностей каждого члена федерации;
- OMT способствует взаимодействию и многократному использованию моделирований и их компонентов;
- OMT упрощает приложение и проект, предоставляя общую структуру и общий набор инструментальных средств для развития моделей объектов HLA.

В HLA объектные модели составлены из взаимодействующих компонентов, которые определяют информацию относительно классов объектов, их атрибутов и их взаимодействий. HLA требует, чтобы эти компоненты были представлены в виде таблиц.

Шаблон ядра модели объекта HLA должен иметь табличную форму и состоять из следующих компонентов:

- таблица идентификации объектной модели связывает необходимую важную информацию идентификации с объектной моделью HLA;
- объектная таблица структуры класса: необходима для описания пространства имен всех объектных классов моделирования и отношений типа класс-подкласс;
- объектная таблица взаимодействия необходима для описания пространства имен всех классов взаимодействия моделирования и отношений типа класс-подкласс;
- таблица атрибутов: необходима для определения особенностей атрибутов объектов в федерации;
- таблица параметров: необходима для определения особенностей параметров взаимодействия в федерации;
- таблица пространственной маршрутизации необходима для определения маршрутизации пространства для объектных атрибутов и взаимодействий в федерации;
- FOM/SOM словарь определяет все термины, используемые в таблицах.

Все федерации и федераты, должны использовать все семь компонентов из ядра OMT для представления модели объекта HLA, хотя, в некоторых случаях, некоторые таблицы могут быть пусты. Однако все объектные модели будут содержать, по крайней мере, один объектный класс или один класс взаимодействия.

Если взаимодействия в данной объектной модели отсутствуют, таблица параметров будет пуста. Заключительный OMT компонент FOM/SOM словарь необходим для того, чтобы гарантировать, что семантика терминов, используемых в модели объекта HLA, понятна и зарегистрирована. Поскольку всегда есть, по крайней мере, один термин в модели объекта HLA, то будет всегда, по крайней мере, один термин, определенный в словаре. Обычно терминов в словаре намного больше.

3.1.6. Проблема взаимодействия федератов

Концептуально инфраструктура выполнения обеспечивает основу, к которой могут быть присоединены независимо разрабатываемые федераты для формирования больших распределенных систем моделирования. Основной целью структуры высокоуровневой архитектуры управления временем (High Level Architecture Time Management, HLA-TM) является поддержка взаимодействия имитационных систем с использованием различных внутренних механизмов управления временем.

Выполнение отдельных федераций может включать в себя:

- ***системы моделирования с различными требованиями к упорядочиванию сообщений***, например, в системах распределенного интерактивного моделирования не требуется никакое упорядочивание (DIS), а в ALSP-системах требуется упорядочивание сообщений по временным меткам;
- системы моделирования с использованием различных внутренних механизмов продвижения времени, например, моделирование с фиксированным шагом и моделирование, управляемое событиями;
- системы моделирования в реальном времени (в масштабируемом реальном времени) и «as-fast-as-possible» моделирование;
- ***системы параллельного моделирования на мультипроцессорных платформах*** с использованием консервативных (без откатов) и оптимистических (с откатами) протоколов синхронизации;
- ***системы моделирования с одновременным использованием сервисов упорядочивания и передачи сообщений***, например, тренажеры с упорядочиванием сообщений и надежной передачей для определенных типов событий (например, взрывы орудий) одновременно используемые с неупорядоченной самой надежной передачей сообщений другим системам.

Поддержка этих сервисов в архитектуре HLA дает возможность федератам придерживаться определенных требований, необходимых для реализации каждого сервиса, например, забегание вперед (см. дальше) требуется для упорядочения сообщений по временным меткам. Более того,

при выполнении отдельных федератов совместно с RTI необходимо освободится от выполнения федераций, управляемых часами реального времени.

Для поддержки взаимодействия основной задачей при проектировании является ***прозрачность управления временем***. Это означает, что механизм управления локальным временем, используемый каждым федератом, не должен быть виден другим федератам.

Для этого был разработан унифицированный подход к управлению временем, обеспечивающий ряд сервисов, необходимых для различных систем моделирования.

Обычно в различных типах моделирования используется только часть всех возможностей RTI. Системы моделирования не требуют явного указания в RTI используемого механизма управления локальным временем. Для определения соотношения времени модели и реального времени моделирования при исполнении каждой федерации определяется ***глобальный масштабный коэффициент реального времени***. В «as-fast-aspossible» федерациях масштабный коэффициент берется равным «бесконечности».

В системах управления временем для федераций используются следующие допущения.

1. Не принимаются никакие общие, глобальные часы. В любой момент времени выполнения различные федераты могут достичь различного времени. В этом случае зачастую федераты должны координировать продвижение времени со временем других федератов для того, чтобы выполнять ограничения причинно-следственных связей. Даже в моделях масштабируемого реального времени отклонение от аппаратных часов в любой момент времени во время выполнения может отразиться на федератах с различными локальными часами.
2. И для RTI, и для отдельных федератов допускается использование внешнего синхронизированного с некоторой точностью реального времени моделирования.
3. Каждому событию назначается временная метка, определяемая федератом, генерирующим событие. Остальные временные метки назначаются событиям для упрощения систем моделирования реального времени. Несмотря на это, сервис упорядочивания сообщений, предоставляемый RTI, основан на временных метках, назначаемых отправителем. Вследствие этого допущения федераты могут генерировать (запланированные) события с временной меткой «в будущем», то есть с временной меткой больше текущего времени федерата.
4. Федераты могут не генерировать события с временной меткой «в прошлом», то есть с временной меткой меньше текущего времени федерата.

5. Не требуется, чтобы федераты генерировали события в хронологическом порядке (по возрастанию их временных меток). Например, федерат может сначала сгенерировать событие с меткой 10, а затем сгенерировать новое событие с меткой 8.

Классы систем моделирования, поддерживаемые в архитектуре HLA, могут быть описаны в соответствии с двумя аспектами.

Первый аспект характеризует отличия между системами моделирования с ограничениями, в которых зафиксировано соотношение продвижение времени модели и реального времени моделирования, и системами моделирования без ограничений, в которых это соотношение не зафиксировано. В данном случае системы моделирования без ограничений опираются на (масштабируемое) моделирование в реальном времени. Системы моделирования без ограничений основаны на «as-fast-as-possible» моделировании.

Второй аспект касается вопроса синхронизации продвижения времени в различных системах. Для того чтобы удовлетворить ограничение порядка следования сообщений, в системах скоординированного моделирования используется протокол (например, Chandy/Misra/Bryant). Тогда как в системах независимого моделирования продвижение локального времени не зависит от других систем и обычно задается реальным временем моделирования. В системах распределенного интерактивного моделирования используется механизм асинхронного продвижения времени. В протоколе ALSP используется скоординированное продвижение времени. Выполнение систем распределенного интерактивного моделирования обычно используют ограничения (то есть реальное время), а выполнение ALSP-систем может быть как с ограничениями, так и без ограничений.

HLA структуры управления временем различаются в двух различных представлениях времени: ***(масштабируемое) реальное время моделирования*** – это система измерения соответствующего глобального времени федерата, обычно получаемая с использованием часом процессора. Для сжатия/растяжения времени используется масштабный коэффициент. Например, масштабный коэффициент, равный 2, показывает, что федерат выполняется в 2 раза быстрее, чем реальная система. Федерат не может управлять продвижением реального времени моделирования. Логическое время ссылается на значение времени, управляемое федератом.

Логическое время – это время, которое в классическом дискретно-событийном моделировании обычно рассматривается как «время моделирования» (обычно «as-fast-as-possible» моделирование) и используется для согласования продвижения времени. Если федерат продвигает его логическое время до момента T, то он объявляет, что смоделированы все объекты до момента T. С точки зрения реализации продвижение логического времени выполняется с использованием сервисов продвижения времени, определенных в RTI. Текущее время

федерата определяется как минимум его реального времени моделирования и логического времени.

Событие в стандарте (технологии) HLA связано с обновлением атрибута, взаимодействием, обработкой или операцией удаления, выполняемой федератом. Соответствующие примитивы должны быть активизированы для того, чтобы сообщить RTI о событии. С точки зрения реализации, событие вызывает передачу сообщений федератам, которые сообщили о заинтересованности в этом событие. В данном случае термин «передача сообщения» федерату иногда используется для сокращения и означает «передача сообщения, содержащего информацию о событии». Следует заметить, что, несмотря на это, сообщения могут использоваться не только для передачи информации о событии, но и для других целей.

3.1.7. Управление временем в стандарте HLA

Управление временем в стандарте (технологии) HLA связано с ***механизмами управления продвижением времени*** во время выполнения федераций. Механизмы продвижения времени должны быть согласованы с другими механизмами, отвечающими за передачу информации (например, обновление атрибутов и взаимодействие) отдельным федератам. Упомянутой информации назначаются временные метки для того, чтобы подтвердить её достоверность. Например, федераты могут требовать, чтобы никакая информация не может быть послана «в прошлое федерата», то есть в момент времени, меньший текущего времени федерата.

Таким образом, службы управления временем, поддерживаемые в стандарте HLA, должны реализовывать два сервиса для выполнения федераций:

- ***сервисы передачи сообщений***: определены различные типы сервисов, что обеспечивает различную степень их надежности, различный порядок сообщений и затраты (латентность и пропускная способность сети);
- ***сервисы продвижения времени***: реализованы различные примитивы для выполнения запросов на продвижение в логическом времени: предусмотрен простой протокол для того, чтобы дать возможность федерату управлять потоком обновлений атрибутов и запросов на взаимодействие для этого федерата.

Сервисы передачи сообщений классифицируются по следующим признакам:

- надежность передачи сообщения;
- порядок сообщений.

Надежная передача сообщений означает, что для увеличения вероятности того, что сообщение, в конечном счете, дойдет до получателя, RTI использует определенные механизмы (например, повторная передача). При этом увеличение надежности обычно происходит за счет увеличения латентности. С другой стороны, сервисы с наиболее надежной передачи

сообщений стремятся минимизировать латентность, но ценой меньшей надежности передачи сообщений. Характеристики дисциплины очереди определяют порядок и время, в которое сообщения могут быть переданы федератам. Эти характеристики описаны ниже.

Примитивы продвижения времени при необходимости обеспечивают средства для согласования продвижения времени федератов с временными метками поступающей информации. Механизм продвижения времени в RTI должен совмещать выполнение в реальном времени, в масштабируемом реальном времени и «as-fast-as-possible» выполнение. Механизмы управления временем в стандарте HLA рассматриваются ниже.

Дисциплины очереди сообщений

Для сервисов управления временем в стандарте HLA основным механизмами являются ***механизмы упорядочивания сообщений (дисциплины очереди сообщений)***, передаваемых федератам. Для поддержки возможности взаимодействия федератов с различными требованиями предоставляется множество сервисов.

В настоящее время в стандарте HLA определено четыре **механизма упорядочивания сообщений**:

- по порядку получения;
- приоритетный;
- по временным меткам;
- каузальный.

Перечисленные механизмы увеличивают функциональность, но и затраты возрастают.

Каждый федерат для различных типов информации может использовать различные сервисы упорядочивания сообщений в одном выполнении федерации.

Упорядочивание сообщений в порядке их получения. Это наиболее простой механизм с наименьшей латентностью. Сообщения передаются федерату в порядке их получения. Логически поступающие сообщения располагаются в конце очереди с дисциплиной FIFO и передаются федерату при их удалении из начала этой очереди.

Этот способ упорядочивания сообщения должен использоваться в приложениях, где более важна минимизация латентность взаимодействия, а не следование порядку сообщений, обеспечивающему причинность. Предполагается, что этот сервис будет использован в системах моделирования с жесткими ограничениями реального времени (например, например, в федератах, связанных с реальным оборудованием). Подобным образом во многих распределенных интерактивных федерациях закономерно используется этот механизм, по крайней мере, в похожих терминах.

Приоритетный порядок. Поступающие сообщения располагаются в очереди с приоритетами. Для определения приоритета сообщения используется его временная метка. Сначала федерату передаются сообщения с наименьшими временными метками. Другими словами, RTI

пытается передать сообщения в порядке временных меток, основанном на локальной информации, доступной RTI во время передачи сообщения. Но сообщение может прийти позже, и оно будет передано федерату. При этом временная метка этого сообщения меньше метки уже переданного сообщения. Более того, этот сервис не предотвращает передачу сообщений «в прошлое» федерата (когда временная метка меньше текущего времени федерата). Несмотря на то, что этот сервис не гарантирует передачу сообщений в порядке временных меток, по сравнению с сервисами, обеспечивающими такой порядок, он более выгодный по латентности и затратам на синхронизацию. Приоритетный порядок не следует использовать, если для правильной работы федерата передача сообщений в порядке временных меток является существенной.

Приоритетное упорядочивание с наиболее надежной передачей сообщений может быть использовано в федератах, где необходимо упорядочивание последовательности сообщений, но увеличение латентности ассоциируется или с надежной передачей, или с тем, что не может быть гарантировано упорядочивание. При использовании упорядочивании сообщений по их получению или приоритетного порядка сообщения могут быть переданы федератам сразу же после их получения. RTI может буферизовать сообщения, используя каузальный порядок или порядок временных меток, до тех пор, пока не будут гарантированы желаемое упорядочивание.

Упорядочивание по временным меткам. При использовании этого сервиса сообщения будут переданы федератам в порядке их временных меток. Для этого поступающие сообщения будут храниться во внутренней очереди до тех пор, пока не будет гарантировано, что не будет получено сообщение с меньшей временной меткой. Системы моделирования не обязаны генерировать сообщения в порядке их временных меток. Для реализации этого сервиса используется консервативный протокол синхронизации для систем параллельного дискретно-событийного моделирования.

Помимо передачи сообщений в порядке их временных меток, RTI также гарантирует, что никакое событие не будет передано федерату «в его прошлое», то есть не будет передано никакое сообщение с временной меткой меньшей, чем текущее время федерата.

Это достигается путем явного запроса федерата на продвижение в логическом времени с использованием сервисов RTI продвижения времени. RTI не обеспечивает «разрешения» на продвижения времени до тех пор, пока она не может гарантировать, что никакое сообщение с временной меткой меньше, чем время разрешения, позже не будет передано.

Такой сервис упорядочивания сообщений по временным меткам необходим в классических системах дискретно-событийного моделирования (например, системы конструктивного моделирования). В этих системах обработка событий в порядке их временных меток является

нормой. Некоторые федераты не используют обработку событий в порядке их временных меток, например, системы распределенного интерактивного моделирования. В то же время для некоторых типов информации они могут быть улучшены с помощью жесткого упорядочивания событий. Такие федераты могут также использовать этот сервис, продолжая при этом использовать менее дорогие сервисы упорядочивания сообщений для других видов информации, где это менее критично.

Важной особенностью сервиса упорядочивания по временным меткам является то, что все федераты, получающие сообщения для общего набора событий, получат эти сообщения в одном и том же порядке, то есть обеспечивается общее (совокупное, полное) упорядочивание событий. Это устраняет «парадоксы времени», которые могут возникать, когда в различных моделях события упорядочены по-разному. RTI обеспечивает механизм последовательного разрыва связи, поэтому события с одинаковыми временными метками будут переданы различным системам в одном о том же порядке.

Каузальный порядок. Так же как и в сервисе упорядочивания по временным меткам, сервис каузального упорядочивания обеспечивает отправку сообщений федератам в порядке, согласованном с предшествующими и последующими событиями, представленными этими сообщениями.

Например, все сведения о совершённых клиентах сделках должны быть доступны банку ранее, чем клиент попытается получить крупную сумму денег в банке, а стрельба из оружия должна произойти до того, как разрушиться цель. В отличие от сервисов упорядочивания по приоритету и в порядке получения, где изменяющаяся латентность взаимодействия может инициировать событие уничтожения, сервисы каузального упорядочивания и упорядочивания по временным меткам гарантируют, что если одно событие «произошло раньше» второго, то все федераты получат сообщения для первого события раньше, чем для второго.

Основное различие между сервисами каузального упорядочения и упорядочивания по временным меткам связано с соответствующим определением отношения «произошло перед». В сервисе упорядочивания по временным меткам, следуя интуитивному понятию порядка в физических системах, говорят, что одно событие произошло перед вторым, если временная метка первого меньше временной метки второго. Федерация, назначая соответствующие временные метки, может точно описать (определить), какие события произойдут перед какими. Например, для описания «событие выстрела произошло до события уничтожения» событию «выстрел» может быть назначена метка 102, а событию «уничтожение» - метка 103.

В сервисе каузального упорядочивания определение отношения «произошло перед» берется по Лампорту [10]. Выполнение каждого федерата может быть рассмотрено как упорядоченная последовательность действий (например, выполнение отдельной машинной инструкции может

быть рассмотрено как действие). Особый интерес представляют два специфических действия – отправка и получение сообщений.

Лампорт определяет отношение «произошло перед» следующим образом:

- если действие A произошло перед действием B в одном и том же федерате, и действие A произошло перед действием B в упорядоченной последовательности действий этого федерата, то действие A произошло перед действием B;
- если действие A – это передача сообщения другому федерату, и B – действие получения этого сообщения во втором федерате, тогда действие A произошло перед действием B;
- если действие A произошло перед действием B, а действие B произошло перед действием C, то действие A произошло перед действием C (транзитивность).

Отношение «произошло перед» может быть распространено на сообщения:

- если сообщение X передано федератом перед тем, как этот же федерат передал сообщение Y, то сообщение X возникло перед сообщением Y;
- если федерат получил сообщение X перед тем, как этот же федерат послал другое сообщение Y, то сообщение X возникло перед сообщением Y;
- если сообщение X возникло перед сообщением Y, а сообщение Y возникло перед сообщением Z, то сообщение X возникло перед сообщением Y.

В стандарте (технологии) HLA одно событие каузально предшествует другому, если первое событие возникло перед вторым по данному выше определению.

Сервис каузального упорядочивания в стандарте (технологии) HLA гарантирует, что если событие E предшествует другому событию F (является причиной его появления), и сообщения для обоих событий переданы федерату, то сообщение для события E будет передано раньше, чем сообщение для события F.

Например, рассмотрим сценарий, изображенный на рис. 3.2. Федерат A запустил ракету, уничтожающую федерат B, а третий федерат C наблюдает эту ситуацию. Событие выстрела федерата A генерирует сообщения для федератов B и C. После получения этого события, федерат B генерирует событие обновления состояния, показывающего, что моделируемая сущность уничтожена.

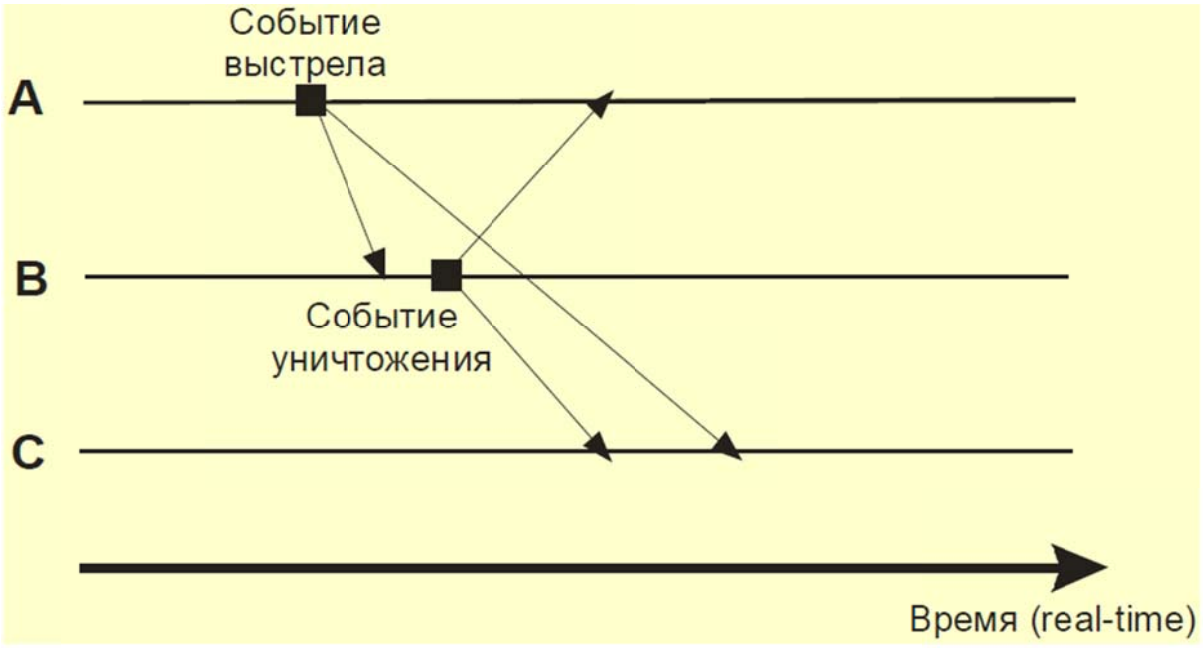


Рис. 3.2. Сценарий, демонстрирующий каузальное упорядочивание событий в стандарте (технологии) HLA

В сценарии, изображенном на этом рисунке, сообщение об уничтожении достигнет федерата С раньше, чем сообщение о выстреле. Так как событие «выстрел» каузально предшествует событию «уничтожение», сервис каузального упорядочивания гарантирует, что событие выстрела будет передано федерату С раньше, чем событие уничтожения. RTI задержит передачу сообщения для события «уничтожение» до тех пор, пока после он не получит и не передаст сообщения для события «выстрел» федерату С. Сервисы упорядочивания по получению и по приоритету не гарантируют каузального упорядочивания, поэтому в этом сценарии сообщение для события уничтожения может быть отправлено раньше, чем сообщение для события выстрела.

В основном сервисе каузального упорядочивания сообщений, соответствующие каузально не связанным событиям (далее именуемые одновременные события), могут быть посланы федератам в любом порядке. Изменение (вариации) каузального упорядочивания должно гарантировать, что все федераты получают сообщения для одновременных событий в одном и том же порядке, таким образом, определяя общее упорядочивание событий. В литературе этот сервис обычно называется CATOCS(causally and totally ordered communications support). Были разработаны и реализованы алгоритмы реализации CATOCS [10]**.**

Сравнение упорядочивания по временным меткам и каузального упорядочивания. Для того чтобы для данного типа информации определить подходящий порядок, важно понимать разницу между упорядочиванием по временным меткам и каузальным

упорядочиванием. Упорядочивание по временным меткам обеспечивает более точные сервисы упорядочивания, чем каузальное упорядочивание. И каузальное упорядочивание, и упорядочивание по временным меткам гарантируют, что сообщения для каузально связанных событий посылаются федератам в порядке, определенном каузальным отношением «произошло перед». Тем не менее, даже при использовании CATOCS упорядочивание одновременных событий в сервисе каузального упорядочивания не является детерминированным, так как оно зависит от латентности сети связи. Таким образом, каузального упорядочивания с полным упорядочиванием или без него недостаточно для получения повторяющихся результатов. Для этого необходимо использовать упорядочивание по временным меткам.

Более того, несмотря на то, что каузального упорядочивания достаточно для устранения определенных аномалий (например, получение 268 > -6.2 < 0 сообщения для снятия клиентом суммы со счёта в банке раньше, чем сообщение о событии, что клиент уже воспользовался кредитом), его недостаточно в других системах моделирования. А именно, каузального упорядочивания недостаточно в том случае, если важны отношения порядка между одновременными событиями или если между событиями есть «скрытые зависимости» (подробнее рассмотрено ниже).

Упорядочивание одновременных событий. Рассмотрим систему моделирования, состоящую из трех федератов, каждый из которых представляет танк. Предположим, что танк А собирается поджечь первую цель, чтобы оказаться в области ее обстрела. Может получиться так, что танк В окажется в этой области раньше, чем танк С. Несмотря на это, так как обновления состояний танков В и С – это одновременные события, то каузальный порядок не дает гарантию того, что танк А получит сообщение обновления состояния танка В раньше, чем сообщение обновления от танка С. Это может вызвать неправильный поджог танка С. Если такое поведение важно для достижения цели федерации, то лучше использовать сервис упорядочивания по временным меткам, а не каузального упорядочивания.

Упорядочивание по временным меткам даст правильный результат, так как обновление состояния танка В в момент входа в область танка А будет иметь меньшую временную метку, чем обновление состояния танка В. Таким образом, сообщение танка В будет послано первым.

Скрытые зависимости. Рассмотрим пример боя во время военной кампании. Бой – это синхронизированная (рассчитанная по времени) последовательность действий. Например, ложная атака (маневр) может быть инициирована определенным модулем в момент времени 100, после чего другой модуль инициирует фактическую атаку в момент времени 150. Разумеется, что, планируя операцию, командующий рассчитал ее выполнение так, что ложная атака была инициирована раньше фактической. Но CATOCS не гарантирует, что федерат, представляющий противоположную сторону, получит сообщение о ложной атаке раньше,

чем сообщение об основной атаке. Проблема заключается в том, что все способы упорядочивания сообщений в CATOCS основаны только на порядке передачи сообщений между федератами. Таким образом, семантические отношения между событиями не видны для RTI. С другой стороны, в этом случае упорядочивание по временным меткам должно быть использовано для обеспечения того, что федераты получат сообщения для событий в правильной временной последовательности.

Основным преимуществом каузального упорядочивания по сравнению с упорядочиваем по временным меткам является то, что оно не требует спецификации забегания вперед. Таким образом, каузальный порядок может обеспечить приемлемую альтернативу упорядочиванию по временным меткам для систем моделирования с небольшим забеганием вперед, если использование оптимистической (основанной на откатах) технологии обработки событий невозможно. В дальнейшем ожидается, что в большинстве реализаций каузальное упорядочивание сообщений (по крайней мере, основные сервисы без глобального упорядочивания) будут давать меньшую латентность при взаимодействии и требовать меньшую пропускную способность для реализации, чем упорядочивание по временным меткам.

Забегание вперед. Для сервиса упорядочивания по временным меткам необходима количественная характеристика, называемая забеганием вперед. Для того чтобы объяснить необходимость этой характеристики рассмотрим федерацию, в которой для всех средств передачи определено упорядочивание по временным меткам. Рассмотрим федерат с наименьшим логическим временем в некоторый момент выполнения. Предположим, что значение логического времени этого федерата равно *T*. Этот федерат может генерировать события с временной меткой, равной *T*, релевантные (соответствующие) всем другим системам этой федерации. Это означает, что RTI не может послать ни одному из федератов сообщения с временной меткой, большей *T*. В свою очередь это означает, что ни один из федератов не может продвинуть свое логическое время до значения большего, чем *T*, так как иначе они могли бы получать события в прошлом.

Эта широко известная проблема была глубоко изучена в параллельном дискретно-событийном моделировании. Были разработаны два основных подхода для решения этой проблемы:

- использование идеи забегания вперед, описанной ниже, для определения консервативного протокола, который предотвращает посылку сообщения не по порядку;
- использование технологий оптимистической синхронизации, которая позволяет послать сообщения не в порядке временных меток, и использование федератами механизма откатов для восстановления после ошибок от передачи сообщений не по порядку.

Расскажем более подробно про забегание вперёд.

Если один федерат дает полномочия, что ни одна система моделирования не может планировать событие с временной меткой меньшей, чем текущее время федерата плюс значение *L*, то RTI может позволить одновременную посылку и обработку сообщений во временном промежутке, равном *L* единиц времени и начинающемся с минимального логического времени какой-либо системы этой федерации. Так как пользователь должен иметь возможность планировать момент времени *L* в будущем, или другими словами, предсказывать обновление атрибута и взаимодействие, по крайней мере, через промежуток времени *L*, то это значение *L* в имитационной системе называется забеганием вперед. Вообще забегание вперед сложно включить в какие-то определенные классы систем моделирования, но, тем не менее, оно является очень важным в том случае, когда для приемлемого выполнения необходимы сервисы упорядочивания сообщений.

Очевидно, что забегание вперед очень тесно связано с особенностями имитационной модели, и поэтому автоматически не может быть определено в RTI. Некоторые примеры забегания вперёд мы уже рассматривали при изучении консервативного алгоритма.

Во время моделирования забегание вперед может динамически меняться. Тем не менее, забегание вперед не может быть уменьшено мгновенно. В любой момент времени забегание вперед на *L* единиц показывает RTI, что федерат не сгенерирует событие (при использовании упорядочивания по временным меткам) с временной меткой меньшей, чем *C*+*L*, где *C* – это текущее время федерата. Если забегание вперед уменьшено на *K* единиц времени, то федерат должен продвинуть время на *K* единиц до того как изменение забегания вперед вступит в силу. Таким образом, не может быть сгенерировано ни одно событие с временной меткой меньше, чем *C*+*L*.

В RTI необходимо, чтобы в каждой системе моделирования при генерации событий, использующих сервис гарантированного упорядочивания событий, была определена информация о забегании вперед. При разработке федерата необходимо позаботится о максимизации забегания вперед, так как это значительно влияет не выполнение. Значение для каждого забегания вперед назначается каждым федератом. Это значение может измениться во время выполнения. Но, как замечено выше, уменьшение значения забегания вперед не может вступить в силу моментально.

Сервисы передачи сообщений. Две категории надежности передачи сообщений (надежная и самая надежная) и четыре категории упорядочивания сообщений (упорядочивание по порядку получения, по приоритетам, по временным меткам и каузальное упорядочивание) дают восемь категорий передачи сообщений.

В настоящее время наиболее полезны пять категорий передачи сообщений:

• категория BRec (или категория I): самая надежная передачи сообщений и упорядочивание по порядку получения сообщений;

• категория RRec (или категория II): надежная передачи сообщений и упорядочивание по порядку получения сообщений;

• категория BP (или категория III): самая надежная передачи сообщений и упорядочивание по приоритету;

• категория RTS (или категория IV): надежная передачи сообщений и упорядочивание по временным меткам;

• категория RCO (или категория V): надежная передачи сообщений и каузальное упорядочивание.

Первая буква названия указывает надежность передачи сообщений: самой надежная (B – best effort) или надежная (R – reliable), а оставшиеся буквы указывают тип упорядочивания Rec (receive order) – по порядку получения, P (priority order) – по приоритету, TS (timestamp order) – по временным меткам и CO (causal order) – каузальное упорядочивание.

Категории, обеспечивающие повышение надежности и/или функциональности, увеличивают латентность передачи сообщений или пропускную способность сети, необходимую для поддержания этого сервиса. Поэтому в системах моделирования всегда должны использоваться наименее дорогие сервисы передачи сообщений, соответствующие целям этой системы. Другие категории сервисов передачи сообщений, особенно типов упорядочивания сообщений, могут быть определены в будущем.

В зависимости от того, какие категории сервисов определены, могут быть использованы различные подходы. Например, следующие:

• каждый федерат, когда подписывается на эти данные, может определить категорию сервиса для получаемых им сообщений, таким образом, позволяя различным федератам просматривать входные данные наиболее подходящим для этого федерата способом (этот способ определяется во время подписания федерата на данные);

• каждый федерат может выбрать наиболее подходящую категорию сервиса для каждой посылки сообщений, таким образом, федераты во время выполнения могут разделять генерацию сообщений, используя любые категории сервисов (это позволяет федератам в зависимости от типа передаваемой информации выбирать подходящий сервис);

• каждый федерат может выбирать категорию передачи сообщений во время обновления информации.

С точки зрения перспектив развития моделирования, первый подход является наиболее подходящим, так как он позволяет федератам обрабатывать поступающую информацию наиболее подходящим в данном случае способом. Поэтому именно этот подход используется в HLA. Но в первых версиях RTI этот подход не будет реализован полностью.

Запланированные и отмененные события. Запланированные и отмененные события – это сервисы управления объектами. Для

планирования событий в RTI используются сервисы Update Attribute Values (обновления значений атрибута) и Send Interaction (передачи взаимодействия). Вызов этих сервисов обычно приводит к генерации одного или нескольких сообщений, передаваемых федератам, подписавшимся на запрашиваемую информацию.

Отмена события означает способность федерата отменить (to retract, в оптимистических федератах используется термин cancel) ранее запланированное событие.

Эти простые примитивы дискретно-событийного моделирования часто используются для моделирования прерываний, в том числе приоритетных. Сервисы RTI Update Attribute Values и Send Interaction возвращают дескриптор события, который используется для определения (описания) отменяемого события.

Отмена события может использоваться в сервисе любой категории. Если RTI федерата-получателя принимает запрос на отмену события, небуферизованного в RTI (например, потому что это событие уже передано федерату или оно задержано в сети), этот запрос на отмену направляется самому федерату.

Сервисы продвижения времени. Сервисы продвижения времени обеспечивают федератам средства управления продвижением их логического времени. Как описано ниже, запросы продвижения времени обычно освобождают новые события для федератов.

Time Advance Request (t) (запрос продвижения времени): запрашивает продвижение логического времени федерата до значения *t*, таким образом, будут переданы все входящие сообщения категории III, IV и V федерату с временной меткой меньшей или равной t и все сообщения категории I и II. Сообщения передаются федерату с помощью сервисов Reflect Attribute Values (передать значения атрибутов) и Receive Interaction (Получить взаимодействие) федерата, которые вызываются инфраструктурой RTI. Как будет показано ниже, запрос завершается с помощью сервиса Time Advance Grant (Разрешение продвижения времени). После выполнения этого запроса федерат гарантирует, что сообщения категории IV с временной меткой меньшей $t+L$ (где L – забегание вперед) не будут сгенерированы.

Next Event Request(t, one или all) (запрос следующего события): запрашивает RTI следующее сообщение для события категории IV, при условии, что событие имеет временную метку не больше, чем *T*. Второй параметр задает номера событий, передаваемых федерату (любой категории), возвращаемых этим сервисом, то есть запрашиваются одно или все доступные события. Если в качестве второго параметра указано «all», то федерату, помимо событий категории IV,будут переданы все полученные события категории I и II, события категории III и IV с временными метками не более чем *T*. Если указано «one», то будет передано не более чем одно событие любой категории. События передаются федератам с помощью сервисов Reflect Attribute Values и

Receive Interaction. Если указано «all», то сервис Time Advance Grant выполняет запрос и сообщает федерату, что его логическое время было продвинуто до значения временной метки переданного события категории IV, если такое событие есть, либо до значения, указанного в параметрах сервиса Next Event Request (например, *t*). Если же указано «all», то в результате запроса вызывается сервис (Reflect Attribute Values, Receive Interaction или Time Advance Grant). Если нет сообщений категории IV с временной меткой меньшей или равной *t*, не будет послано таких сообщений в будущем, то сервис Time Advance Grant посылается федерату без посылки каких-либо событий. После выполнения этого запроса федерат гарантирует, что в случае если не будут получены никакие дополнительные события с временными метками меньше, чем *t*, в будущем федерат не будет генерировать никаких сообщений категории IV с временными метками меньше, чем $t+L$ (где L – забегание вперед).

Следующие сервисы федератов вызываются инфраструктурой RTI.

Time Advance Grant: вызов этого сервиса говорит о выполнении приоритетного запроса на продвижение логического времени. А именно, вызов этого сервиса говорит, что:

- RTI послала все сообщения для событий категории IV системы моделирования с временной меткой меньшей или равной времени, указанном в сервисе Time Advance Request;
- нет сообщения для событий категории IV с временной меткой, меньшей или равной времени, указанном в последнем вызове сервиса Next Event Request с параметром «all»;
- нет сообщений для событий категории IV с временной меткой меньшей или равной времени, указанном в последнем вызове сервиса Next Event Request с параметром «one», и нет сообщений для событий любой категории доступной для передачи.

В любом случае логическое время федерата продвигается до значения, указанного в сервисе Time Advance Request или сервисе Next Event Request. Если передается событие категории IV, и при этом какое-либо событие запрашивается с помощью сервиса Next Event Request, то логическое время продвигается до значения времени переданного события категории IV. RTI не передаст другие события категории IV в будущем со временной меткой меньше, чем это последнее значение.

Reflect Attribute Values: RTI вызывает этот сервис для передачи федерату измененного значения атрибута.

Receive Interaction: RTI вызывает этот сервис для передачи федерату нового взаимодействия.

Следует заметить, что, как определено выше, сервисы Time Advance Request, Next Event Request и Time Advance Grant используется только при продвижении логического времени. Продвижение реального времени, конечно, не зависит от выполнения федерата.

Сервисы управления временем в HLA обеспечивают поддержку интеграции систем моделирования с жесткими каузальными требованиями

(например, ALSP) с системами моделирования, в которых достаточно минимального упорядочивания событий (например, распределенные интерактивные системы моделирования).

Существует разные сервисы передачи, начиная от основных сервисов неупорядоченного взаимодействия, например, используемые в распределенных интерактивных системах моделирования, и заканчивая сервисами, обеспечивающими надежную передачу сообщений и/или упорядоченную по временным меткам посылку сообщений для успешной реализации причинно-следственной взаимосвязи в моделируемой системе.

При выполнении одной федерации могут одновременно использоваться различные сервисы передачи сообщений. Это позволяет федератам использовать наиболее подходящие для данного типа передаваемой информации категории сервисов.

Эволюционная разработка федераций возможна благодаря использованию различных категорий сервисов. При этом со временем постепенно в федерации появляются новые возможности (например, более высокая надежность или непротиворечивое упорядочивание).

Структура управления временем в HLA возможно уникальна, так как позволяет объединить системы распределенного интерактивного моделирования, ALSP, параллельного дискретно-событийного моделирования, распределенных операционных систем и т.д.

3.2. Свойства инструментальных средств распределенного моделирования

Как уже упоминалось ранее, «распределение» [9] является одной из основных тенденций компьютерных технологий настоящего времени (другой тенденцией является интеграция), которые применяются для моделирования сложных систем. Это проявляется и в базах данных, и в искусственном интеллекте (например, мультиагентные системы), и в операционных системах. Появление ***распределенных систем имитационного моделирования*** – еще одно проявление этой тенденции.

В качестве ***инструментальных средств*** распределенного моделирования применяют ***параллельные и распределенные вычислительные системы***. Термины «параллельные вычислительные системы» и «распределенные вычислительные системы» часто путают. Их необходимо разделить, чтобы не возникало неоднозначности.

3.2.1. Параллельные вычислительные системы

Термин ***параллельные системы***, как правило, применяется к суперкомпьютерам для того, чтобы подчеркнуть использование многопроцессорной архитектуры. Основными классами архитектур современных параллельных [12] компьютеров являются:

- MP – симметричные мультипроцессорные системы.
- MPP – массово-параллельные системы.

- NUMA – системы с неоднородным доступом к памяти.
- PVP – параллельные векторные системы.

Кластеры – используются в качестве дешевого варианта MPP. В качестве узлов могут выступать серверы, рабочие станции и даже персональные компьютеры. Для связи узлов используется одна из стандартных сетевых технологий. ***Кластеризация*** может осуществляться на разных уровнях компьютерной системы, включая аппаратное обеспечение, операционные системы, программы-утилиты, системы управления и приложения.

3.2.2. Распределённые вычислительные системы

Термин ***распределенная система*** [10] обозначает набор независимых компьютеров.

Пользователи представляют их единой объединённой системой. В этом определении подчеркиваются два момента. Во-первых, все машины автономны. Во-вторых, распределенная система скрывает сложность и гетерогенную природу аппаратного обеспечения, на базе которого она построена. Организация распределенных систем включает в себя дополнительный уровень программного обеспечения (ПО), находящийся между верхним и нижним уровнем. Верхний уровень представляет собой пользователей и их приложения. Нижний уровень – это операционные системы (ОС) (рис. 3.3).

Дополнительное программное обеспечение называют промежуточным (middleware).

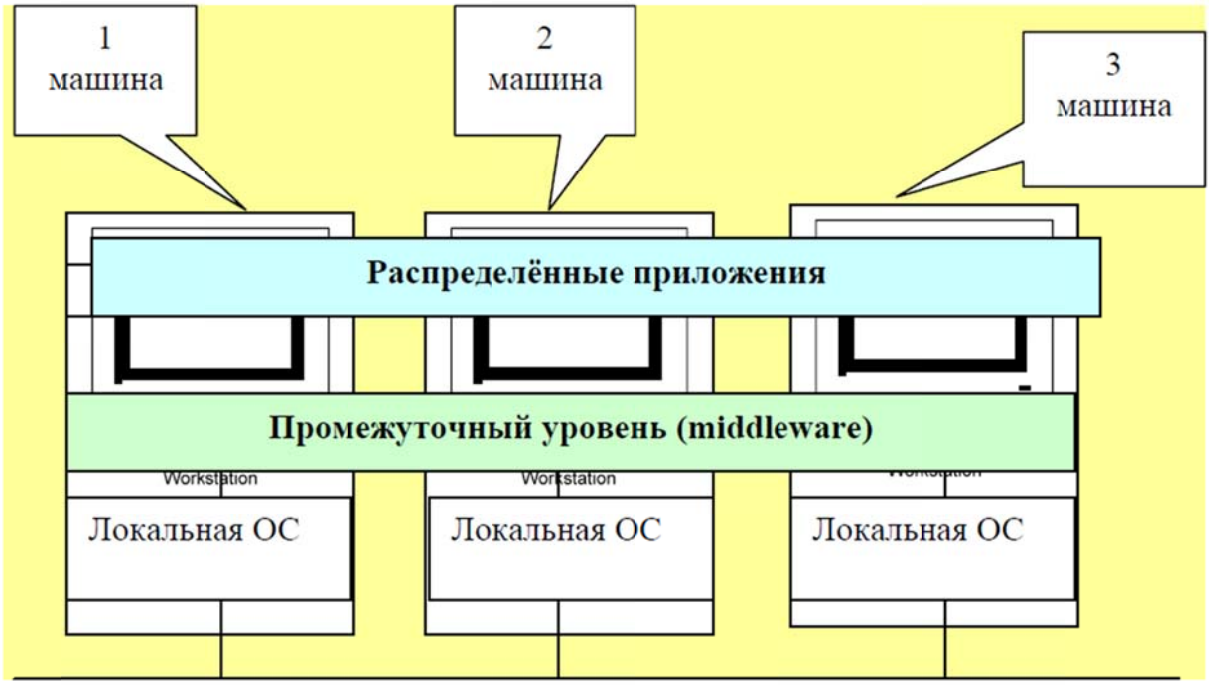


Рис. 3.3. Распределённая система, организованная в виде служб промежуточного уровня

Другие авторы [11] под распределённой системой подразумевают взаимосвязанный набор автономных компьютеров, процессов или процессоров. Компьютеры, процессы или процессоры упоминаются как узлы распределенной системы. Будучи определенными как «автономные», узлы должны быть, по крайней мере, оборудованы своим собственным блоком управления. Таким образом, параллельный компьютер с одним потоком управления и несколькими потоками данных (SIMD) не подпадает под определение распределенной системы.

Поскольку мы определили узлы «взаимосвязанными», то из этого следует, что узлы должны иметь возможность обмениваться информацией. Так как процессы могут играть роль узлов системы, определение включает программные системы, построенные как набор взаимодействующих процессов, даже если они выполняются на одной аппаратной платформе.

Таким образом, распределённой системой можно считать ***локальные*** и ***глобальные*** сети, ***мультипроцессорные вычислительные системы*** и ***взаимодействующие процессы***, которые выполняются на одном компьютере [11].

3.2.3. Характеристики распределённых вычислительных систем

Каждая из этих типов распределённых систем обладает своими отличительными ***характеристиками***, которые следует учитывать при разработке программного обеспечения.

Так ***локальные*** и ***глобальные сети*** отличаются ***характеристиками надёжности***, ***безопасности***, ***временем коммуникации***, ***однородностью*** вычислительных узлов.

Надежность. Под характеристикой надёжности понимают степень надёжности передачи данных в компьютерной сети. Если сеть является глобальной, то вероятность потери переданного сообщения достаточно велика, и алгоритмы должны предусматривать действия, нейтрализующие последствия таких сбоев. Локальные сети более надежны, и алгоритмы для них могут быть разработаны в предположении абсолютной надежности коммуникаций.

Безопасность. При использовании глобальной сети следует уделять большое внимание разработке программных средств защиты вычислительных узлов от внешних атак.

Время коммуникации (время, затрачиваемое на передачу сообщения). Время, затрачиваемое на передачу сообщения в глобальных сетях на порядки больше, чем время передачи в локальных сетях. Временем обработки сообщения в глобальных сетях всегда можно пренебречь при сравнении его со временем, необходимым для передачи.

Однородность. Вычислительные узлы локальных сетей могут отличаться своей производительностью, тем не менее, для обеспечения функционирования сети можно использовать единое программное обеспечение и одни и те же протоколы. В глобальных сетях используется

множество различных протоколов и программное обеспечение, соответствующее разным стандартам. Поэтому следует обращать внимание на преобразование информации при переходе от одного протокола к другому и на совместимость стандартов.

Основное назначение глобальных сетей – это обмен информацией, например, в форме электронной почты, досок объявлений или удаленных файлов. Следует отметить, что глобальные сети всегда организованы как сети типа точка-точка (peer-to-peer).

Рассмотрим более подробно, какие ***проблемы*** могут возникнуть при разработке программного обеспечения ***глобальных сетей***.

Надежность обмена данными по типу точка-точка. Сообщение, которое отослано по линии связи от одного вычислительного узла другому может быть искажено или утеряно из-за падения напряжения в сети или других технических неполадок. Кроме того, сообщения могут быть доставлены с большим опозданием и в порядке, отличном от того, в котором они посылались.

Выбор путей коммуникации. Для взаимодействия двух вычислительных узлов в глобальной сети часто используют промежуточные узлы, и в этом случае актуальным является решение проблемы маршрутизации (выбора пути между взаимодействующими узлами).

Контроль перегрузок. Если сообщения в сети генерируются несколькими узлами одновременно, то сеть становится перегруженной и пропускная её способность заметно падает.

Предотвращение тупиков. Глобальные сети называют ещё сетями типа «сохранить-и-передать», потому что сообщение, которое посылается, минуя несколько промежуточных узлов, должно временно сохраняться в локальной памяти узла, а затем при первой же возможности должно быть отослано. Следовательно, возникает необходимость в управлении памятью, поскольку память вычислительного узла ограничена.

Обеспечение безопасности. Поскольку в глобальной сети может зарегистрироваться любой пользователь и, ко всему прочему, этот пользователь может находиться в любой точке мира, то необходимы надежные методы для аутентификации пользователей, криптографические методы и сканирование входящей информации.

Основное назначение компьютеров, объединённых в локальную сеть – разделение ресурсов (памяти) и повышение скорости вычислений (разделение задачи на подзадачи и выполнение их на разных узлах), а также повышение надёжности вычислительной системы за счёт резервирования некоторых узлов.

При использовании ***локальных сетей*** также возникают ***проблемы распределённого управления процессами***, выполняющимися на разных узлах.

Широковещание и синхронизация. Распределённый алгоритм должен иметь схему передачи сообщений, с помощью которой каким-либо

образом можно «дозваниваться» до всех процессов, с тем, чтобы они могли дождаться выполнения некоторого глобального условия.

Выборность. Распределённый алгоритм должен уметь выбирать один из множества процессов для выполнения конкретной задачи, например, для генерирования вывода или инициализация структуры данных.

Обнаружение завершения. Распределённый алгоритм должен уметь обнаруживать, что произошло завершение распределённых вычислений, тем самым, определив момент получения окончательных результатов.

Распределение ресурсов. При выполнении распределённого алгоритма некоторый узел может затребовать ресурс, не зная, где тот располагается. Для опроса вычислительных узлов о наличии нужного ресурса используют волновые механизмы.

Взаимное исключение. Проблема взаимного исключения возникает при использовании конкурирующими распределёнными процессами общего ресурса, например, принтера или файла, который должен быть перезаписан.

Обнаружение тупиков и их разрешение. При разделении ресурсов некоторыми процессами распределённого алгоритма может возникнуть циклическое ожидание. По этой причине распределённый алгоритм должен предусматривать обходы тупиковых ситуаций.

Распределенная поддержка файлов. При запросах на чтение и запись удаленного файла распределённый алгоритм должен обеспечивать целостность файлов.

Многопроцессорный компьютер представляет собой набор процессоров, объединённых коммуникационной системой. В отличие от компьютерных сетей процессоры однородны и находятся на небольшом расстоянии друг от друга (порядка одного метра или менее). Назначение многопроцессорных компьютеров: повышение скорости вычислений (в этом случае его принято называть параллельным) или повышение надёжности (репликационная система). В параллельном компьютере вычисления поделены на подвычисления, каждое из которых осуществляется одним из узлов. В репликационной системе каждый узел проводит вычисление целиком, после чего результаты сравниваются для того, чтобы обнаружить и скорректировать ошибки.

Рассмотрим ***проблемы***, ***возникающие при выполнении вычислений на многопроцессорном компьютере***.

Разработка системы передачи сообщений. В многопроцессорном компьютере возникают те же проблемы, что и в компьютерной сети: маршрутизация, предотвращение тупиков и перегрузок, но решение проблем упрощается из-за регулярности сетевой топологии.

Балансировка загрузки. Вычислительные узлы должны быть загружены равномерно, иначе нельзя получить выигрыша от применения нескольких вычислительных узлов. Различают статическую (шаги вычислений определяются во время компиляции) и динамическую

балансировки. Очень часто задачу структурируют, и реализация её приводит к разработке сложных программных систем. Однако задачу можно упростить, организовав программу в виде набора (последовательных) взаимодействующих процессов. Каждый процесс при этом является реализацией хорошо определенной, простой задачи.

Приложение, состоящее из взаимодействующих процессов, выполняющихся на одном компьютере, является локально распределенным. ***Взаимодействующие процессы***, имеют доступ к одной физической памяти, при этом возникают хорошо известные ***проблемы при записи в память и чтения информации из памяти***. Эти проблемы разрешаются применением взаимоисключений с использованием разделяемых переменных. Кроме того, возникает проблема сборки мусора.

Итак, под определение распределённых систем подходят следующие типы вычислительных систем, которые находят широкое применение в настоящее время.

Кластер – простая вычислительная система, ресурсы которой используются одной рабочей группой. Это несколько десятков компьютеров, на которых производятся вычисления, объединенных с помощью локальной сети. В отличие от кластера, определенного в параллельных системах, в распределенных системах кластеризация осуществляется только на уровне программного обеспечения.

Вычислительная система корпоративного уровня – это вычислительная система, которая обслуживает несколько групп пользователей, работающих над разными проектами. В такой сети уже необходимо устанавливать правила совместного использования ресурсов, а в некоторых случаях и взаиморасчетов.

Масштаб таких систем, как правило, небольшой, и можно обходиться «ручным» администрированием для организации работы ресурсов и пользователей.

Глобальная система (грид-система) – это система, в которой участвуют несколько отдельных организаций, географически удаленных друг от друга.

Организации предоставляют друг другу свои ресурсы по определенным правилам и с определенными протоколами взаимодействия. Методы «ручного» администрирования в этом случае либо неэффективны, либо неприменимы.

Организационные проблемы можно решить, разработав соответствующее программное обеспечение.

К этому списку следует добавить многопроцессорную вычислительную систему и взаимодействующие процессы, выполняемые на одном компьютере.

Таким образом, употребление терминов зависит от особенностей конкретных систем. При использовании термина «параллельные вычислительные системы» делается акцент на архитектурные особенности, такие как MPP, SMP, NUMA и т.д. Если необходимо описать систему,

состоящую из независимых компьютерных архитектур, работающих как единое целое, и сделать акцент на программное обеспечение, благодаря которому эта работа возможна, то следует употреблять термин «распределенная вычислительная система».

3.2.4. Распределенное имитационное моделирование с использованием модели параллельных вычислений

Современная концепция построения имитационных моделей сложных систем естественным образом вписывается в модель параллельных вычислений MPMD (Multiple Program – Multiple Data). Для реализации на параллельных вычислительных системах распределенного имитационного моделирования сложных систем наиболее подходит именно модель вычислений MPMD по следующим причинам.

Внутренний параллелизм. Сложные системы состоят, как правило, из параллельно функционирующих компонентов. Поэтому имитационная модель сложной системы содержит множество модулей, соответствующих различным элементам и подсистемам моделируемого объекта.

Разнородность подсистем и элементов, составляющих сложную систему, порождает и разнородность математических схем, описывающих функционирование различных элементов. Отсюда следует разнородность программной реализации модулей ИМ.

Большие вычислительные затраты в имитационном моделировании сложных систем связаны с размерностью исследуемых систем, а также объемом вычислений для получения статистически обоснованных результатов.

Для переноса имитационной модели на распределенную вычислительную платформу, на первый взгляд, кажется естественным просто разместить составные модули по процессорам вычислительной системы. Однако простое распределение модулей моделирующей программы по процессорам не будет способствовать эффективному ускорению процесса имитации из-за многочисленных передач информации между компонентами имитационной модели, сильно связанных по управлению. Так как пересылка данных наиболее «времяемкий ” процесс в параллельных вычислениях на многопроцессорных системах.

Следовательно, если в качестве модели вычислений выбрать MPMD, то необходимо решить задачу декомпозиции программного обеспечения (ПО) имитационной модели (ИМ) на блоки, по критерию минимизации обмена данными между блоками ИМ. Для решения этой задачи проведем следующие рассуждения.

1. ИМ сложной технической системы представляет собой стохастическую дискретно-событийную модель (ДСМ), созданную в соответствии с принципами построения моделирующих алгоритмов.

Имитационная модель сложной технической системы строится, как множество процессов, исполняемых квазипараллельно в условном времени

и взаимодействующих между собой через общие наборы данных. ИМ имеет частично упорядоченное в модельном времени множество событий. Событие – это изменение состояния системы. В обрабатывающих модулях, соответствующих наступлению событий, вычисляются метки готовых к обработке событий.

Такие события заносятся в список (календарь) событий. Управляющая программа упорядочивает эти метки по не убыванию модельного времени и устанавливает модельное время равным метке первого события, а затем передает управление соответствующему обрабатывающему модулю, где это событие обрабатывается. В результате появляются новые готовые к обработке события. Они в правильном порядке заносятся в календарь. Причем события со временем, меньшим текущего появиться не могут. За один шаг имитации имитируется только одно событие, т.е. выполняется один обрабатывающий модуль.

Обозначим через $E=\{e_1, e_2,..., e_n\}$ – множество всех возможных событий в модели; через $E^*=\{s_1, s_2,..., s_k,..., s_m\}$ – набор всех возможных конечных последовательностей событий $s_k = e_1, e_2,..., e_{ki}$ в E. Последовательность $s = e_1, e_2, e_3$ означает, что событие e_{i+1} имеет место после события e_i и описывает логическое поведение моделируемой системы.

Временная последовательность s учитывает времена наступления событий $s = (e_1, t_1) (e_2, t_2) (e_3, t_3) ..., t_1 \pi t_2 \pi t_3...$ и описывает поведение системы во времени.

Стохастическая дискретно-событийная модель, выполняет последовательности вида: $s(w)=(e_1 (w), t_1 (w)) (e_2 (w), t_2 (w)) (e_3 (w), t_3 (w)) ...$, где e_i ,t_i – случайные переменные, w - выборочная точка, так что $s(w)$ – выборочный путь развития моделирующего процесса. Каждый выборочный путь является здесь временной последовательностью, и стохастическая ДСМ определяет распределение вероятностей по набору всех временных последовательностей.

Выполнение программы имитационной модели на обыкновенном компьютере есть воспроизведение последовательности дискретных событий $s(w)$ и обработка этих событий. Реализация конкретной последовательности определяется тем, как срабатывают условные операторы. В свою очередь срабатывание условных операторов зависит исключительно от входных данных и распределений вероятностей наступления событий e_i. Поэтому при моделировании на обыкновенном компьютере всегда выполняется какая-то последовательность действий, которая однозначно определена программой, реализующей моделирующий алгоритм, входными данными и распределениями вероятностей наступления событий, которые так же относятся к входным данным. В рассматриваемом случае на каждом шаге имитации может выполняться только один обрабатывающий модуль ИМ.

2. В случае выполнения имитационной программы на многопроцессорной системе на каждом шаге имитации одновременно

может выполняться целый ансамбль обрабатывающих модулей, расположенных на разных процессорах и независимых по данным в рассматриваемый момент времени. Моделирующий алгоритм программы и входные данные должны однозначно определять состав ансамблей.

Как на обыкновенном, так и на параллельном компьютере, искомые оценки характеристик моделируемой системы получают в результате выполнения последовательности обрабатывающих модулей (ОМ) имитационной модели, которая, в свою очередь, определяется последовательностью наступления событий $s(w)=(e_1(w), t_1(w))(e_2(w), t_2(w))(e_3(w), t_3(w))$ ... Все ОМ имеют некоторое число аргументов. Конкретными значениями аргументов ОМ являются либо входные данные, либо результаты выполнения других ОМ. Любой обрабатывающий модуль – потребитель аргументов не может начать выполняться раньше, чем закончится выполнение всех ОМ - поставщиков для него аргументов. То, какие модули являются поставщиками аргументов для конкретного обрабатывающего модуля, зависит от событий, свершившихся на предыдущем шаге. Необходимо заметить, что у одного и того же ОМ на разных шагах имитации могут быть разные поставщики.

Следовательно, взаимная связность событий на множестве $E=\{e_1, e_2,..., e_n\}$ определяет конкретную последовательность реализаций ОМ. А взаимосвязь событий, в свою очередь, зависит от структуры моделируемой системы, правил функционирования, распределений вероятностей и частоты взаимодействия элементов системы – все это вместе составляет входные данные имитационной модели.

3. Следовательно, задача распараллеливания имитационной программы состоит в том, чтобы на множестве всех возможных событий $E=\{e_1, e_2,..., e_n\}$ выделить группы событий, которые наиболее связаны по входным данным. Это возможно вследствие того, что в некотором событии е участвуют не все элементы моделируемой системы, и его наступление приводит лишь к локальному изменению состояния системы, то есть при свершении события следует анализировать изменения состояний лишь части элементов системы, то есть обращаться только к соответствующим им записям в базе данных ИМ.

Поясним на примере. Пусть имеется множество возможных событий $E = \{e_1, e_2,..., e_r\}$ между которыми существует такая связь: свершение события e_1 обязательно влечет за собой событие e_3 , а свершение события e_2 влечет за собой событие e_r , то есть e_1 и e_3 более сильно связаны, чем e_1 и e_2 . При этом моментами времени наступления событий e_1, e_2, e_3, e_r являются, соответственно t_1, t_2, t_3 и t_r, причем $t_1 \le t_2 < t_3 \le t_r$.

Тогда, при выполнении имитационной программы на обыкновенном компьютере реализуется последовательность действий: $s(w)=(e_1(w), t_1(w))(e_2(w), t_2(w))(e_3(w), t_3(w))$ ... $(e_r(w), t_r(w))$.

Так как e_1 и e_2 обрабатываются модулями, независимыми по входным аргументам, то реализацию последовательности $s(w)$ на параллельном компьютере можно осуществить, объединив события e_1 и e_3

в группу $E^1=\{e_1, e_3\}$, а события e_2 и e_r в группу $E^2=\{e_2, e_r\}$, $E = E^1 Y E^2$. Соответственно, последовательность действий $s(w)$ разбивается на последовательности $s^1(w)=(e^1{}_1\ (w), t_1\ (w))\ (e^1{}_3\ (w), t_3\ (w))$ и $s^2(w)=(e^2{}_2\ (w), t_2\ (w))\ (e^1{}_r\ (w), t_r\ (w))$, которые можно выполнить на разных процессорах. Взаимосвязь между процессами $s^1(w)$ и $s^2(w)$ сохраняется в виде обмена сообщениями, который необходим для синхронизации модельного времени, но обмен будет минимальным ввиду отсутствия обращения к общим наборам данных.

Схема проведенных рассуждений показана на рис. 3.4.

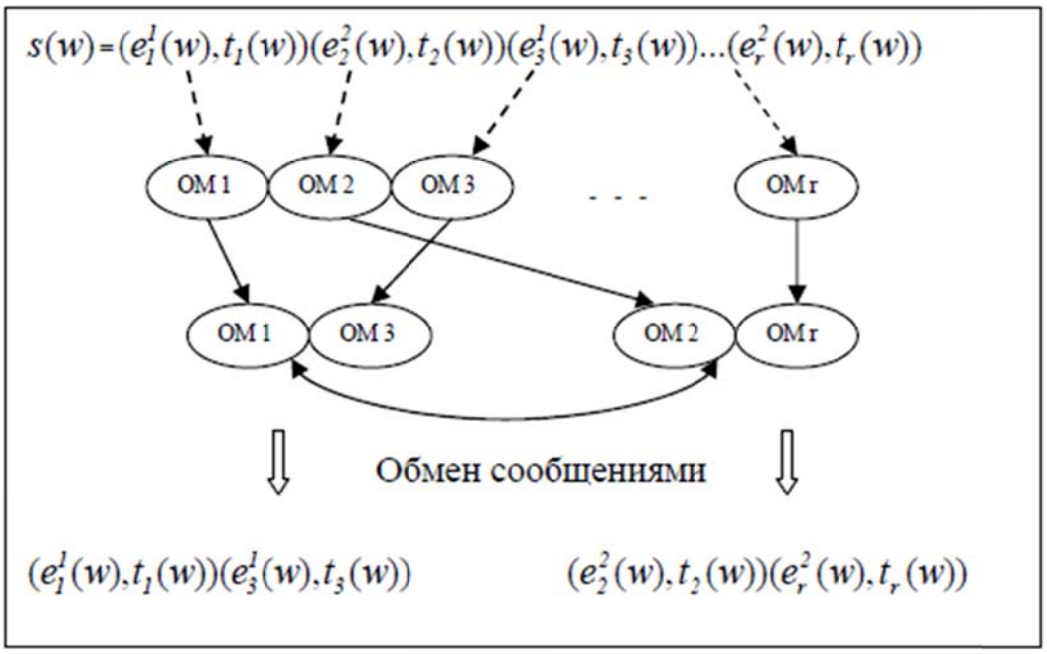


Рис. 3.4. Последовательное и параллельное выполнение моделирующего алгоритма

Таким образом, для декомпозиции программного обеспечения ИМ по критерию минимизации обмена данными необходимо выделить в множестве всех возможных событий $E=\{e_1, e_2,..., e_n\}$ такие группы событий E^1, E^2,…, E^m , чтобы взаимосвязь событий внутри группы была максимальной, а между группами минимальной.

Это можно осуществить, применив метод автоматической классификации объектов.

Разбиение E^1, E^2,…, E^m позволит сформировать блоки ИМ, состоящие из обрабатывающих модулей событий, принадлежащих этим группам, а так же распределенную базу данных.

Для сохранения глобальной логики поведения модели требуется обеспечить синхронизацию работы модельных блоков, выполняющихся на разных процессорах, так как минимизированный обмен данными сохраняется. Схематическая иллюстрация последовательностей моделирования на обыкновенном компьютере и многопроцессорной системе приведена на рис. 3.5.

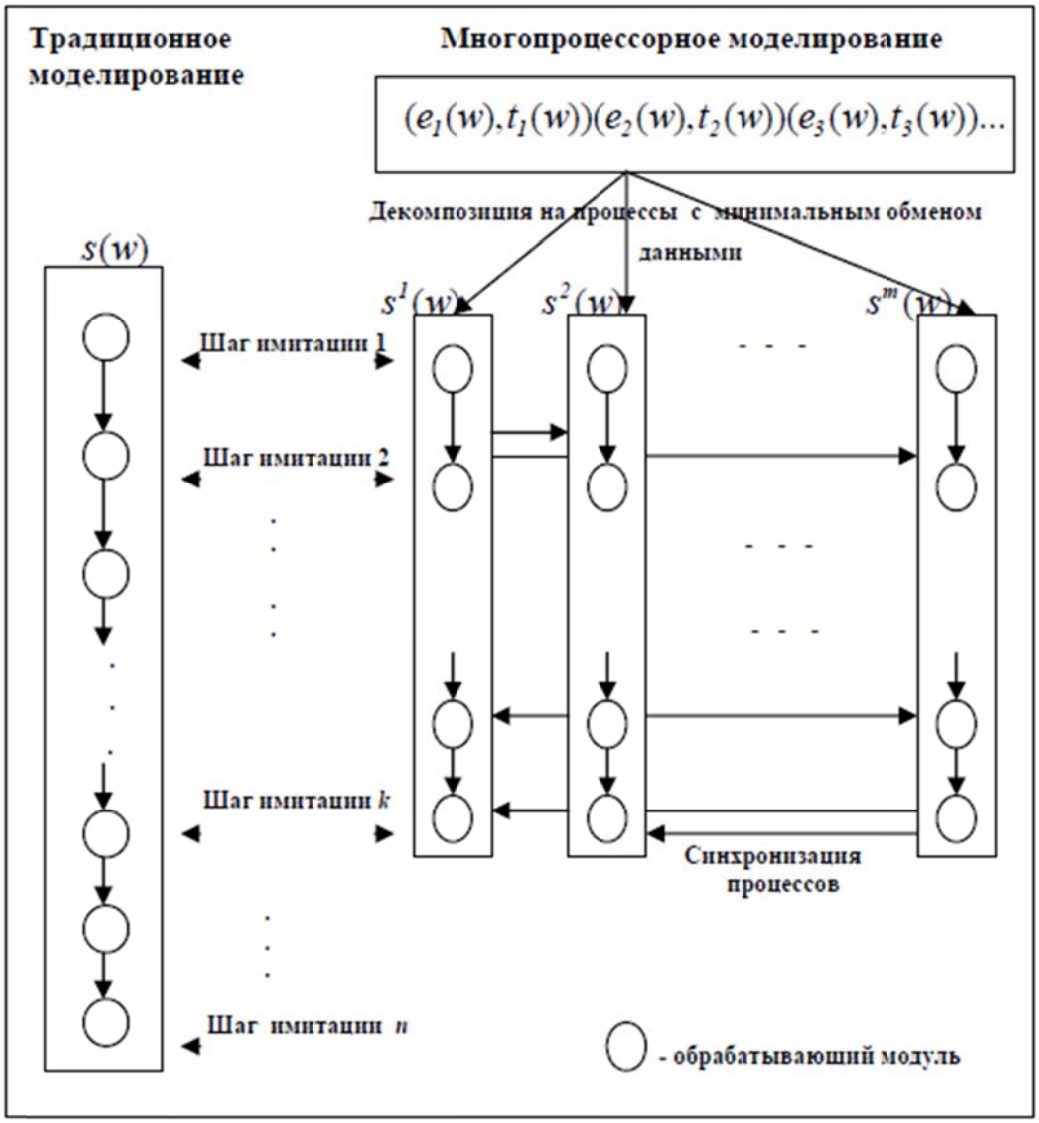


Рис. 3.5. Схемы последовательного и распределенного моделирования

Так как в качестве параллельного вычислителя для имитационного моделирования сложной технической системы выбрана локальная сеть или многопроцессорный кластер, встает задача оптимального распределения полученных блоков ИМ по процессорам неоднородной вычислительной сети, характеризующейся различной производительностью процессоров. Необходимо так распределить блоки ИМ, чтобы общее время выполнения распределенной имитационной модели было минимальным по всем возможным вариантам размещений. Правильная подстройка задачи под вычислительную систему предполагает размещение подзадач по процессорам, учитывающее специфику архитектуры и конфигурации конкретной вычислительной системы.

Итак, основные принципы организации распределенного имитационного моделирования сложной технической системы с использованием модели вычислений MPMD заключаются в следующем.

1. Декомпозиция программного обеспечения ИМ на блоки, по критерию минимизации обменов сообщениями, должна проводиться на основе выделения групп наиболее связанных по входным данным обрабатывающих модулей ИМ.

2. Сохранение глобальной логики поведения модели в едином времени требует обеспечения синхронизации работы блоков ИМ, выполняющихся на разных процессорах компьютерной сети.

3. Блоки ИМ должны размещаться по процессорам вычислительной системы, по критерию минимизации общего времени выполнения моделирования.

Далее излагаются методы решения перечисленных задач.

Эти методы предназначены для автоматизированного распараллеливания программного обеспечения ИМ и решения обозначенных задач по организации распределенного имитационного моделирования. Они являются алгоритмическим наполнением технологических этапов программного инструментария, способного оказать исследователю помощь за счет выдачи результатов измерений отдельных параметров, оценивающих качество проекта распределенной (параллельной) имитационной программы. Такими параметрами являются оценки потенциального параллелизма, т.е. прогнозируемого ускорения вычислительного процесса распределенного имитационного моделирования. Программный инструментарий назовем автоматизированным средством организации распределенного имитационного моделирования (АСОРИМ). АСОРИМ предназначено для отделения аспектов, связанных с разработкой программного обеспечения ИМ, от вопросов организации параллельных (распределенных) вычислений.

3.3. Классификация систем распределенного имитационного моделирования

3.3.1. Общие свойства систем распределенного моделирования

В настоящее время известно более шестисот наименований систем, отдельных средств и языков распределенного имитационного моделирования [10]. Их главной особенностью является ориентация на конечного пользователя, выражающаяся в очень высокой эффективности и доступности для пользователей-непрофессионалов, желающих самостоятельно решать свои задачи. Подробный обзор коммерческих продуктов для имитационного моделирования можно найти на сайте ежегодной конференции Winter Simulation Conference, а также в [9, 10]. В наиболее часто публикуемый список «первой десятки» языков и пакетов, широко применяемых сегодня для моделирования производственных и логистических систем, входят GPSS/H, SIMULA, SLX, Arena, AutoMod, Extend, ProModel, QUEST, SIMFACTORY II.5, SIMPLE++ (eMPlant), Tailor

ED, WITNESS. Все эти системы моделирования обладают в той или иной степени общими свойствами.

1. **Поддержка работы программиста на этапах создания и отладки моделей**. Здесь предпочтение отдается идеографическому способу формирования модели, т.е. модель первоначально создается в виде 2D или 3D-структуры из заранее подготовленных графических компонентов, работая в режиме Drag&Drop.

2. **Возможности, предоставляемые пакетом для создания анимации**. Анимационные пакеты могут быть уже интегрированы в систему имитации, также немаловажно существование возможности использования анимационных пакетов сторонних разработчиков (например, 3D Max, True Spase, Light Wave, SPAZZ_3D, ISA). Наибольшее предпочтение в современных системах имитации отдается пакету VRML, как наиболее открытому для сопряжения с другими программными средствами.

3. **Способность к интеграции с другими программными системами**. Здесь можно выделить три аспекта интеграции:

- взаимодействие с различными информационными базами данных и программными приложениями, позволяющее вводить исходные данные из соответствующих баз данных или в форме Exell – таблиц;
- возможность работы с моделями через Internet или Intranet. Средства Java Applet или ActiveX дают пользователю возможность, работая с обычным браузером, подготавливать, реализовывать и анализировать имитационные эксперименты;
- возможность использования моделей в системах поддержки принятия решений (моделирование рассматривается здесь, как один из инструментов поддержки принятия соответствующих технических или экономических решений).

Резюмируя, можно утверждать, что все современные системы имитационного моделирования имеют больше аналогий, чем различий, являются, прежде всего, объектно-ориентированными, что позволяет сравнительно легко создавать любые адекватные решаемой задаче классы статических и динамических объектов.

Их главной особенностью является ориентация на конечного пользователя и самый широкий рынок потребителей. Их несомненное достоинство заключается в доступности и наглядности работы модели, хотя анимация для рассматриваемого нами класса задач имеет второстепенное значение. Важнейшей остается задача оценки показателей функционирования системы, которые в конечном итоге используются для решения соответствующей прикладной задачи. Научный базис моделирования в них остался прежним. Решение по-настоящему трудных задач на их основе ничуть не облегчилось. Как образно выразился известный специалист по имитационному моделированию Ю.Толуев [9]: «…дерево имитационного моделирования стало настолько большим, что оно перестало помещаться только лишь в пространстве «искусство и

наука», как это было на момент опубликования книги Роберта Шеннона в 1975 году. Оставив свои корни в этом пространстве, значительная часть дерева находится в пространстве «техника и бизнес», где и можно сейчас наблюдать активный рост и цветение его кроны. Этот факт хочется подчеркнуть еще и потому, что в области «корней» активного роста в последние годы не наблюдается: фундаментальные основы имитационного моделирования не подвергаются пока ревизии и существенно не расширяются…».

Проблемы распределенного имитационного моделирования, связанные с быстродействием, особенно для систем большой размерности, планированием имитационных экспериментов, применением моделей в «контуре управления», т.е. в реальном масштабе времени, все еще остаются.

3.3.2. Проблемы распределенного имитационного моделирования сложных систем

Несмотря на совершенствование современных имитационных систем, их интеграцию с последними достижениями информационных технологий, проблемы, связанные с ускорением процесса моделирования, минимизацией вычислительных затрат без потери точности результатов моделирования, требуют решения.

Рассмотрим некоторые из них [9, 10].

Большой объем вычислений. Метод имитационного моделирования характеризуется тем, что решение всегда носит частный характер. Оно соответствует фиксированным значениям параметров системы и начальных условий. Для анализа системы приходится многократно моделировать ее процесс функционирования, варьируя исходные данные задачи. Вычисления различных интегрированных характеристик процессов, как правило, требуют отображений вида $F: \Omega \rightarrow Z$, где Ω – множество траекторий поведения системы; Z – множество значений характеристик системы. Отсюда следует, что также требуется многократное проведение экспериментов по получению траекторий модели системы, а затем осуществление преобразования F. В этих условиях, особенно в случае решения задач статистического характера, требуется выполнить огромный объем вычислений, что требует эффективных методов реализации моделирования на ЭВМ.

Размерность. Наиболее полное исследование общесистемных проблем можно получить, моделируя объекты большого масштаба в целом, в условиях, близких к реальным. Для того, чтобы модель давала описание, хорошо отражающее реальность, она необходимо должна быть достаточно сложна. Но в таком случае машинный эксперимент требует больших затрат машинного времени и памяти, а большое количество экспериментов – необходимое условие любого анализа. Проведение же экспериментов с малым числом объектов, т.е. понижение размерности

исследуемой системы, искажает результаты, так как оно создает нереальные условия функционирования системы. Ограниченность ресурсов используемой вычислительной техники являлась в недавнем прошлом едва ли не центральной задачей исследования, требующей упрощения модели, замены ее другой, более доступной для анализа.

Разномасштабность. Трудности, связанные с воспроизведением модели на ЭВМ определяются не только ее размерностью, еще более трудной оказывается организация вычислений характеристик разномасштабных процессов, составляющих исследуемое явление, когда в системе появляются процессы с малыми характерными временами, т.е. такие, которые описываются быстро изменяющимися переменными. Анализ подобных моделей требует огромных затрат машинного времени. Здесь также требуется упрощение модели. Но упрощения модели не должны при этом сопровождаться потерей качественных особенностей – модель должна сохранить свое соответствие реальности.

Проблемы, связанные с размерностью, ресурсоемкостью, вычислительными затратами решаются различными способами. Это разработка методов декомпозиции моделируемых объектов и декомпозиции задач анализа систем, применение методов регенеративного моделирования, методов понижения дисперсии [7, 9], разработка комбинированных и аналитико-статистических методов [7, 9, 10].

Рассмотренные проблемы имитационного моделирования тесно связаны с понятием технической реализуемости метода. Под технической реализуемостью метода понимают возможность решения задачи данным методом на выбранных инструментальных средствах в приемлемое для пользователя время.

Техническая реализуемость прямо связана с понятием технической сложности, под которой понимается $Q = P \times T$, где P – требуемый объем оперативной памяти; T – требуемое количество вычислений или машинное время для реализации метода на выбранной инструментальной системе [7]. Выход за приемлемые границы технической сложности может привести к отказу от выбранного метода решения задачи и к корректировке частных моделей. Такая корректировка может потребовать решения технологических задач оптимизации моделей системы в части технической сложности. Методы решения этих задач базируются на использовании операций эквивалентного преобразования моделей, заданных в фиксированных формальных схемах, например, эквивалентных преобразований в теории автоматов, или для агрегативных систем [1, 6, 7]. Не исключено использование и других, специализированных методов понижения технической сложности. Например, возможен отказ от прямых алгоритмов имитации и использование аналитико-имитационных алгоритмов, приводящих к существенному ускорению процесса моделирования [5].

Расширение возможностей моделирования различных классов больших систем неразрывно связано с совершенствованием средств

вычислительной техники [8]. Основным барьером на пути решения проблем распределенного имитационного моделирования является уровень эффективного и полноценного применения средств вычислительной техники. Эти проблемы могут быть решены лишь на достаточно мощной аппаратной базе с применением эффективных методов программирования.

3.3.3. Два основных класса систем распределенного имитационного моделирования

Расширение возможностей моделирования различных классов больших систем неразрывно связано с совершенствованием средств вычислительной техники [8]. Основным барьером на пути решения проблем распределенного имитационного моделирования является уровень эффективного и полноценного применения средств вычислительной техники. Эти проблемы могут быть решены лишь на достаточно мощной аппаратной базе с применением эффективных методов программирования.

Выделим два основных класса современных систем распределенного имитационного моделирования.

Монолитные высокоэффективные системы параллельного дискретно-событийного моделирования. В настоящее время успешно развивается новое направление в компьютеризации – массово-параллельные вычислительные системы. Производительность наиболее мощных вычислительных установок – суперкомпьютеров возрастает, ориентировочно, на порядок за пятилетие. Однако огромная производительность параллельных компьютеров и супер-ЭВМ компенсируется сложностями их использования. Реализовать потенциальные возможности таких систем возможно только на основе целенаправленной работы как по адаптации существующих, так и по построению новых алгоритмов, созданию инструментальных средств выявления скрытого параллелизма, осуществления вычислительной балансировки работы вычислительных модулей сование структуры программ и алгоритмов с особенностями архитектуры параллельных вычислительных систем [1, 2, 6].

Кроме того, стоимость суперкомпьютеров очень высока, например, при стоимости микропроцессора порядка тысячи долларов, для зарубежных суперкомпьютеров стоимость, в пересчете на процессорный узел, составляет 10-100 тысяч долларов. Эти причины обусловили бурное развитие нового направления: построение многопроцессорных параллельных вычислительных систем – кластеров.

Разнородные системы распределенного моделирования, объединенные с помощью специального программного обеспечения. Разнородные системы распределенного моделирования, реализованные на многопроцессорных параллельных вычислительных системах – ***кластерах***, построенных на базе стандартных общедоступных технологий и компонентов массового производства с использованием

стандартизованных в мировой практике интерфейсов и программных средств. Причем, как оказалось, на многих классах задач эти системы дают производительность, сравнимую с производительностью классических суперкомпьютеров [6].

С другой стороны, ***локальные и глобальные вычислительные сети*** тоже стали рассматриваться как дешевая альтернатива дорогим суперкомпьютерным установкам. Связано это с улучшением технико-экономических показателей (сочетание «цена / производительность») персональных компьютеров и сетевого оборудования. Начиная с 1992 года, по скорости роста производительности сетевое оборудование обгоняет процессоры. Для многих организаций перспектива использования сетей рабочих станций и персональных компьютеров в качестве суперкомпьютеров весьма заманчива. Сети не могут соревноваться по скорости с супер-компьютерами-рекордсменами, но они на несколько порядков дешевле, их можно использовать там, где объемы расчетов велики, а суперкомпьютеры экономически не оправданы [5]. Вычислительные узлы локальных сетей могут отличаться своей производительностью, тем не менее, для обеспечения функционирования сети можно использовать единое программное обеспечение и одни и те же протоколы. В глобальных сетях используется множество различных протоколов и программное обеспечение, соответствующее разным стандартам. Поэтому следует обращать внимание на преобразование информации при переходе от одного протокола к другому и на совместимость стандартов. Основное назначение глобальных сетей – это обмен информацией, например, в форме электронной почты, досок объявлений или удаленных файлов. Следует отметить, что глобальные сети всегда организованы как сети типа точка-точка (peer-to-peer).

Развитие сетевых технологий способствует использованию сетей в качестве параллельных вычислителей. В первую очередь это Fast Ethernet и Gigabit Ethernet, также технологии коммутации и сетевые протоколы, поддерживающие широковещание и мультикастинг. Используемые в сетях Ethernet и Token Ring протоколы IPX, SPX , NetBIOS позволяют организовать взаимодействия между программами, работающими на разных станциях в сети без обращения к файловому серверу [6, 9]. Одноуровневые взаимодействия в сети могут быть использованы в самых разных пользовательских приложениях, а также могут быть использованы в качестве основы для сложных распределенных систем обработки. Сетевая прикладная программа может быть спроектирована так, что часть работы выполняют компьютеры, действующие, как вспомогательные функциональные процессоры.

Разработка способов построения алгоритмов распределенного имитационного моделирования с распределенной (параллельной) обработкой в локальной вычислительной сети и кластерных системах, использование их, как технической базы для распределенного моделирования сложных систем, позволит решить проблемы

ресурсоемкости и вычислительных затрат при имитационном моделировании.

3.4. Основные направления в развитии систем распределенного моделирования

Задача распределенного имитационного моделирования сложных систем относится к классу задач, требующих больших вычислительных ресурсов. В связи с этим направления развития систем распределенного моделирования обусловлены развитием ***инструментальных средств***: ***высокопроизводительных вычислительных систем*** и ***компьютерных сетей***.

Можно выделить **два основных направления** в развитии систем распределенного моделирования:

- развитие монолитных высокоэффективных системы параллельного дискретно-событийного моделирования, построенных на базе высокопроизводительных вычислительных систем;
- развитие разнородных систем распределенного моделирования, объединенных с помощью специального программного обеспечения, построенных на базе локальных и глобальных компьютерных сетей.

3.4.1. Направления развития и классификация высокопроизводительных вычислительных систем как инструментальных средств монолитных высокоэффективных систем параллельного дискретно-событийного моделирования

В настоящее время развитие высокопроизводительных вычислительных систем идет по четырем направлениям [2].

1. **Векторно-конвейерные компьютеры.** Особенностью таких машин являются конвейерные функциональные устройства и набор векторных инструкций в системе команд. Векторные команды оперируют целыми массивами независимых данных, что позволяет эффективно загружать доступные конвейеры. Характерным представителем данного направления является семейство векторно-конвейерных компьютеров CRAY, куда входят: CRAY EL, CRAY J90, CRAY T90.

2. **Массивно-параллельные компьютеры с распределенной памятью.** К данному классу можно отнести компьютеры Intel Paragon, IBM SP1, Parsitex, IBM SP2, CRAY TD3/T3E.

3. **Параллельные компьютеры с общей памятью.** Вся оперативная память таких компьютеров разделяется несколькими одинаковыми процессорами. В данное направление входят многие современные многопроцессорные SMP-компьютеры или, например, отдельные узлы компьютеров HP Exemplar и Sun Star Fire.

4. Последнее направление представляет собой **комбинация предыдущих трех**. Из нескольких процессоров (традиционных или векторно-конвейерных) и общей для них памяти формируется вычислительный узел. Если полученной вычислительной мощности недостаточно, то объединяются несколько узлов высокоскоростными каналами. Подобную архитектуру называют кластерной, и по такому принципу построены CRAY SV1, HP Exemplar, Sun Star Fire, NEC SX-5, последние модели IBM SP2 и др. Именно это направление в настоящее время является наиболее перспективным для конструирования компьютеров с рекордными показателями производительности.

Основным параметром ***классификации*** параллельных компьютеров является наличие общей (SMP) или распределенной памяти (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем.

Рассмотрим основные классы современных высокопроизводительных вычислительных систем с целью определения тех средств, которые наиболее подходят в качестве технической базы для решения задач распределенного имитационного моделирования сложных систем.

1. **Симметричные мультипроцессорные системы** (SMP). Система состоит из нескольких однородных процессоров и массива общей памяти. Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины, либо с помощью crossbar-коммутатора (HP 9000). Примеры: RM 600 E, HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compag, Dell, ALR, Unisys, DG, Fujitsu и др.) Вся система работает под управлением единой ОС (Обычно UNIX-подобной, но для некоторых платформ поддерживается Windows NT). ОС автоматически в процессе работы распределяет процессы-нити по процессорам, но иногда возможна и явная привязка.

Основное преимущество SMP – относительная простота программирования. В ситуации, когда все процессоры имеют одинаково быстрый доступ к общей памяти, вопрос о том, какой из процессоров какие вычисления будет выполнять, не столь принципиален. И значительная часть вычислительных алгоритмов, разработанных для последовательных компьютеров, может быть ускорена с помощью распараллеливающих и векторизующих трансляторов. SMP-компьютеры – это наиболее распространенные сейчас параллельные вычислители. Однако число процессоров ограниченно, так как увеличение не дает выигрыша из-за конфликтов при обращении к памяти.

2. **Массивно-параллельные системы** (MPP). Система состоит из однородных вычислительных узлов, включающих: один или несколько центральных процессоров (обычно RISC), локальную память (прямой

доступ к памяти других узлов невозможен), жесткие диски. К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.). Примеры: IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, RM 1000 (Siemens-Piramid). Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain). На каждом узле работает полноценная UNIX - подобная ОС (вариант близкий к кластерному подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле.

Каждый процессор имеет доступ лишь к своей локальной памяти, а если программе нужно узнать значение переменной, расположенной в памяти другого процессора, то задействуется механизм передачи сообщений. При создании программ необходимо учитывать топологию системы и специальным образом распределять данные между процессорами, чтобы минимизировать число пересылок и объем пересылаемых данных. Это обстоятельство мешает широкому внедрению подобных архитектур. Из-за того, что MPP-системы включают различное число процессоров, объединенных коммуникационными подсистемами различной топологии и пропускной способности, поддерживающими разные наборы примитивов, программы, стремящиеся максимально полно использовать возможности компьютера, оказываются жестко привязанными к его архитектуре.

3. **Системы с неоднородным доступом к памяти** (NUMA). Система состоит из однородных базовых модулей, состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной памяти. Примеры: HP, HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPS 10000, IBM/Sequent NUMA-Q 2000, SNI RM600. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000). Обычно вся система работает под управлением единой операционной системой, как в SMP.

В рассмотренной архитектуре главной особенностью является виртуальная общая память. Каждый процессор может обращаться напрямую только к своей локальной памяти, однако все узлы используют единое адресное пространство. Если программа обратилась по адресу, принадлежащему локальной памяти другого процессора, генерируется аппаратное прерывание по особому случаю адресации, и операционная система выполняет пересылку страницы с одного узла на другой. В результате можно ускорять программы, первоначально разработанные для последовательных систем, транслируя их с помощью распараллеливающего компилятора. Благодаря чрезвычайно высокому быстродействию коммуникационной системы (пиковая скорость передачи

данных между двумя узлами составляет 300Mb/sec в каждом направлении) этот подход в целом оправдывает себя.

Однако когда на одну страницу виртуальной памяти попадает несколько «популярных» переменных, считываемых и модифицируемых множеством процессоров, возникает эффект «пинг-понга»: страница непрерывно мигрирует с узла на узел и производительность системы падает.

3.4.2. Направления развития и классификация компьютерных сетей как инструментальных средств разнородных систем распределенного моделирования

В качестве инструментальных средств разнородных систем распределенного моделирования применяются кластерные системы, локальные и глобальные компьютерные сети.

1. **Кластерные системы**. Набор рабочих станций (или даже персональных компьютеров общего назначения), используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о гетерогенных (неоднородных) кластерах. Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку. Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.

Прогресс в области сетевых технологий, появление недорогих, но эффективных коммуникационных решений предопределили появление кластерных вычислительных систем, фактически являющихся одним из направлений развития компьютеров с массовым параллелизмом (MPP). Вычислительный кластер – это совокупность компьютеров, объединенных в рамках некоторой сети для решения одной задачи. Кластерные системы во многом похожи на MPP, однако, имеются существенные отличия [2, 3, 9]. Прежде всего, в большинстве случаев архитектурные особенности связаны с применением обычных компьютерных сетей, что обусловливает высокие накладные расходы при межмашинном взаимодействии. Кроме того, за счет менее жестких требований к составу технического и программного обеспечения имеется возможность построения кластера из неоднородных узлов (так называемые гетерогенные кластеры) – с различной программно-аппаратной организацией и производительностью.

Данная особенность снимает привязку к конкретной платформе, но, с другой стороны усложняет задачу разработки общесистемного и прикладного программного обеспечения для подобных систем. В последнее время все большую популярность приобретают системы NOW

(Network of Workstations), которые можно классифицировать как один из видов систем с кластерной обработкой. Многие фирмы предлагают готовые решения по кластеризации на основе либо собственных разработок (Sun, HP, DEC, NCR, SGI), либо с применением стандартных компонентов высокой готовности и распространенного базового ПО (Linux/Beowulf, Avalon). Несмотря на имеющиеся недостатки и ограничения кластеров, наблюдается устойчивая тенденция к переходу на эти относительно дешевые вычислительные архитектуры, представляющие реальную альтернативу дорогим суперкомпьютерам.

2. **Локальные и глобальные компьютерные сети**. Локальная компьютерная сеть представляет собой набор персональных компьютеров общего назначения, объединенных средой передачи. В локальных компьютерных сетях для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Token Ring). Ethernet, на базе шинной архитектуры или коммутатора, широко используется в локальных сетях. Token Ring реализует технологию передачи маркера, использует кольцевую или звездообразную топологии. Среда передачи определяется стандартами сетевых технологий, для высокоскоростных технологий это витая пара UDP-5, оптоволоконный кабель. Используются стандартные для локальных сетей сетевые операционные системы, чаще всего, Windows NT, Novell NetWare.

Глобальные компьютерные сети обеспечивают возможность работы нескольких исследователей над одним и тем же имитационным проектом через Интернет. Проект может выполняться на суперкомпьютере или на вычислительной системе с кластерной архитектурой и, таким образом, исследователи используют вычислительные ресурсы суперкомпьютера, наблюдают за ходом имитационного эксперимента и просматривают результаты, находясь в разных городах или странах.

Кроме того, используя распределённую систему моделирования с удалённым доступом через Интернет, можно сдать в «аренду» неиспользуемые в данный момент времени вычислительные ресурсы.

В глобальных сетях используется множество различных протоколов и программное обеспечение, соответствующее разным стандартам. Поэтому следует обращать внимание на преобразование информации при переходе от одного протокола к другому и на совместимость стандартов. Основное назначение глобальных сетей – это обмен информацией, например, в форме электронной почты, досок объявлений или удаленных файлов. Следует отметить, что глобальные сети всегда организованы как сети типа точка-точка (peer-to-peer).

Локальные и глобальные компьютерные сети стали рассматриваться как дешевая альтернатива дорогим суперкомпьютерным установкам. До последнего времени основным препятствием для применения компьютерных сетей в качестве параллельных вычислителей было отсутствие инструментария. Для создания сетевых приложений можно использовать интерфейс сокетов. Хотя существует мнение, что интерфейс

сокетов – слишком низкоуровневый для большинства применений. Так в [5] замечено, что написать и отладить программы, основанные непосредственно на интерфейсе сокетов, не легче, чем написать вручную и отладить программу в машинных кодах. Поэтому крупным достижением считалось создание и стандартизация интерфейса передачи сообщений MPI (Message Passing Interface). MPI поддерживает несколько режимов передачи данных: синхронную и асинхронную передачу, коллективные операции, широковещательную передачу. Библиотеки MPI реализованы практически на всех современных суперкомпьютерах, а также могут использоваться и в кластерных системах, и в локальных сетях, т. е. пригодны для использования в различных системах с распределенной памятью.

В [7] приведены результаты исследований, связанных с выбором программных средств для организации параллельных вычислений в компьютерной сети. Были испытаны пакеты PVM, MPI. Предварительные испытания пакетов показали значительное преимущество MPI как по возможностям, так и по скорости обмена данными между процессами. С помощью MPI была написана параллельная программа, реализующая численное моделирование выбранного объекта исследования. После проведения серии экспериментов для решения задачи повышенной сложности авторы пришли к выводу, что для повышения эффективности (уменьшения накладных расходов при передаче данных между процессами) необходимо использовать протоколы более низкого уровня.

При всех достоинствах MPI, попытка обеспечения удобного сервиса для прикладного программиста не совсем благоприятно сказывается на характеристиках эффективного исполнения функций обмена сообщениями, не способствует уменьшению времени организации обмена сообщениями между процессами, что особенно важно для локальных сетей. Так как стандартные программные интерфейсы, как в стиле MPI, так и языковые в стиле стандарта HPF сами построены на базе эффективных низкоуровневых библиотек обмена сообщениями. Исходя из этого, в качестве основы для сложных распределенных систем обработки лучше использовать низкоуровневые взаимодействия в сети на уровне сокетов, не пугаясь сложности программирования на этом уровне. Здесь большее значение имеет то, насколько грамотно будет спроектирована сетевая программа, а расстановка процедур передачи и приема сообщений по уже определенным местам в программе не представляет задачи большой сложности. Можно воспользоваться и сервисом, предлагаемым Windows Sockets.

При таком разнообразии параллельных вычислительных систем, тем не менее, следует отметить, что до сих пор применение параллелизма не получило столь широкого распространения.

Одной из причин подобной ситуации являлась до недавнего времени высокая стоимость высокопроизводительных систем.

Современная тенденция построения параллельных вычислительных комплексов из типовых конструктивных элементов, массовый выпуск которых освоен промышленностью, снизила влияние этого фактора. Другая и, пожалуй, теперь основная причина сдерживания массового распространения параллелизма в том, что для проведения параллельных вычислений необходимо «параллельное» обобщение традиционной последовательной технологии решения задач на ЭВМ. Алгоритмы решения задач на многопроцессорных системах должны проектироваться, как системы параллельных и взаимодействующих между собой процессов, допускающих исполнение на независимых процессорах.

Таким образом, наиболее доступной технической платформой для решения проблемы вычислительных затрат в имитационном моделировании являются кластерные системы и компьютерные сети.

3.5. Примеры систем распределенного моделирования

3.5.1. Системы распределенного моделирования, использующие язык XML

XML (eXtensible Markup Language – расширяемый язык разметки) [9] возник в середине девяностых как новый язык для World Wide Web. Его предшественником является стандартный общий язык разметки - SGML (Standart Generalised Markup Language), который был утвержден ISO в качестве стандарта в 1986 году. Также как и SGML, XML является метаязыком разметки, то есть он предназначен для создания других (проблемно-ориентированных) языков разметки. Такие языки разметки можно создавать на основе XML, определяя допустимый набор символов разметки, или «тэгов», их атрибуты и внутреннюю структуру документа. XML рациональнее и проще, чем SGML, но поддерживает его функциональность.

Язык разметки HTML (HyperText Markup Language – гипертекстовый язык разметки), созданный на основе SGML, является на сегодняшний день самым популярным языком гипертекстовой разметки. Язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор тэгов, при помощи которых осуществляется процесс разметки. Тэги HTML, в первую очередь, предназначены для управления процессом вывода содержимого документа на экран web-браузера и определяют этим самым способ представления документа, но не его структуру. Сами отображаемые данные никак не связаны с теми тэгами, которые используются для форматирования, поэтому нет возможности использовать тэги HTML для поиска нужных нам фрагментов документа. Таким образом, можно сказать, что HTML не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. И ему на смену был предложен новый язык разметки, мощный, гибкий, и, одновременно с этим, удобный язык XML.

Достоинства языка XML. XML – это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. XML-документы содержат структурированную информацию. Сам по себе XML, в отличие от HTML, не содержит никакого ограниченного набора тэгов, предназначенных для разметки, он просто определяет порядок их создания. Автор XML-документа создает его структуру, строит необходимые связи между элементами, используя свои собственные пользовательские тэги и структурные отношения, которые удовлетворяют его требованиям, и добивается такого типа разметки, который необходим ему для выполнения операций просмотра, поиска, анализа документа.

С помощью XML можно организовать более качественный поиск, поскольку данные в XML идентифицированы с помощью тэгов однозначно. Это свойство XML дает возможность использования его в качестве универсального языка запросов к хранилищам информации. Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и представления ее на стороне клиента.

XML также позволяет осуществлять контроль данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные.

Также одним из достоинств XML является то, что программы-обработчики XML-документов просты в написании. Все это дает основания предполагать, что, скорее всего, в ближайшем будущем XML станет основным языком обмена информации для информационных систем, заменив собой, тем самым, HTML.

Язык XML применяется в распределенном имитационном моделировании и имитационном программном обеспечении. Наиболее частым использованием XML является создание стандартизированных форматов данных для базовых файлов ввода и вывода, применяющихся в имитационном моделировании. Открытые, стандартизированные представления для таких данных, как определенные пользователем и/или эмпирические распределения, последовательный во времени вывод, статистические оценки производительности и т.д. являются полезными для межплатформенных анализа, вычислений и обмена результатами.

Другим способом применения языка XML в имитационном моделировании является его использование в качестве основы для файлов, в которых хранится имитационная модель. Пользователи не знают, что отличает новый формат файла от старого, и поэтому они не меняют привычные для них способы ввода информации. Применение XML-файлов в пакетах имитационного программного обеспечения дает преимущества стандартизованного доступа к данным, облегчает добавление новых возможностей и плагинов, создает условия для интеграции программных продуктов.

Существовали попытки использования XML для связи между имитационной моделью и другим программным обеспечением, а также для связи между имитационными моделями. Qiao, Riddick и McLean (2003) [6] и Lu, Qiao и McLean (2003) [7] описывают использование разработанного NIST языка разметки, основанного на XML, для стандартизации обмена производственной информацией между приложениями, включая имитационные приложения. В дополнение к этому, Fishwick (2003) [9] описывает два языка (MXL и DXL), которые были разработаны на основе XML для применения в имитационном мультимоделировании.

Kilgore (2001,2002) представляет язык SML (Simulation Modeling Language – Язык имитационного моделирования), который стал первой попыткой создать независимый от платформы, имитационный язык с открытым кодом. Wiedemann (2002)[95] развивает эту идею, предлагая независимый от языка стандарт для представления операторов SML, основанный на XML. Для генерации операторов конкретного языка (например, Java, C++) из первоначальных XML-операторов во время исполнения модели, он предлагает использовать конвертеры кода. Альтернативный имитационный проект, основанный на XML, описывается Reichenthal (2002) [11]. Этот проект предполагает создание языка разметки SRML, чтобы стандартизировать описания имитационных моделей, и имитатора SR Simulator для того, чтобы создавать и выполнять SRML-модели.

Ниже мы более подробно рассмотрим данные способы применения XML в имитационном моделировании.

Проект NIST

Любая система имитационного моделирования должна осуществлять эффективное управление данными. Наибольших успехов в этом направлении на сегодняшний день удалось добиться в моделировании производственных процессов. В данном разделе речь пойдет о спецификации имитационного интерфейса, разрабатываемой Национальным институтом стандартов и технологий (NIST).

Чтобы разработать систему имитационного моделирования, которую можно было бы использовать в моделировании производственных процессов, нужно решить проблему информационной интеграции. Информация, которая нужна для производственного процесса, часто разрознена и хранится в различных источниках данных, таких, как базы данных, системы PDM (Product Data Management – Управление данными о продукции), созданные компьютером или от руки чертежи, простые текстовые и бинарные файлы и крупноформатные таблицы. Подобная информация может храниться на разных компьютерах, не всегда бывает полной и может иметь несовместимые форматы. В то же время информация, имеющая одинаковый формат, может предназначаться для разных целей. Все это ведет к проблемам хранения и восстановления информации, а также обмена информацией между имитационными системами и другими производственными приложениями.

Для того чтобы решить некоторые из этих проблем NIST разработал информационную модель, хранящуюся в файле обмена, формат которого основан на XML. Использование языка XML облегчает обмен производственной информацией между производственными системами имитационного моделирования и другими производственными приложениями и/или источниками данных.

Вся информация, которая необходима имитационному процессу, находится в файле обмена, обозначаемого аббревиатурой SDF (Shop Data File). Он основан на информационной модели SDIM (Shop Data Information Model). Модель SDIM содержит описания важных элементов производственных операций, атрибуты этих элементов и отношения между этими элементами. Для создания SDIM используются схемы XML и статические структурные диаграммы языка UML (Unified Modeling Language – унифицированный язык моделирования). Статические структурные диаграммы используются для графического описания модели, в то время как схемы XML используются для текстового описания модели, что облегчает создание SDF-файлов.

С помощью транслятора, SDF-файл транслируется в файл, с которым непосредственно работает система имитационного моделирования. Для этой трансляции в данном случае необходим основанный на XML язык XSL (eXtensible Stylesheet Language – Язык расширяемых таблиц стилей), который как раз и используется для перевода XML-документов в файлы других форматов.

SDF-файл содержит не только исполняемые и исчислимые данные, которые обрабатываются в процессе имитации, но также и наглядную информацию, которая предназначена непосредственно для человека. Он также содержит сеть перекрестных ссылок между различными типами данных, которая требуется для того, чтобы планировать и управлять операциями. SDF-файлы поддерживают ссылки на внешние компьютерные файлы и/или бумажные документы, что предоставляет более подходящие механизмы для кодирования и отображения информации.

Информационная модель SDIM содержит четыре вспомогательных структуры данных и пятнадцать производственных структур данных. Элементы данных модели SDIM могут быть трех видов: ключевые элементы данных (data element keys), обычные элементы данных (commonly used data elements) и уникальные производственные элементы данных (unique manufacturing shop data elements). Ключевые и обычные элементы данных могут быть найдены на всех уровнях структур данных внутри модели.

Ключевые элементы данных служат индивидуальными указателями или коллекциями указателей на модель данных. Обычные элементы данных делятся на основные элементы данных, элементы данных с префиксами/суффиксами и сложные элементы данных.

Сложные элементы данных состоят из нескольких основных элементов, которые в свою очередь больше не делятся ни на какие элементы.

Одним из преимуществ использования SDF-файла является то, что при изменении требований, имитационная модель может быть быстро изменена для выполнения анализа согласно новым данным.

Описанная выше разработанная NIST, основанная на XML спецификация имитационного интерфейса дает возможность улучшения методов имитационного моделирования.

Проект Rube

В данном разделе обсуждается использование языка XML в мультимоделировании. Термин «мультимодель» может означать следующее:

- многочисленность: схема моделирования поддерживает более одной модели;
- разнородность: схема моделирования поддерживает более одного типа модели;
- иерархичность: схема моделирования поддерживает иерархию моделей;
- настраиваемость: схема моделирования поддерживает альтернативные представления для одного и того же типа модели.

Проект rube, поддерживает все четыре аспекта мультимоделирования, но особое внимание при его разработке уделялось поддержке настраиваемости. Создатели проекта rube ставили перед собой задачу разработать систему, которая максимально облегчала бы создание динамической мультимодели, а также ее дальнейшее повторное использование в трехмерной среде. Они использовали язык XML как средство для описания модели и семантики ее представления.

Ниже приведена структура проекта rube (рис. 3.6).

На входе имеются два файла: файл представления (scene file) и файл модели (model file). Файл модели описывает свойства и топологию элементов динамической модели, в то время как файл представления описывает, как выглядят эти элементы и что собой представляют связи между ними.

Файл представления может быть представлен в одном из трех видов. Например, в проекте rube можно представить модель в формате 3D, 2D и 1D (причем под 1D понимается простое текстовое представление модели). Для представления динамической модели в 3D используется основанный на XML язык X3D, для представления модели в 2D – язык SVG (Scalable Vector Graphics – Масштабируемая векторная графика). Оба языка поддерживают встроенное программирование на JavaScript, с помощью которого можно задать способ движения объектов модели. Для представления модели в 1D используется смесь таких языков, как HTML, XHTML (XML-версия языка HTML) и MathML.

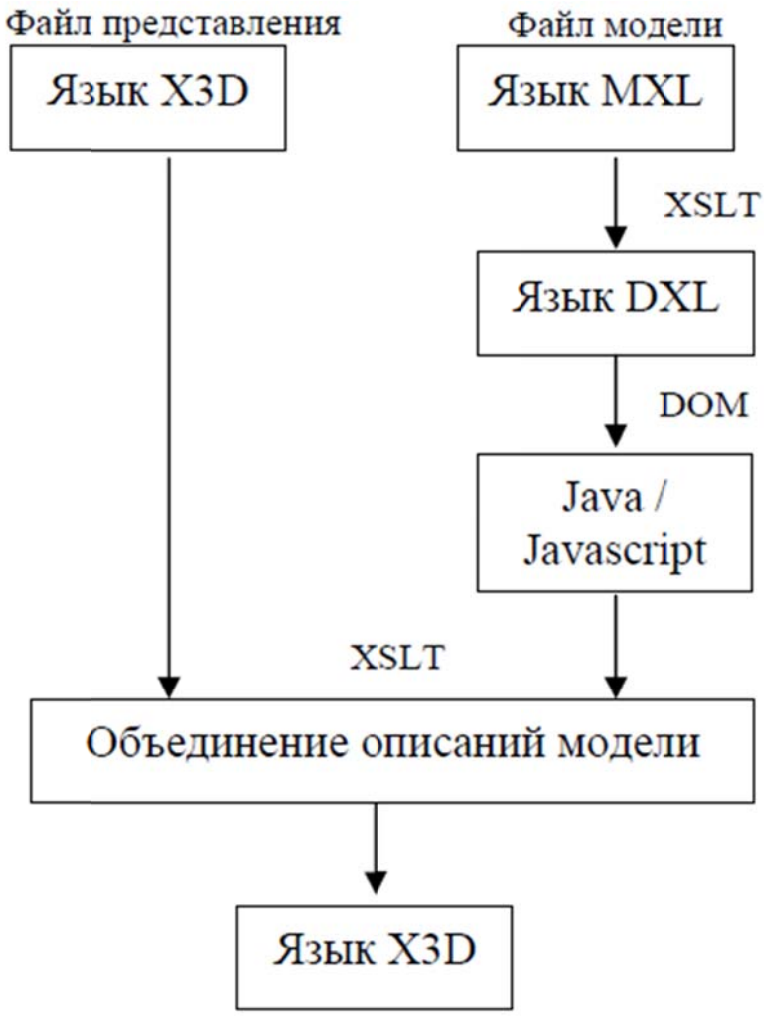


Рис. 3.6. Структура проекта Rube

После определения желаемого внешнего вида объектов модели описывается формальная структура модели с помощью основанного на XML языка MXL (Multimodeling eXchange Language). Это описание хранится в файле модели.

На следующей стадии с помощью языка XSL текст файла модели с языка MXL транслируется в текст на языке DXL (Dynamics eXchange Language – Язык обмена динамики).

В то время как текст на языке MXL содержит описания всех элементов модели и их атрибутов, текст на языке DXL содержит описание блоков, составленных из этих элементов.

Каждый блок содержит входные и выходные порты и иногда описания локальных переменных. Блоки соединены между собой с помощью коннекторов.

Текст файла модели на языке DXL транслируется непосредственно на язык Java с использованием технологии DOM (Domain Object Model). На стадии объединения описаний модели из файла представления и Java-кода, созданного на основе файла модели, получается представление модели с внедренным кодом.

Проект rube использует XML в качестве языка ядра, поэтому на примере этого проекта можно увидеть все плюсы и минусы использования языка XML в подобном качестве.

К ***достоинствам*** можно отнести то, что XML отделяет содержание от представления. Также плюсом является распространенность языка XML в web-сообществе. Это гарантирует, что язык XML еще долго будут использовать, или, по крайней мере, на его основе будет создано что-то еще более мощное и гибкое.

К ***недостаткам*** можно отнести потери времени при обработке XML-документов, а также невозможность предсказать, какие типы документов XML приобретут популярность, а какие перестанут использоваться в скором времени.

Проект OpenSML

Проект OpenSML был представлен на Зимней Конференции по моделированию в 2001. Этот проект основан на языке SML (Simulation Modeling Language – Язык имитационного моделирования) и является web-проектом с открытым кодом.

Язык SML предназначен для создания повторно используемого имитационного программного обеспечения.

Требование ***повторного использования*** подразумевает под собой как минимум то, что код будет ***удобочитаемым***, состоящим из ***модулей*** и ***расширяемым***.

Требование ***удобочитаемости кода*** означает, что аудитория, для которой предназначен код, ближе по своему уровню к человеку с очень ограниченными знаниями в программировании, чем к опытному хакеру.

Требование ***модульности*** означает, что разработчик может произвести изменение в коде целого SML-модуля или даже полностью заменить этот модуль, и при этом ему не нужно вносить изменения в код других модулей. Ему даже не обязательно понимать смысл их кода.

Требование ***расширяемости*** означает, что SML спроектирован так, чтобы его можно было легко приспосабливать для использования в приложениях.

Для выполнения этих и других требований было решено в качестве базового синтаксиса языка SML использовать XML. Использование XML сделало SML независимым от синтаксиса всех известных до этого языков программирования.

Единственной проблемой при использовании XML оказалось описание динамических действий и событий. Описание их полностью на XML было длинным и сложным для понимания. Поэтому решено было использовать как XML-структуры, так и традиционные операторы.

Для того чтобы иметь возможность ***транслировать*** SML-программы с любого SML-диалекта (например, Java-SML) на язык SML-XML, и обратно, была разработана полностью автоматическая программа трансляции кода. Она использует в своей работе шаблоны кода XML, а для

проверки правильности XML-документов используются DTD. Схема трансляции приведена на рис. 3.7.

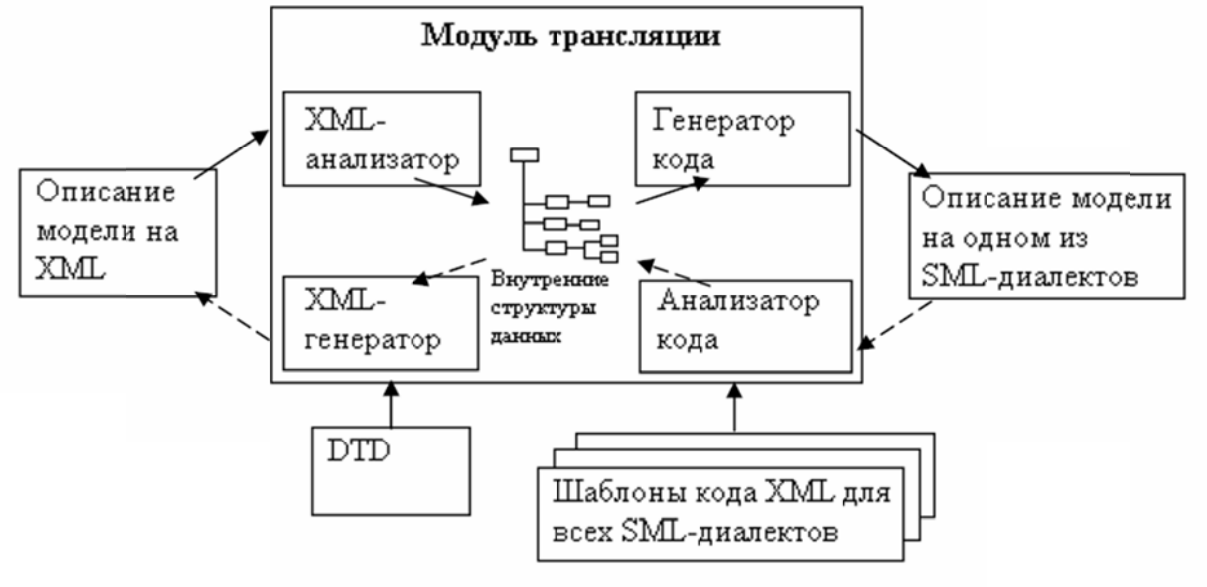


Рис. 3.7. Схема трансляции

С использованием такой схемы трансляции, которая включает два последовательных процесса трансформации кода, имитационная модель может быть передана между двумя различными платформами совсем без каких-либо ручных изменений. Это вместе с основной идеей языка SML, дает основания полагать, что в ближайшем будущем появится универсальная и не зависящая от языка система имитационного моделирования.

Язык SRML

Архитектура HLA (High Level Architecture – Архитектура высокого уровня) была разработана в середине девяностых Министерством Обороны США как метод объединения нескольких автономных имитационных моделей в одну распределенную имитационную систему. Обсудим, как язык SRML (Simulation Reference Markup Language) используется совместно с HLA для достижения высочайших уровней взаимодействия между имитационными моделями и их повторного использования.

HLA представляет собой стандартное средство, используя которое отдельные имитационные модели могут взаимодействовать между собой. При этом взаимодействующие модели имеют общий документ FOM (Federation Object Model), в котором описываются все типы информации, которой они обмениваются. Документы FOM структурированы по правилам OMT (Object Model Template – Шаблон модели объекта). Часть документа FOM, в которой описывается отдельная модель, называется SOM (Simulation Object Model). В дальнейшем данная имитационная модель может быть повторно использована в другой имитационной системе, если ее SOM будет включена в FOM этой системы.

Структура документа FOM является достаточно сложной, поэтому в сообществе разработчиков HLA было создано несколько способов, облегчающих создание документов FOM. Среди них BOM (Base Object Models). Отдельный объект BOM – это документ, который содержит связанную и внедренную информацию, построенную по правилам OMT, а также другие мета-данные, официально не включенные в OMT. Группа индивидуальных документов BOM, которая называется Mega-BOM, может быть преобразована в документ FOM. Таким образом, облегчается повторное использование и разработка документов FOM.

С помощью XML могут описываться данные для BOM и Mega-BOM, включая символьные данные и мета-данные. Однако использование простого XML не дает возможности описать поведение данных. Основанный на XML язык SRML предоставляет средства для описания этого поведения.

Большинство современных языков программирования, которые используются в имитационном моделировании, хорошо приспособлены для представления программной логики модели, но плохо описывают символьные и другие данные. Плюс их структура предполагает отделение программной логики от данных, что создает препятствия для повторного использования модели. Разработчики SRML поставили перед собой целью создать язык программирования, который, на равных работая как с программной логикой, так и с данными, устранял бы эти недостатки.

3.5.2. Системы распределенного моделирования, использующие онтологии

Важной проблемой имитационного моделирования является возможность использования объектов модели в других моделях для упрощения их создания и поддержки. Для этого может использоваться некоторая библиотека объектов, из которых новые модели строятся как дом из строительных блоков. Однако даже если базовые объекты можно легко настроить, компоновка модели и ее настройка для конкретного случая могут быть очень трудоемкой задачей.

Также существует проблема определения уровня описания базовых объектов. При выборе высокого уровня описания, потребуется отдельное описание множества сложных объектов, большая часть из которых будет использоваться очень редко, при этом нужно четко определять в каком случае должен быть использован каждый из них. При этом для каждого конкретного случая придется не только правильно выбрать базовые объекты из библиотеки, но и настраивать их взаимодействие. На низком уровне описания библиотека стандартных объектов будет невелика, она будет состоять лишь из элементарных элементов, например, очередь, генератор заявок и т.п., однако, это практически не упростит создания моделей.

Другой вариант решения – ***пользоваться некоторыми знаниями***, как про моделируемую систему, так и про различные примитивы, из которых строится модель.

Обычно, модель сама по себе не несет смысловой нагрузки, т.е. смысл модели и ее интерпретация в основном существует лишь в разуме ее создателя. Генерируемые при моделировании графики и числа означают или предсказывают некоторые значения существующей в реальности или предполагаемой системы. В настоящее время большую роль играет графическое представление (визуализация), поскольку оно способствует лучшему пониманию модели.

Однако, остается вопрос, как представить модель таким образом, чтобы она была понятна не только человеку, но и машине, чтобы позволить на основе знаний о моделях и моделируемой области автоматическую генерацию, или хотя бы достраивание моделей.

Понимание модели человеком строится не только на ее визуальном представлении, но и на том, как новая информация соотносится с уже существующим знанием, каково ее место в семантическом пространстве. Для того чтобы передать эти знания системе моделирования, нужен некоторый способ их представления. В области искусственного интеллекта для этих целей используются онтологии.

Онтология – формальное описание предметной области, задающее общий словарь для определения концептов и взаимосвязей этих концептов в конкретной предметной области, а так же для описания объектов, поведения и знаний, включающихся в эту предметную область.

Онтологии создаются во множестве областей знаний. Например, в последнее время было создано несколько онтологий для биологического домена. Создание онтологий для моделирования более сложно по двум причинам [10]:

- моделирование не ограничено конкретным доменом, поскольку модели могут представлять биологические, химические, физические, транспортные, военные и т.п. системы;
- моделирование и его методы основаны на математике, вероятностных и статистических расчетах, и, таким образом, должно их придерживаться, поэтому онтологии для этих областей должны служить основой для всех остальных (т.н. онтологии среднего уровня).

Таким образом, задача по созданию общей онтологии для всех возможных моделируемых систем является весьма трудоемкой.

Для успешной автоматизации достраивания моделей не обязательно иметь общие знания о мире в целом [10]. Альтернативой может быть создание онтологии, охватывающей домен моделей и содержащей информацию о том, как примитивы модели (например, генераторы, очереди, стоки и т.п.) взаимодействуют друг с другом, какие основные классы более крупных объектов существуют и для чего они используются. При существовании такой онтологии, достаточно будет дополнить ее

сведениями об обычных приемах моделирования в той или иной предметной области без полного ее описания.

Для создания онтологии необходим некоторый инструмент, позволяющий быстро и удобно описывать, просматривать и искать описанные в ней классы. Для того чтобы онтологию можно было использовать для автоматизации достраивания или полного построения модели необходим некоторый механизм вывода необходимых знаний из онтологии, механизм принятия решений.

Средства представления онтологий

Ключевым моментом в проектировании онтологий является выбор соответствующего ***языка спецификации онтологий*** (Ontology specification language). Цель таких языков – предоставить возможность указывать дополнительную машинно-интерпретируемую семантику ресурсов, сделать машинное представление данных более похожим на положение вещей в реальном мире, существенно повысить выразительные возможности концептуального моделирования слабоструктурированных Web-данных.

Существуют традиционные языки спецификации онтологий Ontolingua, CycL, языки, основанные на дескриптивных логиках, такие как LOOM, и языки, основанные на фреймах – OKBC, OCML, Flogic. Более поздние языки, основанные на Web-стандартах, такие как XOL, SHOE или UPML, RDF(S), DAML, OIL, OWL созданы специально для обмена онтологиями через Web.

В целом, различие между традиционными и Web- языками спецификации онтологий заключается в выразительных возможностях описания предметной области и некоторых возможностях механизма логического вывода для этих языков. Типичные примитивы языков дополнительно включают:

- конструкции для агрегирования, множественных иерархий классов, правил вывода и аксиом;
- различные формы модуляризации для записи онтологий и взаимоотношений между ними;
- возможность мета-описания онтологий, что полезно при установлении отношений между различными видами онтологий.

Первыми предложениями по описанию онтологий на базе RDFS были DARPA DAML-ONT (DARPA Agent Markup Language) и European Commission OIL (Ontology Inference Layer). Эти стандарты спецификации и обмена онтологиями полезны для процесса обмена знаниями и интеграции знаний. DAML обеспечивает примитивы для объявления пересечений, объединений, дополнений классов и т.д. OIL основан на описании логик. Другое расширение RDFS - DRDFS. Также как OIL, он дает возможность для выражения классов и определения свойств, однако выразительная мощность языков DRDFS и OIL такова, что ни один из них не может быть рассмотрен как фрагмент другого.

На базе этих предложений DAML и OIL возникло совместное решение – DAML+OIL, которое послужило толчком для создания в рамках инициативы Semantic Web отдельной группы по пересмотру этого решения и стандартизации языка описания Web-онтологий (OWL - Web Ontology Language). Адаптация к Web систем логики и искусственного интеллекта составляет вершину «пирамиды Semantic Web», обеспечивая адекватный семантически поиск информации и машинную интерпретацию семантики.

OIL также можно рассматривать в сравнении с Ontolingua, разработанной в рамках инициативы On-To-Knowledge. По сравнению с Ontolingua, OIL менее выразителен, но все же позволяет делать логические выводы: поддержка вывода обеспечивается системой FaCT – классификатором, который работает на основе описания логик.

Стандарт BOM

BOM (Base Object Models) – развивающийся XML стандарт для быстрого создания моделей и средств моделирования, базирующийся на концепциях онтологий. BOMы можно назвать переиспользуемыми пакетами информации, представляющими шаблон взаимодействия моделей (интерфейс). Этот шаблон отражает множество действий концептуальных сущностей, используемых для достижения общей цели, возможности или предназначения [4].

Путем подбора шаблонов и определения компонентов, моделирующих необходимые для поддержки этих шаблонов возможности, можно обеспечить создание «строительных блоков» для модели.

Для описания модели существует два вида BOMов.

Interface BOMs – описывают интерфейсы взаимодействия элементов модели. Они создаются за счет анализа взаимодействия активностей объектов системы. Interface BOMs используются для:

- обеспечения общего описания уровня метаданных, называемого идентификацией модели;
- описания активностей, используемых в шаблоне;
- определения событий и ключевых элементов объектной модели (в применении к системе HLA это HLA OMT 1516).

Encapsulated BOMs – описывают поведение объектов системы, создаются за счет анализа предполагаемого поведения концептуальных сущностей, на основе событий ассоциируемых с шаблонами взаимодействия. Они используются для:

- обеспечения общего описания уровня метаданных, называемого идентификацией модели;
- описания состояний или действий концептуальных сущностей некоторого шаблона.

Ключевые преимущества BOMов – ориентирование на использование с HLA – широко используемый стандарт взаимодействия систем моделирования, и хранение данных на базе XML, что позволяет

описывать платформенно независимые модели и уже сейчас создавать обширные репозитории моделей.

Язык OWL

OWL (Web Ontology Language) – язык для представления онтологий и связанной информации в виде семантической сети. OWL был создан WWW консорциумом в 2004-м году как открытый стандарт. OWL развился из языка разметки агентов DARPA и OIL – слоя предположения онтологий Европейского Сообщества. OWL дополняет схему среды описания ресурсов (RDFS – Resource Description Framework Schema) онтологическими конструкциями для описания объектно-ориентированных классов, свойств и экземпляров [5].

Для замены конструкций используется RDF/XML синтаксис. Существует несколько вариантов языка, варьирующихся в зависимости от полноты, выразительности, простоты и скорости обработки информации. Полная спецификация языка называется OWL Full, более простые – OWL DL и OWL Lite.

Онтологии OWL состоят из сущностей (концептов) и отношений. Концепты предметной области определяются в виде OWL классов, отношения определяются как свойства [6]. Так же для обеспечения описания необходимых и/или достаточных отношений классов используются дополнительные конструкции OWL (т.н. аксиомы). В их число входят: subClassOf, equivalentClass, unionOf, disjointWith.

В дополнение к концептам, определяемым в виде классов, OWL включает свойства, которые описывают взаимоотношения между концептами. Отношения можно рассматривать как направленную связь между вершинами – концептами. Стартовая вершина называется элементом-доменом, конечная – элементом-значением. Например, некоторый объект (домен) может иметь некоторую особенность (значение) через свойство hasFeature.

В OWL определены два типа свойств: свойства-объекты и свойства-типы данных. Для свойств-объектов значением является экземпляр класса, для свойств-типов данных значение – это примитивный тип данных, такой как байт, дата или число.

Так же OWL имеет широкие возможности по определению собственных типов свойств и отношений.

На основе OWL уже сейчас существуют несколько решений, используемых в системах имитационного моделирования.

В [7] описано использование онтологий портов для автоматического построения моделей:

Порты описывают интерфейс, определяющий границы компонентов или подсистем в конфигурации системы. Система представляется как конфигурация подсистем или компонентов, соединенных друг с другом через четко определенные интерфейсы.

Конфигурация интерфейса компонента состоит из портов, которые определяют предполагаемое взаимодействие объекта с окружающей

средой. Взаимодействия могут проходить в виде обмена энергией, материей или сигналов (т.е. информации). При построении модели взаимодействие двух подсистем друг с другом обозначается наличием связи между соответствующими портами подсистем.

Такая концепция сходна с концепцией описания слоев структуры и сообщений языка моделирования Triad, за исключением меньшей гибкости описания сложных взаимодействий. В Triad существует возможность описывать сложные типы сообщений для создания модели со сложными взаимодействиями, в данной же работе такие взаимодействия описываются комбинациями из небольшого множества основных типов портов.

В [8] представлено модульное иерархическое представление для дискретно-событийного моделирования, базирующееся на портах. Таким образом, построение онтологии портов позволит автоматизировать определение того, какие порты могут быть соединены друг с другом не только по принципу одинаковых типов передаваемых сигналов (энергии, материи), но и на основании семантического описания порта.

Supply Chain Operations Reference model (SCORmodel) [10] был разработан 1996 году. SCOR-model – справочная модель бизнес процессов, которая позволяет описывать широкий диапазон цепей поставок и может описать любую цепь поставок на любом уровне детализации. Сейчас доступна уже седьмая версия SCOR-model.

При создании онтологии сетей поставок в качестве ядра использовалась сама SCOR-model, как единственный на тот момент времени, широко используемый источник знаний о сетях поставок. Следующий уровень онтологии, называемый средней онтологией, был построен поверх ядра для того, чтобы включить в него все точки зрения на цепи поставок. Он формально определяет все существующие на этот момент концепты, используемые при их описании. Для поддержки использования этих онтологий при моделировании конкретных цепей конкретными пользователями был создан еще один слой онтологий – динамическая онтология. Она используется для автоматизированного определения конкретной цепи поставок, партнеров, в ней участвующих, и их характеристик из ядра и средней онтологии. Все три уровня онтологии были объединены в одну, для упрощения работы с ней.

Стандарты IDEF

Для поддержания процесса построения онтологий в IDEF5 существуют специальные онтологические языки: схематический язык (Schematic Language-SL) и язык доработок и уточнений (Elaboration Language-EL). SL является наглядным графическим языком, специально предназначенным для изложения компетентными специалистами в рассматриваемой области системы основных данных в форме онтологической информации. Этот несложный язык позволяет естественным образом представлять основную информацию в начальном развитии онтологии и дополнять существующие онтологии новыми данными. Язык EL представляет собой структурированный текстовой

язык, который позволяет детально характеризовать элементы онтологии. Язык SL позволяет строить разнообразные типы диаграмм и схем в IDEF5.

Основная цель всех этих диаграмм – наглядно и визуально представлять основную онтологическую информацию.

Несмотря на кажущееся сходство, семантика и обозначения схематичного языка SL существенно отличается от семантики и обозначений других графических языков. Дело в том, что часть элементов графической схемы SL может быть изменен или вовсе не приниматься во внимание языком EL. Причина этого состоит в том, что основной целью применения SL является создание лишь вспомогательной структурированной конструкции онтологии, и графические элементы SL не несут достаточной информации для полного представления и анализа системы, тем самым они не предназначены для сохранения при конечном этапе проекта. Тщательный анализ, обеспечение полноты представления структуры данных, полученных в результате онтологического исследования, являются задачей применения языка EL.

Стандарт IDEF5 для построения онтологий в системах имитационного моделирования используется в проекте MODELISM.

Система MODELISM разделяется на три уровня. ***Уровень предметной области*** описывает онтологию моделируемой системы с помощью моделей онтологии IDEF5 и моделей процессов IDEF3. ***Уровень проектирования*** автоматизирует построение модели на основе знаний уровня предметной области, а так же знаний о моделировании в целом (о порядке анализа входов и выходов моделирования, оптимизации, управляемой моделированием и т.п.). ***Уровень исполнения и анализа*** производит анализ входных данных модели, имитационный прогон, анализ выходов и возможностей для оптимизации.

3.5.3. Агентные системы распределенного моделирования

Агентное моделирование (АМ) – это новый подход к моделированию систем, содержащих автономных и взаимодействующих агентов. АМ может найти широкое применение в бизнесе для принятия решений. АМ обещает дать хорошие результаты при его использовании исследователями электронных лабораторий с целью поддержки их разработок.

В настоящее время мультиагентные системы получили широкое применение в таких областях как системы телекоммуникации, поисковые системы в Internet, логистика, компьютерные игры, САПР, системы управления и контроля сложными процессами в медицине и промышленности, программы для электронной коммерции, системы защиты информации. Это и моделирование поведения агентов на фондовых рынках, и моделирование поставок, и предсказание распространения эпидемий, и угрозы биологических войн и т.д. [3, 4, 5].

Выясним более подробно, что такое агенты, предложим их классификацию, рассмотрим архитектуру мультиагентных систем, сферы

применения мультиагентных систем, различие между дискретным и агентным моделированием.

Мультиагентные системы и агенты

Мультиагентные системы (кросс-платформенные распределённые интеллектуальные системы) представляют собой совокупность интеллектуальных ***агентов***.

Агенты – это автономные объекты, которые могут самостоятельно реагировать на внешние события и выбирать соответствующие действия. В настоящее время в рамках мультиагентных технологий и мультиагентных систем (МАС) разработаны различные типы агентов, которые характеризуются конкретной моделью поведения и свойствами, а также, семейства архитектур и библиотек компонентов, для которых свойственны распределённость и автономность.

Поскольку агенты применяются в самых различных областях, то понятие агента для каждого автора имеет свою смысловую нагрузку: так на производстве в качестве агента может выступать робот, а в системах телекоммуникации понятие «агент» связано с программой.

В зависимости от среды обитания агента наделяют конкретным набором свойств.

Существует большое количество типов агентов: автономные агенты, мобильные агенты, персональные ассистенты, интеллектуальные агенты, социальные агенты и т.д. По способу поведения агенты могут делиться на классы: интеллектуальные (рассуждающие, коммуникативные, ресурсные) и реактивные (не обладающие представлением о внешнем мире).

Существует множество определений агента, данные разными авторами – исследователями МАС.

Одним из авторов (Wooldridge M.) [6] дано следующее определение: агент – это аппаратная или программная сущность, способная действовать в интересах достижения целей, поставленных пользователем.

Для реализации некоторого поведения агент должен иметь специальные устройства (в том числе и программные): ***рецепторы*** – устройства, непосредственно воспринимающие воздействие внешней среды и ***эффекторы*** – исполнительные органы, воздействующие на среду, а также процессор – блок переработки информации и память.

Под памятью здесь понимается способность агента хранить информацию о своем состоянии и состоянии среды.

Перечислим ***свойства агента***:

- адаптивность (способностью обучаться);
- автономность (агент работает как самостоятельная программа, которая ставит себе цели и предпринимает действия для их достижения);
- коллаборативность (взаимодействие с другими агентами, причём агент может играть разные роли при взаимодействии с одним и тем же агентом);

• способность к рассуждениям (агенты могут обладать частичными знаниями или механизмами вывода, например, знаниями, как приводить данные из различных источников к одному виду);

• коммуникативность (агенты могут общаться с другими агентами);

• мобильность (способность к передачи кода агента с одного сервера на другой).

Итак, агент должен взаимодействовать со ***средой*** или с ***другими агентами***. Агенты ***децентрализованы*** [4], а ***глобальное состояние*** моделируемой системы можно вывести из взаимодействия агентов, зная индивидуальную логику поведения каждого из них.

Программной реализацией агента является объект (в смысле объектно-ориентированного программирования) некоторого класса. Однако существует два основных отличия агента от простого объекта. Во-первых, обмен сообщениями между объектами сводится в основном к вызову методов, в то время как для агентов это наиболее важный ***процесс взаимодействия***, поэтому они способны к ведению сложных переговоров на основе теории речевых актов для выработки общего решения. Во-вторых, агенты должны обладать интенциональными (т.е. психическими) характеристиками, сближающими их с живыми существами.

Итак, про агента можно сказать следующее.

1. Агент ***идентифицируем***, его можно определить как индивидуум, состоящий из отдельных частей с определённым набором характеристик и правил, управляющих его поведением и отвечающих за принятие решений. Агенты состоят из модулей. Требование ***модульности*** означает, что агент имеет чётко видимый контур и любой может определить является ли что-либо частью агента, не является его частью, или является общей характеристикой нескольких агентов.

2. Агент помещён в определённую ***среду***, живёт в ней и взаимодействует с другими агентами. Агенты имеют ***протоколы для взаимодействия*** с другими агентами типа коммуникационного протокола, и способны реагировать на среду. Агенты способны распознавать и различать различные черты других агентов.

3. Агент ***целеустремлён***, имеет ***цели***, которые необходимо достичь, исходя из его поведения.

4. Агент ***автономный*** и ***самонаводящийся***. Агент может функционировать в своей среде независимо от сделок с другими агентами, по крайней мере, в небольшом числе ситуаций.

5. Агент ***гибкий*** и может ***обучаться*** и ***адаптировать*** своё поведение по времени, основываясь на опыте. Агент может иметь ***метаправила***, правила, изменяющие другие правила поведения.

Агент, его характеристики и реагирование агента на среду, в которой он живет и взаимодействует с другими агентами, графически представлено на рис. 3.8.

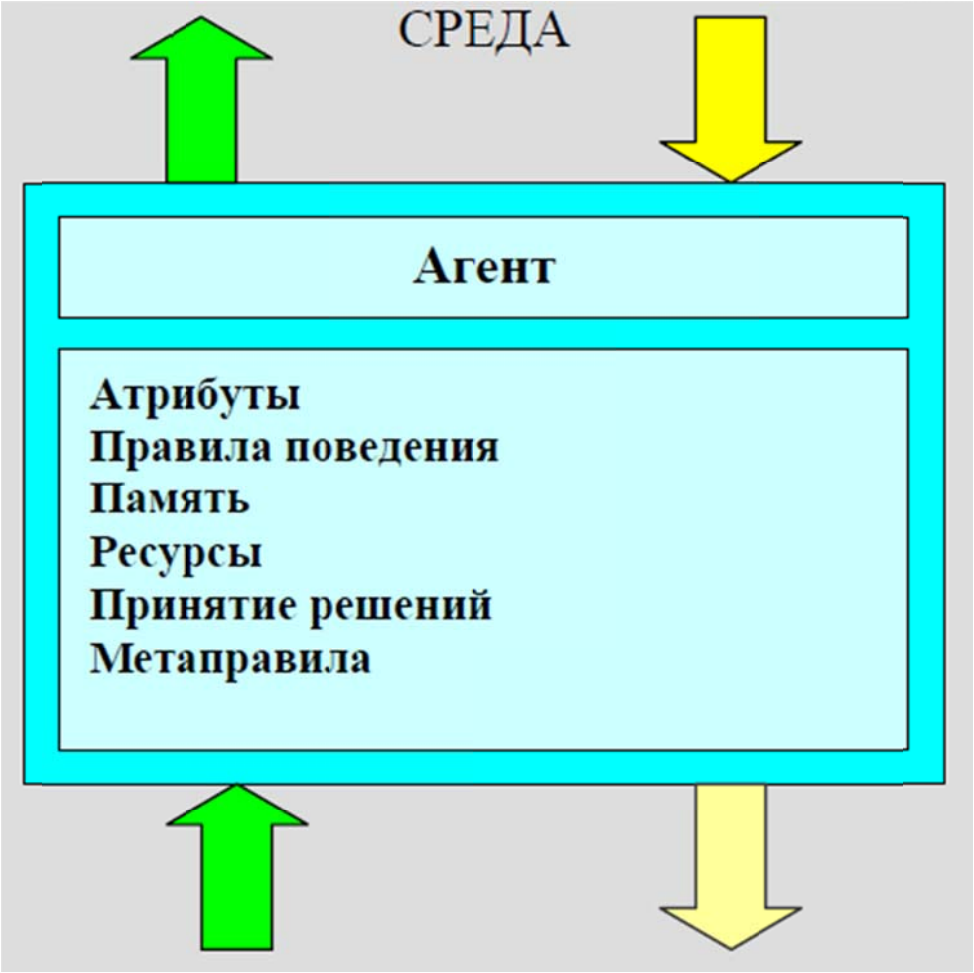


Рис. 3.8. Агент и его характеристики

Агенты ***разнообразны***, ***разнородны*** и ***динамичны***. Агенты активны по отношению к своим атрибутам и правилам поведения. ***Правила поведения*** варьируются: они зависят от того, какое количество информации агент учитывает в своих рассуждениях, от внутренней модели внешнего мира, включающей других агентов, от размера памяти для хранения прошлых событий. Все перечисленные факторы агент использует в своих решениях.

Агенты также различаются по своим ***атрибутам*** и накопленным ***ресурсам***. Многообразная природа агентов делает агентное моделирование особенно интересным.

Существуют разные термины для обозначения агентного моделирования [5]. Это и ***индивидуумное моделирование***, и ***системы агентного моделирования*** (САМ).

Агенты АМ отличны от типичных агентов мобильных систем. Мобильные агенты – легковесные программные агенты-прокси, которые выполняют различные функции для пользователей и в некоторой мере действуют автономно. АМ – это не то же самое объектно-ориентированное моделирование, хотя объектно-ориентированная парадигма широко используется для представления агентов. По этой причине

инструментальные средства для реализации агентного моделирования почти всегда объектно-ориентированные.

АМ уходит корнями к мультиагентным системам (МАС) и робототехнике в сфере искусственного интеллекта (ИИ). Но АМ не ограничивается проектированием искусственных агентов. Его основные задачи состоят в моделировании социального поведения людей и принятии решений индивидуумами. Вместе с этим приходится представлять взаимодействие в обществе, сотрудничество индивидуумов, их групповое поведение и появление высокоорганизованных социальных структур.

Причины развития систем агентного моделирования

Агентное моделирование находит всё большее и большее применение. Это объясняется тем, что мы живем во все более и более сложном мире. Прежде всего, системы, которые мы должны анализировать и моделировать, становятся всё более сложными из-за сложных взаимозависимостей их компонентов. Таким образом, традиционные подходы к моделированию (системная динамика, дискретное моделирование) не могут удовлетворить исследователей.

Рассмотрим пример в прикладной области – это моделирование экономических рынков, которое традиционно полагалось на представление об совершенных рынках, однородных посредниках, и долговременном равновесии, потому что эти предположения позволяли решить эти задачи аналитически. Если же отказаться от этих предположений, то используемый ранее подход к исследованию экономических рынков становится неприемлемым.

Кроме того, вычислительные мощности быстро увеличиваются. Теперь можно просчитывать крупномасштабные микромодели, а это выглядело бы неправдоподобным ещё пару лет назад.

Вышесказанное позволяет сделать заключение о том, что прежние подходы к моделированию не всегда являются адекватными и следует искать ***новые подходы***. Таким подходом является агентное моделирование.

Связь агентного моделирования с другими науками

Агентное моделирование имеет связи со многими сферами деятельности, включая:

- системную динамику;
- компьютерные науки;
- науки об управлении;
- социальные науки;
- традиционное моделирование и т.д.

АМ заимствует из этих областей теоретические основы, их концептуальный взгляд на мир и философию, и использует это для применимых техник моделирования. АМ уходит своими историческими корнями к сложным адаптивным системам (САС) и к лежащему в их основе представлению о системах, построенных снизу вверх. Это

представление является противоположным к взгляду на систему сверху вниз, как это принято в системной динамике.

Развитие САС первоначально было связано с исследованиями в области биологических систем. САС способна самоорганизоваться и динамически реорганизовать свои компоненты выживания в некоторой среде. Джон Холланд (John Holland), первооткрыватель в этой области, обнаружил свойства и механизмы, общие для всех САС [9].

Свойства САС:

- агрегирование: позволяет группам организовываться;
- нелинейность: сводит на нет простую экстраполяцию;
- течение: позволяет перемещать и преобразовывать ресурсы и информацию;
- многообразие: позволяет агентам вести себя по-разному, что часто приводит к повышению устойчивости (надежности) системы.

Механизмы САС следующие:

- пометка: позволяет агентам иметь имена и быть различимыми;
- внутренняя модель: позволяет агентам рассуждать об их внутренних мирах;
- строительные блоки: позволяет компонентам и системе в целом быть сформированной из большого числа уровней более простых компонентов.

Эти свойства САС и механизмы предоставляют полезную инфраструктуру для проектирования агентных моделей. Должно быть упомянуто, что Холланд (Holland) также разработал генетические алгоритмы [9] при исследовании САС. Генетические алгоритмы – это общая процедура поиска, основывающаяся на генетических механизмах и естественной селекции, и является одной из основ для оптимизационных алгоритмов.

Boids моделирование

Boids моделирование – это хороший пример того, как простые правила приводят к возникновению организованных систем, чье поведение напоминает плавание косяков рыб или поведение стай птиц.

В Boids модели каждый агент руководствуется тремя правилами, управляющими его движением:

- ***связность***: каждый агент придерживается средней позиции относительно его товарищей по стае;
- ***разделение***: каждый агент поступает так, чтобы избежать скопления товарищей по стае;
- ***выравнивание***: каждый агент следует среднему курсу его локальных товарищей по стае.

Взаимодействие агента с расположенными вблизи или локальной окрестности агентами определено некоторым расстоянием (пространственной мерой). Даже только эти три простых правила, примененные на внутреннем уровне агента и только к агентам в его

окрестности, приводят к тому, что поведение агентов начинает выглядеть скоординированным (рис. 3.9).

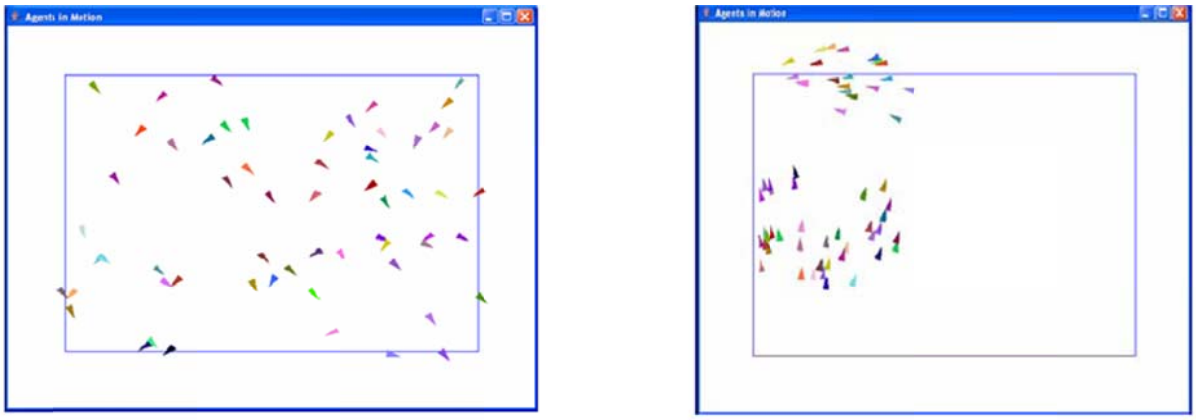


Рис. 3.9. Boids моделирование (начальная конфигурация и колония после 500 обновлений)

Сферы применения агентного моделирования и примеры агентных моделей

Агентные модели и приложения находят широкое применение в разных областях науки, производства и т.д.

Ниже представлены ***сферы применения*** агентного моделирования:

- рынок и конкуренция;
- динамика населения;
- управление проектами;
- экосистемы;
- динамика персонала;
- экономика здравоохранения;
- цепочки поставок;
- переработка отходов;
- перевозки;
- транспорт: макро-модели;
- энергетические сети;
- сервисные центры;
- отделение скорой помощи;
- транспорт: микро-модели;
- компьютерные системы;
- движение пешеходов;
- склад и логистика;
- производство;
- системы управления.

Далее рассмотрим ***примеры агентных моделей*** [5].

EMCAS – это сложная адаптивная система рынка электроэнергии. Она представляет собой агентную модель рынка электроэнергии,

спроектированную для исследования реструктуризации конкурентного рынка цен на электричество. Агенты в EMCAS представляют участников рынка электроэнергии. Различные типы агентов отражают разнообразие взаимодействующих компонентов. Они включают генерирующие компании, компании-потребители, передающие компании, распределяющие компании, независимые системные операторы, конечные потребители и регулировщики. Эти агенты выполняют различные задачи, используя специализированные правила принятия решений. Агенты получают знания о рынках как результат ответной реакции на их предложения (товара-цены), и, зная стратегии поведения конкурентов, адаптируют свои действия.

Агенты пытаются использовать новые стратегии в ответ на динамически изменяющиеся поставки и внешние требования, определяют лучшие стратегии. Агенты генерирующих компаний ведут ценовую разведку. Они пытаются влиять на рынок, при этом происходит их обучение. Агенты EMCAS взаимодействуют при условии физических ограничений на индивидуальные пропускные мощности для конкретной региональной сети передачи электроэнергии.

Агентная модель цепочек поставок

Агентная модель цепочек поставок демонстрирует АМ подход. За основу была выбрана «Bear model» (эта модель первоначально была использована в системной динамике). Цепочки поставок состоят из четырех частей: производство, дистрибьюторы, оптовые торговцы и розничные торговцы, которые отвечают на запросы конечных потребителей.

Для упрощения модели были предложены следующие условия:

- существует только один продукт;
- не происходят превращения товаров;
- также недопустима и не требуется сборка различных материалов в конечный продукт.

Потоки товаров и информации отображены в форме заявок между различными этапами (агентами), в модель включены переводы (денег), но отсутствуют потоки оплаты, а также переговоры и финансовые расчеты. Как бы то ни было, эти аспекты поведения для агентов цепочек поставок легко могут быть впоследствии включены в агентную версию модели.

Агенты цепочек поставок состоят из потребителя, розничного торговца, оптового торговца, дистрибьютора и производителя (рис. 3.10). Каждый период агенты цепочек поставок выполняют действия, руководствуясь правилами своего поведения:

- потребитель подает заявку розничному торговцу;
- розничный торговец удовлетворяет заявку немедленно из соответствующих запасов, если имеет достаточно запасов на складе (если запасы закончились, то заявка потребителя помещается в невыполненные заявки и выполняется при пополнении склада);

• розничный торговец получает очередную партию груза от оптового торговца в ответ на предыдущие заявки, решает, сколько груза заказать оптовому торговцу, основываясь на «правиле заявок»;

• розничный продавец оценивает будущие заказы потребителей, используя правила «прогноза требований (заявок)», затем розничный торговец заказывает товар у оптового торговца, чтобы покрыть ожидаемые требования и любой дефицит, относительно явно задаваемых запасов и целей;

• аналогично, каждый оптовый продавец получает товар от производителя, и подаёт заявки дистрибьюторам, основываясь на заявках розничных торговцев.

Этот процесс продолжается дальше вверх по цепочке до производителя, который решает, сколько произвести продукции.

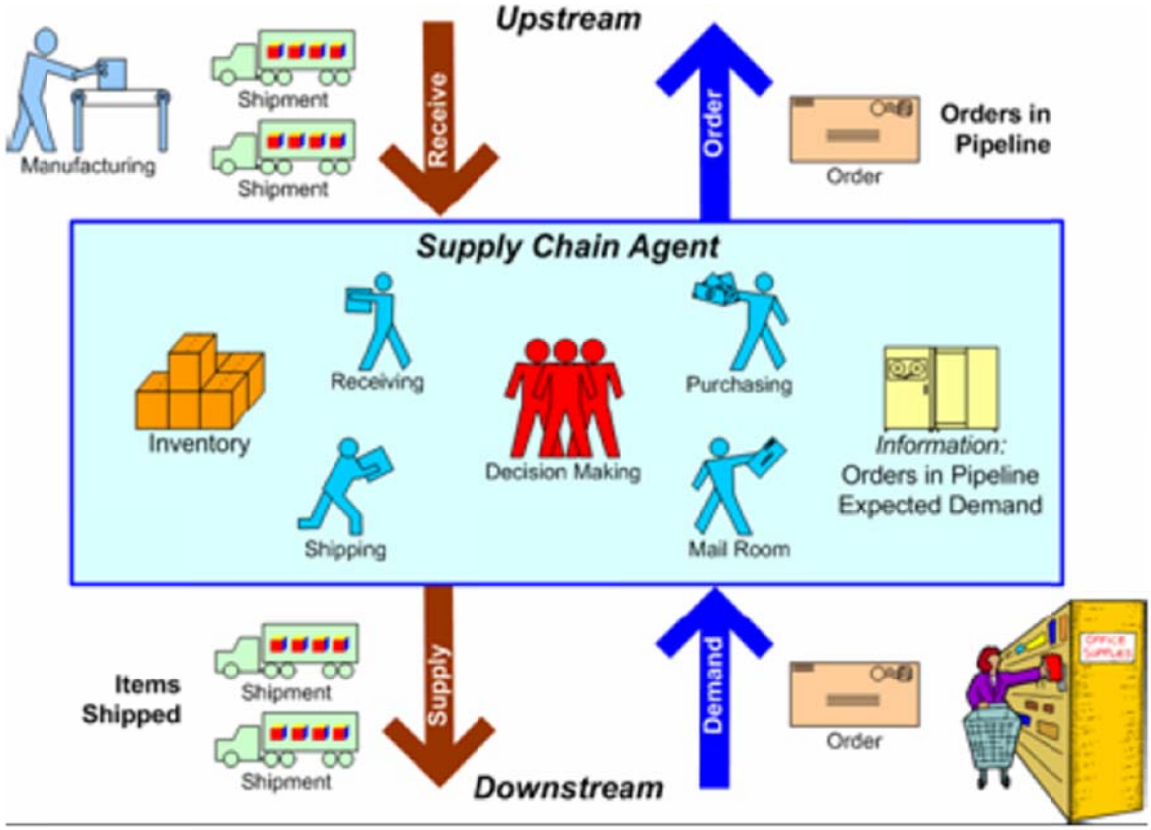


Рис. 3.10. Цепочка поставок

Целью агентов в этом моделировании является управление их собственными запасами таким образом, чтобы минимизировать их стоимость с помощью разумных решений, основывающихся на том, сколько товара заказать в каждый период. Когда запасы практически пусты и существует угроза их опустошения, агенты заказывают больше; когда запасов слишком много и агенты несут большие затраты на хранение этих запасов, и потому заказывают меньше товаров. Каждый агент несет расходы на содержание хранение товара на складе. Агенты также несут отложенные расходы, когда они получают заявку (заказ) и не могут его

сразу же выполнить по причине отсутствия запасов на складе. Каждый агент делает выбор между слишком большими запасами на складе, которые увеличивают расходы на хранение, и слишком маленькими запасами на складе, что увеличивает риск того, что закончатся товары и ему придется нести отложенные расходы.

В этом примере, агенты цепочек поставок имеют доступ только к локальной информации. Никто из агентов не видит всей цепочки поставок или не может оптимизировать всю систему в целом. Адаптивные правила принятия решений агентов используют только локальную информацию для принятия решений.

Результаты агентной модели цепочек поставок полностью повторяют результаты оригинальной модели и тем доказывают её состоятельность. Эта простая агентная модель является полезной основой для более реалистичных моделей цепочек поставок, такие как модели, основанные на топологии сетей поставок или на альтернативных правилах принятия решений агентами.

Инструментальные средства агентного моделирования

Агентное моделирование может быть реализовано на небольших, настольных компьютерах, или с использованием крупных кластеров компьютеров, или на любом другом варианте между первыми двумя.

Настольные агентные модели могут быть простыми. Обычно они разрабатываются программистом с помощью утилит, который может изучить их за несколько недель.

Настольные системы АМ могут быть использованы для обучения тому, как моделировать с использованием агентов, протестировать концепции разработки агентных моделей и проанализировать результаты. Настольные утилиты включают в себя электронные таблицы и математические вычислительные системы.

Одно из ключевых преимуществ настольных систем АМ состоит в том, что они являются интерпретируемыми средами, не требующими шагов компиляции и компоновки, что необходимо выполнить при использовании традиционных языков программирования.

Для построения агентных моделей используют электронные таблицы (в частности, Excel) и математические вычислительные системы (МВС) такие, как Mathematica и MATLAB. Применение настольные АМ систем ограничено максимальным количеством управляемых агентов, которое обычно находится в диапазоне от нескольких десятков до нескольких сотен.

Крупномасштабное агентное моделирование расширяет возможности простых агентных настольных моделей и позволяет большему числу агентов (от тысяч до миллионов) участвовать в сложных взаимодействиях.

Крупномасштабное агентное моделирование обычно выполняется с использованием специальных сред моделирования. Эти среды включают:

- планировщика (по времени);

- механизмы коммуникации;
- гибкие топологии взаимодействий агентов;
- широкий выбор устройств для хранения и отображения состояния агентов.

Крупномасштабные агентные модели могут быть запущены на настольных компьютерах в дополнение к высокопроизводительным вычислительным системам. Как бы то ни было, крупномасштабные агентные модели в основном требуют больших навыков и больших ресурсов на разработку по сравнению с настольными средами.

Благодаря значительным открытым разработкам и инвестициям в разработку, многие из сред для АМ доступны широкому пользователю. Это системы агентного моделирования Repast, Swarm, NetLogo и Any Logic.

Repast

The REcursive Porous Agent Simulation Toolkit (Repast) – это ведущий открытый и свободный источник библиотек для крупномасштабного агентного моделирования.

Repast поддерживает разработку чрезвычайно гибких моделей агентов и используется в моделировании социальных процессов. Пользователь строит свою модель, включая в свои программы компоненты из библиотеки Repast или используя визуальный Repast для среды Python Scripting.

Существует **три версии Repast**, названных **Repast for Python (Repast Py)**, **Repast for Java (Repast J)** и **Repast for the Microsoft.NET framework (Repast.NET)**.

Repast Py – это кросс-платформенная визуальная система для моделирования, которая позволяет пользователям строить модели, используя графический интерфейс, и описывать поведение агентов (скрипты Python). Все из возможностей системы Repast доступны и в Repast Py, но система Repast Py спроектирована для быстрой разработки прототипа агентных моделей. Модели Repast Py могут быть автоматически экспортированы в крупномасштабную среду разработки для Repast Py.

Repast J – это среда моделирования, написанная исключительно на Java. Она предназначена для разработки крупномасштабных агентных моделей и включает:

- параллельный дискретный планировщик по времени (календарь событий);
- среду для визуализации модели, средства интеграции с географическими информационными системами с целью моделирования агентов на реальных картах);
- адаптивные средства поведения, такие как нейронные сети и генетические алгоритмы.

Интерфейс Repast J представлен на рис. 3.11. Линии на нижнем экране отображают взаимодействие между соединенными агентами.

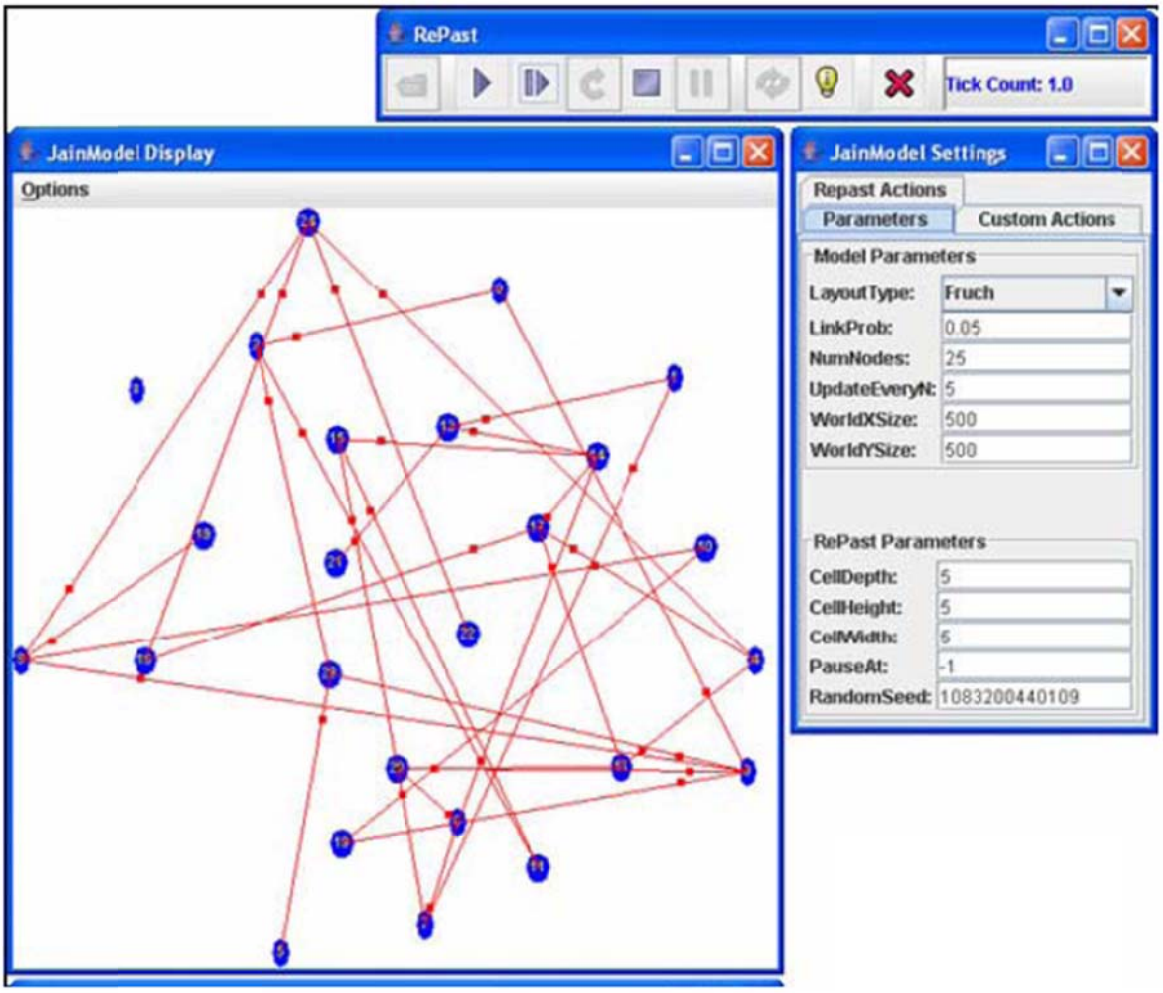


Рис. 3.11. Интерфейс Repast J

Repast .NET – это среда моделирования, написанная исключительно на C#, и она переносит все возможности Repast J на платформу Microsoft.NET. Repast .NET модели могут написаны на любом языке поддерживаемом платформой Microsoft.NET, таком как Managed C++, C#, Visual Basic или даже Managed Lisp Managed или Prolog10.1.

Repast имеет сложный встроенный планировщик, который поддерживает дискретно-событийное моделирование. Repast позволяет использовать большой набор коммуникационных механизмов с разнообразными топологиями взаимодействия, включает полный набор утилит для хранения и отображения состояния агентов. В систему также включены утилиты для автоматической интеграции как с коммерческими так и свободно доступными географическими информационными системами (ГИС). Интеграция с коммерческими ГИС включает в себя автоматическое подключение к широко используемым географическим информационным системам, таким как ESRI и ArcGIS.

Более того, так как Repast базируется на языке Java, платформе Microsoft.NET и на скриптах Python, он полностью объектно-ориентирован.

SWARM

Система моделирования Swarm (стая, рой) была первой средой разработки АМ приложений, впервые запущен в 1994 году Chris Langton at the Santa Fe Institute. Swarm – это открытый и бесплатный набор библиотек с открытым кодом и в настоящее время поддерживается Swarm Development Group (SDG). Swarm стремиться создать распределенную платформу для моделирования АМ и содействовать разработке широкого круга моделей.

Пользователь создает модели включением компонентов из библиотек Swarm в свои программы. Больше информации про Swarm, также как и загрузки, могут быть найдены на домашней странице SDG.

Система моделирования Swarm состоит из двух основных компонентов.

Компоненты ядра запускают код моделирования, написанный на языке общего назначения Objective-C, Tcl/Tk, и Java. В отличие от Repast, Swarm-планировщик поддерживает только продвижение времени через фиксированные промежутки. Swarm поддерживает полный набор коммуникационных механизмов и может моделировать все основные топологии. Swarm включает хороший набор утилит для хранения и отображения состояния агентов. Так как Swarm базируется на комбинации Java и Objective-C, то он объектно-ориентирован. Но эта смесь языков является причиной некоторых трудностей с интеграцией в некоторые крупномасштабные среды разработки, например Eclipse. Swarm поддерживает ГИС через библиотеку Kenge.

NetLogo

NetLogo – это другая кросс платформенная мультиагентная среда для моделирования, которая широко используется и поддерживается. Первоначально основанная на системе StarLogo, NetLogo приспосабливает агентные системы, состоящие из комбинации живых и программных агентов-участников.

AnyLogic

Система моделирования AnyLogic – это отечественная разработка. В настоящее время она нашла широкое применение среди отечественных и зарубежных пользователей.

AnyLogic – единственный инструмент имитационного моделирования, который поддерживает все подходы к созданию имитационных моделей: процессно-ориентированный (дискретно-событийный), системно динамический и агентный, а также любую их комбинацию, т.е. комбинированный подход (рис. 3.12). Уникальность, гибкость и мощность языка моделирования, предоставляемого AnyLogic, позволяет учесть любой аспект моделируемой системы с любым уровнем детализации. Графический интерфейс AnyLogic, инструменты и библиотеки позволяют быстро создавать модели для широко спектра задач от моделирования производства, логистики, бизнес-процессов до стратегических моделей развития компании и рынков.

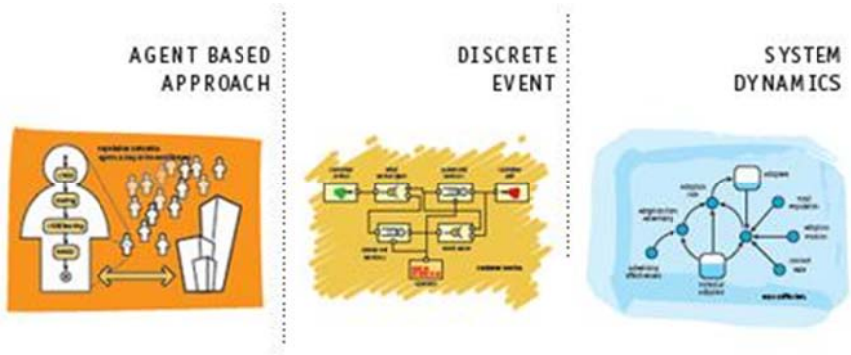


Рис. 3.12. Комбинированный подход к созданию моделей в AnyLogic

Дискретно-событийное моделирование. Термин «дискретно-событийное моделирование» обычно используется для обозначения «процессного» моделирования, в котором динамика системы представляется как последовательность операций (прибытие, задержка, захват ресурса, разделение, ...) над некими сущностями (entities, по-русски – транзакты, заявки), представляющими клиентов, документы, звонки, пакеты данных, транспортные средства и т.п. (рис. 3.13). Эти сущности пассивны, они сами не контролируют свою динамику, но могут обладать определёнными атрибутами, влияющими на процесс их обработки (например, тип звонка, сложность работы) или накапливающими статистику (общее время ожидания, стоимость). Процессное моделирование – это средне-низкий уровень абстракции: здесь каждый объект моделируется индивидуально, как отдельная сущность, но множество деталей «физического уровня» (геометрия, ускорения/замедления) обычно опускается.

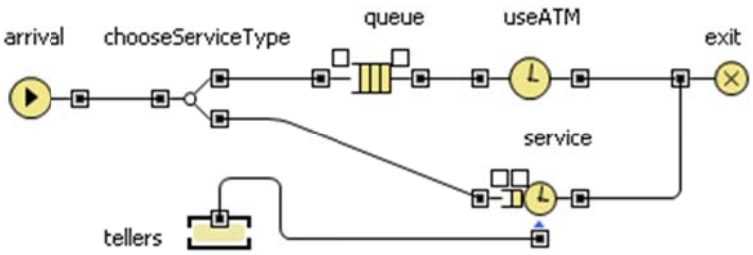


Рис. 3.13. Дискретно-событийное моделирование в AnyLogic

AnyLogic включает специальную библиотеку (Enterprise Library) для моделирования дискретно-событийных (процессных) систем. Библиотека позволяет создавать любые дискретно-событийные модели из блоков. Для каждого блока библиотеки задана анимация по умолчанию. Кроме того пользователь может создать свою анимацию любой сложности, которая будет отображать моделируемые процессы и позволять управлять параметрами модели в процессе ее выполнения.

В Enterprise Library также входят объекты, разработанные для моделирования процессов, происходящих в пространстве (и зависящих от) пространства: таких, где объекты-заявки и ресурсы перемещаются в некой сети. Это подмножество объектов значительно упрощает моделирование некоторых типов систем, например, производства, внутризаводской логистики, супермаркета, склада, госпиталя.

Для использования этого подхода, называемого ***сетевым моделированием*** (Network Based Modeling), необходимо определить топологию сети (например, используя векторную графику AnyLogic поверх плана или чертежа здания или сооружения), множества ресурсов (статических, движущихся или перемещаемых), и собственно процесс (рис. 3.14). Процесс в данном случае – это комбинация объектов типа «переместиться туда-то» или «присоединить к себе ресурс» и обычных объектов Enterprise Library. Заявки и ресурсы автоматически анимируются движущимися по сегментам сети или находящимися в её узлах; эта анимация может также комбинироваться с обычной.

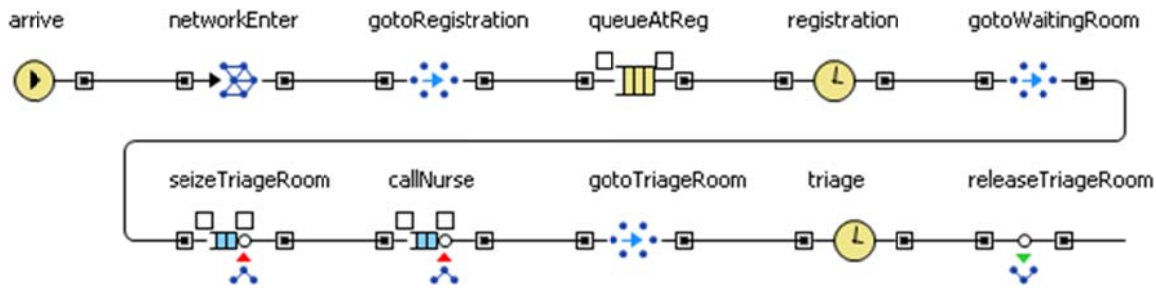


Рис. 3.14. Сетевое моделирование в AnyLogic

Системно-динамическое моделирование. Системно-динамическое моделирование (системная динамика) – это способ имитационного моделирования, своими методами и инструментами позволяющий понять структуру и динамику сложных систем. Также системная динамика – это метод моделирования, использующийся для создания точных компьютерных моделей сложных систем для дальнейшего использования с целью проектирования более эффективной организации и политики взаимоотношений с данной системой. Вместе, эти инструменты позволяют нам создавать микромиры-симуляторы, где пространство и время могут быть сжаты и замедлены так, чтобы мы могли изучить последствия наших решений, быстро освоить методы и понять структуру сложных систем, спроектировать тактики и стратегии для большего успеха.

Системная динамика главным образом используется в долгосрочных, стратегических моделях и принимает высокий уровень абстракции. Люди, продукты, события и другие дискретные элементы представлены в моделях системной динамики не как отдельные элементы, а как система в целом (рис. 3.15).

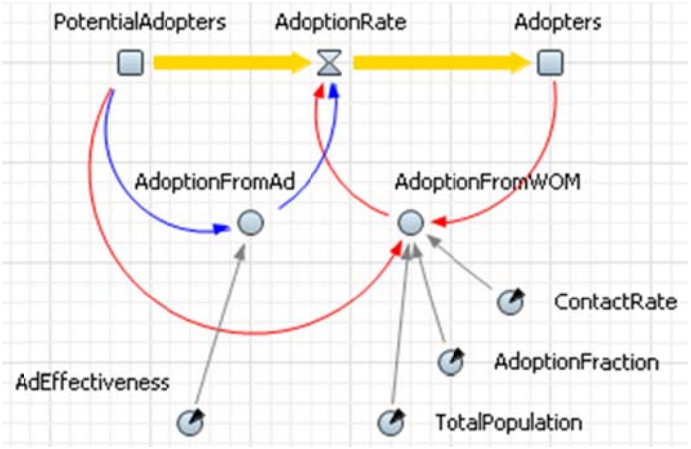


Рис. 3.15. Системно-динамическое моделирование в AnyLogic

AnyLogic не только полностью поддерживает потоковые диаграммы, как и любой инструмент для системной динамики, но и предоставляет значительно больше возможностей.

AnyLogic позволяет создавать интерактивную анимацию моделей, которая будет значительно более понятна пользователям, чем потоковые диаграммы, понятные только разработчикам модели.

Пользователь может комбинировать свою модель с другими. Например, для моделирования рынка можно использовать системную динамику, а производство и цепочку поставок моделировать, используя процессное моделирование и агентное. так как эти методы лучше подходят для данных задач. Пользователь может импортировать свои модели из VenSim.

Кроме того, AnyLogic создан на основе открытой, инновационной Java платформы Eclipse, которая стала стандартом де-факто для разработки бизнес приложений. Благодаря Eclipse AnyLogic работает на всех платформах – Windows, MacOS, Linux, а среда разработки основана на современных технологиях и проста в использовании.

AnyLogic поддерживает разработку и моделирование систем обратной связи (диаграммы потоков и накопителей, правила решений, включая массивы переменных).

Итак, с помощью AnyLogic пользователь может:

- определять потоковые переменные одну за другой или использовать инструмент «flow tool»;
- использовать авто-заполнение при работе с формулами;
- создавать копии переменных для лучшей читаемости модели;
- использовать табличные функции со ступенчатой, линейной, сплайновой интерполяцией;
- определять поведение функции за пределами допустимой области;
- определять поддиапозоны и подразмерности;
- объявлять переменные-массивы с заданной размерностью;

• задавать различные уравнения для различных наборов элементов массива;

• использовать как специальные инструменты Системной динамики, так и возможности языка Java.

Одна из примечательных особенностей диаграммы потоков и накопителей это то, что стрелки зависимостей синхронизируются с формулами: стрелка зависимости от A до B появится автоматически, как только будет введена A в формулу переменной B, и исчезнет, если удалить A из формулы. Для стрелок потоков это правило работает наоборот: если удалить стрелку, то A будет исключена из формулы B. Значения переменных можно просмотреть непосредственно на диаграмме: щелкнув мышью на интересующем элементе во время и после прогона модели.

Являясь объектно-ориентированным инструментом, AnyLogic предоставляет все преимущества объектно-ориентированного подхода в системной динамике. Пользователь может создавать модели в иерархическом виде, где части диаграммы потоков и накопителей логически разделены. Кроме того, можно создать отдельный класс со своими функциями и методами и использовать его в пределах одной или нескольких моделей.

Модель, использующая метод системной динамики (как и любая другая модель в AnyLogic) может быть наглядной и интерактивной: пользователь может добавить диаграммы и произвольную графику, чтобы оживить модель; добавить «ползунки», кнопки, текстовые поля и т.д., чтобы управлять моделью во время выполнения (рис. 3.16). Набор поддерживаемых AnyLogic графических инструментов, богаче, чем в любом другом инструменте имитационного моделирования.

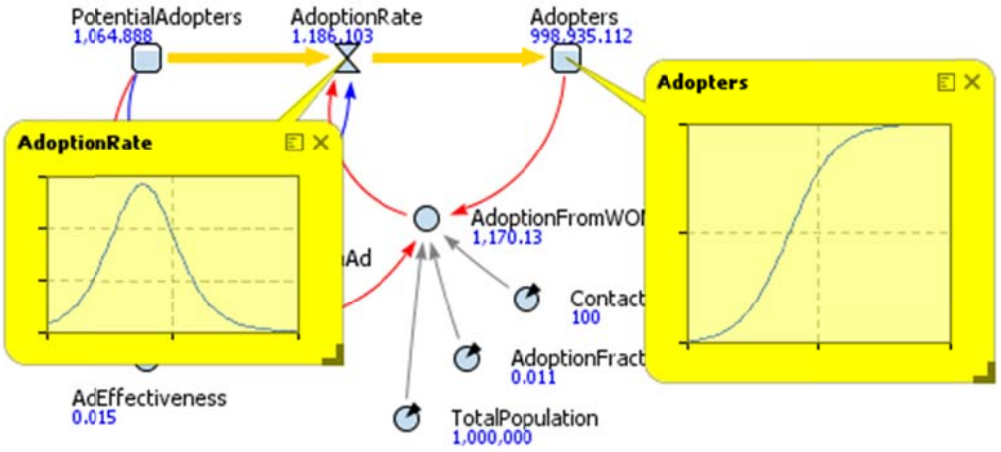


Рис. 3.16. Модель AnyLogic, использующая метод системной динамики

Поскольку модели в AnyLogic являются Java приложениями, они могут быть размещены в сети как апплеты. Отличие AnyLogic от других существующих решений в том, что модель полностью самостоятельна и не нуждается в установке на веб-сервер специального программного

обеспечения. Модель будет работать на машине клиента в пределах апплета. Для доступа пользователя может быть открыта не вся модель, а лишь некоторые ее части.

Агентное моделирование. AnyLogic поддерживает метод агентного моделирования (так же как системную динамику и дискретно-событийное моделирование) и позволяет эффективно комбинировать этот метод с другими известными подходами.

С точки зрения практического применения агентное моделирование (рис. 3.17) можно определить как метод имитационного моделирования, исследующий поведение децентрализованных агентов и то, как это поведение определяет поведение всей системы в целом.

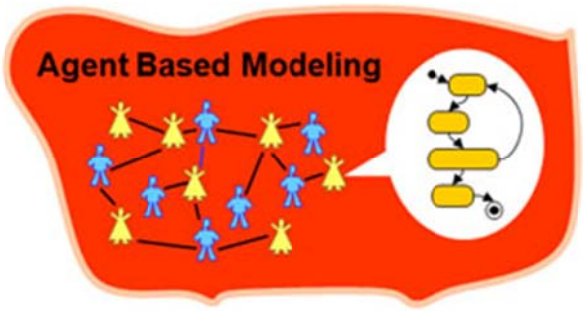


Рис. 3.17. Агентное моделирование в AnyLogic

Агентная модель – это ряд взаимодействующих активных объектов, которые отражают объекты и отношения в реальном мире. Таким образом, агентное моделирование делает шаг вперед в понимании и управлении совокупностью сложных социальных и бизнес процессов.

При разработке агентной модели в AnyLogic, пользователь вводит параметры агентов (это могут быть люди, компании, активы, проекты, транспортные средства, города, животные и т.д.), определяет их поведение, помещает их в некую окружающую среду, устанавливает возможные связи, после чего запускает моделирование. Индивидуальное поведение каждого агента образует глобальное поведение моделируемой системы.

AnyLogic предоставляет графический язык, который значительно упрощает создание агентных моделей:

- диаграммы состояний (statecharts) для задания поведения агентов;
- диаграмма действий для описания сложных алгоритмов;
- элемент «Среда» используется для описания «мира», в котором агенты «живут», и сбора различной статистики;
- элемент «Событие» используется для описания случайных или периодически происходящих событий.

Данные инструменты позволяют описать практически все поведенческие особенности агентов. Кроме того, всегда можно использовать Java, если нужно смоделировать какое-то специальное поведение или логику. Агентные модели могут комбинироваться с дискретно-событийными и системно-динамическими.

Существуют, однако, некоторые «шаблоны» моделирования агентных систем, которые упрощают создание модели и включены в AnyLogic:

- стандартная архитектура;
- агентная синхронизация («шаги»);
- состояние (непрерывное или дискретное);
- подвижность и анимация;
- агентные связи (сети, например социальные) и коммуникация;
- динамическое создание и уничтожение агентов.

Создание ***стандартной агентной модели*** в AnyLogic заключается в объявлении двух ***активных классов***. К примеру, объявляются класс Main для описания высокоуровневого объекта, где содержаться все агенты, и класс Person для описания агента нижнего уровня. Класс Person, в большинстве случаев, был бы объявлен как Agent – специальный класс подкласса ActiveObject, который расширил бы класс Person под цели агентого моделирования. Число агентов было бы включено в класс Main как дубликаты объекта «человек» класса Person. Одна или более конструкций типа Environment могут быть определены на уровне класса Main для установления свойств агентов. Впрочем, можно легко определить другие иерархии в агентной модели, например, могут быть в качестве классов объявлены компании-агенты, которые содержат служащих-агентов и общаются с потребителями-агентами.

3.6. Резюме

Технология HLA предполагает, что отдельные системы имитации (или несколько систем имитаций), предназначенных для использования в одном приложении, могут быть легко использованы в другом приложении, если их разработчики придерживаются ***концепции федератов***.

Федерат – это одна или набор взаимодействующих систем имитации. Природа федератов может быть весьма разнообразной (программные системы, тренажёры). Федераты объединены в федерации.

Федерация – это объединение компонентов имитационного моделирования, называемых ***федератами*** (federates).

В качестве федератов могут рассматриваться:

- имитационные модели и системы имитации (по терминологии стандарта HLA – «созданные» (constructive) участники);
- тренажёры и модели, интерактивно управляемые людьми – по терминологии стандарта HLA – «виртуальные» (virtual) участники;
- реальные образцы техники или системы связи и управления (по терминологии стандарта HLA – «живые» (live) участники);
- специализированное программное обеспечение, предназначенное, например, для визуализации или для сбора и обработки информации.

RTI (Run Time Infrastructure) – это второй функциональный компонент технологии HLA. Этот компонент представляет по сути дела операционную систему, обеспечивающую взаимодействие федераций и федератов.

Итак, стандарт (технология) HLA включает следующие компоненты:

- ***спецификация интерфейса*** (Interface Specification);
- ***шаблон объектных моделей OMT*** (Object Model Template), задающим формат информации, представляющей общий интерес для нескольких участников процесса моделирования;
- ***правила стандарта HLA***, к которым относятся десять базовых правил, определяющих основные принципы разработки программного обеспечения имитационного моделирования в среде HLA;
- специально разработанная для поддержки HLA ***операционная система Run Time Interface***, которая включает шесть базовых групп по управлению интерфейсом.

Стандартизированная структура (шаблон) моделей объектов HLA ***Object Model Template*** (OMT) необходима по следующим причинам:

- OMT обеспечивает доступный для понимания механизм обмена данными и общей координации среди членов федерации;
- OMT обеспечивает общий стандартизированный механизм для описания возможностей каждого члена федерации;
- OMT способствует взаимодействию и многократному использованию моделирований и их компонентов;
- OMT упрощает приложение и проект, предоставляя общую структуру и общий набор инструментальных средств для развития моделей объектов HLA.

Для определения соотношения времени модели и реального времени моделирования при исполнении каждой федерации определяется ***глобальный масштабный коэффициент реального времени***.

Службы управления временем, поддерживаемые в стандарте HLA, должны реализовывать два сервиса для выполнения федераций:

- ***сервисы передачи сообщений***: определены различные типы сервисов, что обеспечивает различную степень их надежности, различный порядок сообщений и затраты (латентность и пропускная способность сети);
- ***сервисы продвижения времени***: реализованы различные примитивы для выполнения запросов на продвижение в логическом времени: предусмотрен простой протокол для того, чтобы дать возможность федерату управлять потоком обновлений атрибутов и запросов на взаимодействие для этого федерата.

Сервисы передачи сообщений классифицируются по следующим признакам:

- надежность передачи сообщения;
- порядок сообщений.

В качестве ***инструментальных средств*** распределенного моделирования применяют ***параллельные и распределенные вычислительные системы***.

Термин ***параллельные системы***, как правило, применяется к суперкомпьютерам для того, чтобы подчеркнуть использование многопроцессорной архитектуры.

Термин ***распределенная система*** обозначает набор независимых компьютеров.

Кластер – простая вычислительная система, ресурсы которой используются одной рабочей группой.

Мультиагентные системы (кросс-платформенные распределённые интеллектуальные системы) представляют собой совокупность интеллектуальных ***агентов***.

Агент – это аппаратная или программная сущность, способная действовать в интересах достижения целей, поставленных пользователем.

Агенты ***разнообразны***, ***разнородны*** и ***динамичны***. Агенты активны по отношению к своим атрибутам и правилам поведения.

EMCAS – это сложная адаптивная система рынка электроэнергии. Она представляет собой ***агентную модель*** рынка электроэнергии, спроектированную для исследования реструктуризации конкурентного рынка цен на электричество.

Настольные системы агентного моделирования могут быть использованы для обучения тому, как моделировать с использованием агентов, протестировать концепции разработки агентных моделей и проанализировать результаты.

Крупномасштабное агентное моделирование расширяет возможности простых агентных настольных моделей и позволяет большему числу агентов (от тысяч до миллионов) участвовать в сложных взаимодействиях.

Repast – это ведущий открытый и свободный источник библиотек для крупномасштабного агентного моделирования.

Swarm – это открытый и бесплатный набор библиотек с открытым кодом и в настоящее время поддерживается Swarm Development Group (SDG).

NetLogo – это другая кросс платформенная мультиагентная среда для моделирования, которая широко используется и поддерживается. Первоначально основанная на системе StarLogo, NetLogo приспосабливает агентные системы, состоящие из комбинации живых и программных агентов-участников.

AnyLogic – единственный инструмент имитационного моделирования, который поддерживает все подходы к созданию имитационных моделей: процессно-ориентированный (дискретно-событийный), системно динамический и агентный, а также любую их комбинацию, т.е. комбинированный подход.

СПИСОК ЛИТЕРАТУРЫ

1. Алиев, Т.И. Основы моделирования дискретных систем. / Т.И. Алиев.– СПб: СПбГУ ИТМО, 2009. – 363 с.

2. Советов, Б.Я., Яковлев, С.А. Моделирование систем: Учебник для вузов. – 6-е изд., стер. / Б.Я. Советов, С.А. Яковлев. – М.: Высшая школа, 2009. – 343 с.

3. Советов, Б.Я., Яковлев, С.А. Моделирование систем Практикум: Учебное пособие для вузов. – 4-е изд., стер. / Б.Я. Советов, С.А. Яковлев. – М.: Высшая школа, 2009. – 295 с.

4. Емельянов, В. В., Ясиновский, С. И. Имитационное моделирование систем. Язык и среда РДО. / В.В. Емельянов, С.И. Ясиновский. – М.: МГТУ имени Н.Э. Баумана, 2009. – 584 с.

5. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен. – СПб: Питер, 2003. – 877 с.

6. Цимбал, А.А. Технологии создания распределенных систем / А.А. Цимбал, М.Л. Аншина. – СПб: Питер, 2003. – 732 с.

7. Кельтон, В., Лоу, А. Имитационное моделирование. Классика CS. 3-е изд. / В. Кельтон, А. Лоу. – СПб: Питер; Киев: Издательская группа BHV, 2004. – 847 с.

8. Эндрюс, Г. Основы многопоточного, параллельного и распределенного программирования / Г. Эндрюс. – М.: «Вильямс», 2003. – 512 с.

9. Бэкон, Д. Операционные системы. Параллельные и распределенные системы / Д. Бэкон, Т. Харрис. – СПб: Питер, 2004. – 800 с.

10. Гофф, Макс К. Сетевые распределенные вычисления: достижения и проблемы / Макс К. Гофф. – М.: КУДИЦ-ОБРАЗ, 2005. – 320 с.

11. Дейтел, Х.М. Операционные системы. Распределенные системы, сети, безопасность / Х.М. Дейтел, П.Дж. Дейтел, Д.Р. Чофнес. – М.: «Бином-Пресс», 2006. – 704 с.

12. Павловский, Ю.М., Белотелов, Н.В., Бродский, Ю.И. Имитационное моделирование. / Ю.М. Павловский, Н.В. Белотелов, Ю.И. Бродский. – М.: Академия, 2008. – 236 с.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра вычислительной техники (ВТ) Санкт-Петербургского государственного университета информационных технологий, механики и оптики является одной из авторитетнейших научно-педагогических школ России. Более 70 лет кафедра ведет подготовку высококвалифицированных специалистов в области информатики и вычислительной техники. За годы своего существования на кафедре подготовлено более 5000 высококвалифицированных специалистов, подготовлено более 230 кандидатов наук и 36 докторов наук. Преподавателями и научными сотрудниками кафедры изданы десятки монографий, учебников и учебно-методических пособий. Кафедра имеет высококвалифицированный состав преподавателей, среди которых 9 профессоров и 14 доцентов.

Традиционно основной упор в подготовке специалистов на кафедре ВТ делается на фундаментальную базовую подготовку как в рамках общепрофессиональных дисциплин, так и специальных дисциплин, охватывающих все наиболее важные разделы информатики и вычислительной техники. Студенты старших курсов могут специализироваться в более узких профессиональных областях, таких как информационно-управляющие системы, открытые информационные системы и базы данных, сети ЭВМ и корпоративные системы. В 2011 году начата подготовка бакалавров и магистров по новому направлению «Программная инженерия».

Муравьева-Витковская Людмила Александровна

Основы распределенного моделирования

Учебное пособие

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ № 3029
Тираж 100 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики
и оптики
197101, Санкт-Петербург, Кронверкский пр., 49