СОВРЕМЕННЫЕ ТЕОРИИ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Специальный курс для магистров второго курса

Специальности 010500,010501 (направления)
Прикладная математика и информатика.

Аннотация

В настоящее время имитационное моделирование остаётся общепризнанным методом для исследований в различных областях науки, производства, бизнеса и т.д. С ростом сложности задач, которые стоят перед имитационным моделированием, возрастает необходимость в вычислительных ресурсах. По этой причине возникает необходимость в разработке распределённых систем имитации, в основе которых лежат алгоритмы синхронизации объектов, выполняющихся на различных компьютерах в сети (или различных процессорах многопроцессорной ЭВМ). Наряду с проблемами, которые возникают с синхронизацией объектов необходимо следить за сбалансированностью вычислительных ресурсов. Поэтому распределённая система имитации должна содержать подсистему балансировки, которая выполняет равномерное распределение вычислительных ресурсов в сети. Важной проблемой, которой в настоящее время уделяется большое внимание, является качество информации. В специальном курсе существует раздел, посвящённый вопросам получения «валидной» модели. Кроме того, рассматриваются вопросы хранения промежуточной информации (XML), интеллектуальной обработки результатов моделирования и интеллектуальной генерации моделей, а также рассматриваются вопросы агентного моделирования.

Спецкурс не претендует на законченность, много тем, разделов имитационного моделирования ещё следует подробно рассмотреть (например, визуальное моделирование, удалённый доступ через Интернет, анализ результатов моделирования и т.д.), а представленные в спецкурсе лекции также требуют серьёзной доработки.

Спецкурс предназначен для студентов 6 курса (2 курс магистратуры) по специальности 010501,010500 (направления) «Прикладная математика и информатика». Материалы курса могут также использоваться при проведении занятий для студентов старших курсов направления «Информационные технологии».

Автор выражает большую благодарность аспиранту и коллеге Вячеславу Ланину за консультации при написании лабораторных работ и за поддержку.

Хочется поблагодарить группу студентов-магистров 6 курса, которые выполняли лабораторные работы, готовили рефераты, презентации к ним. Это Илья Немец, Евгений Кубрак, Антон Цыбин, Алексей Данилов, Михаил Стрелков, Алексей Городилов и Дмитрий Басов.

Большую помощь оказали студенты, выполнявшие курсовые и дипломные работы по имитационному моделированию (Любовь Стретинских, Глеб Чудинов, Сергей Райс, Марк Бахтин, Сергей Кондратьев, Антон Фирсов, Павел Мальков, Андрей Хлызов, Мария Чичагова, Артём Шафранов, Михаил Попов, Стас Лавринский и др.)

Содержание

Лекция 1. **Параллельное и распределённое имитационное моделирование**

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.1.)

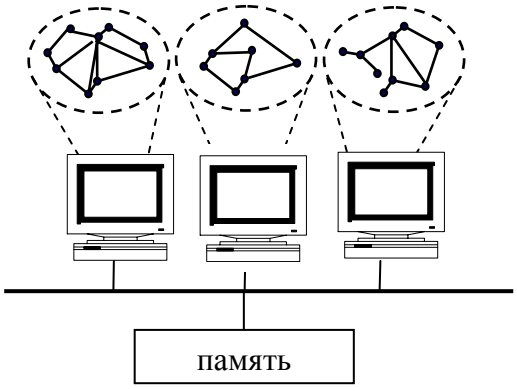


Рис. 1. Выполнение имитационной модели на нескольких процессорах многопроцессорной ЭВМ

Выполнение вычислительного эксперимента с имитационной моделью на нескольких компьютерах, объединённых в сеть (локальную или глобальную), можно рассматривать как распределённое имитационное моделирование (рис.2).

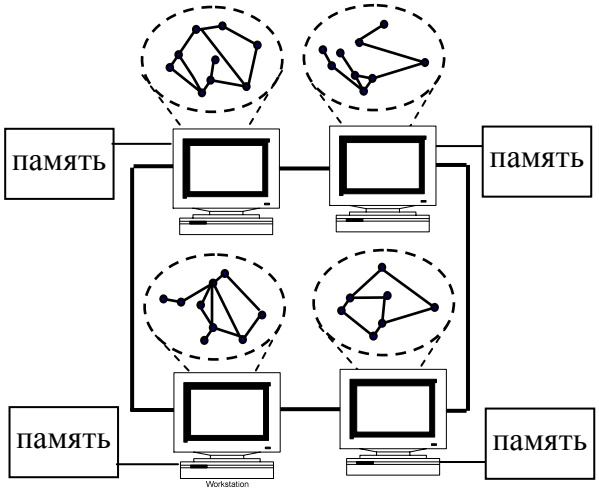


Рис. 2 Выполнение имитационной модели на нескольких компьютерах, объединённых в сеть

1.1. Причины перехода к параллельному и распределённому имитационному моделированию

Причины перехода к параллельному и распределённому имитационному моделированию обусловлены, по мнению R.Fujimoto, следующим[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.

– 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. 3).

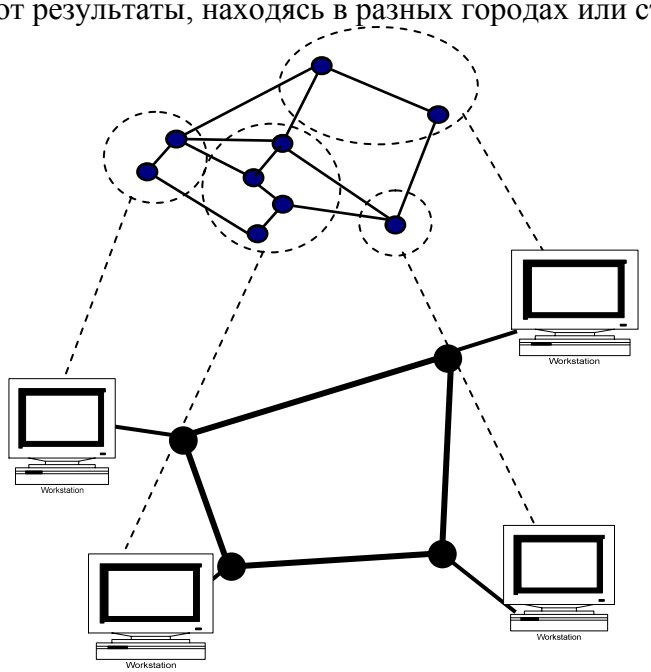


Рис. 3. Выполнение модели в глобальной сети. Совместная работа над имитационными проектами

– Кроме того, используя распределённую систему моделирования с удалённым доступом через Интернет, можно сдать в «аренду» неиспользуемые в данный момент времени вычислительные ресурсы[49].
– 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.

1.2. Параллельные и распределённые вычислительные системы

Как уже упоминалось ранее, «распределение»[9] является одной из основных тенденций компьютерных технологий настоящего времени (другой тенденцией является интеграция). 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.

Существует два термина, которые часто путают: «распределенные вычислительные системы» и «параллельные вычислительные системы». Их необходимо разделить, чтобы не возникало неоднозначности.

1.2.1. Параллельные вычислительные системы

Термин параллельные системы, как правило, применяется к суперкомпьютерам для того, чтобы подчеркнуть использование многопроцессорной архитектуры. Основными классами архитектур современных параллельных [14] компьютеров являются:

– SMP - 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 ПК. 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.

1.2.2. 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) (см.рис.4). Дополнительное программное обеспечение называют промежуточным (middleware).

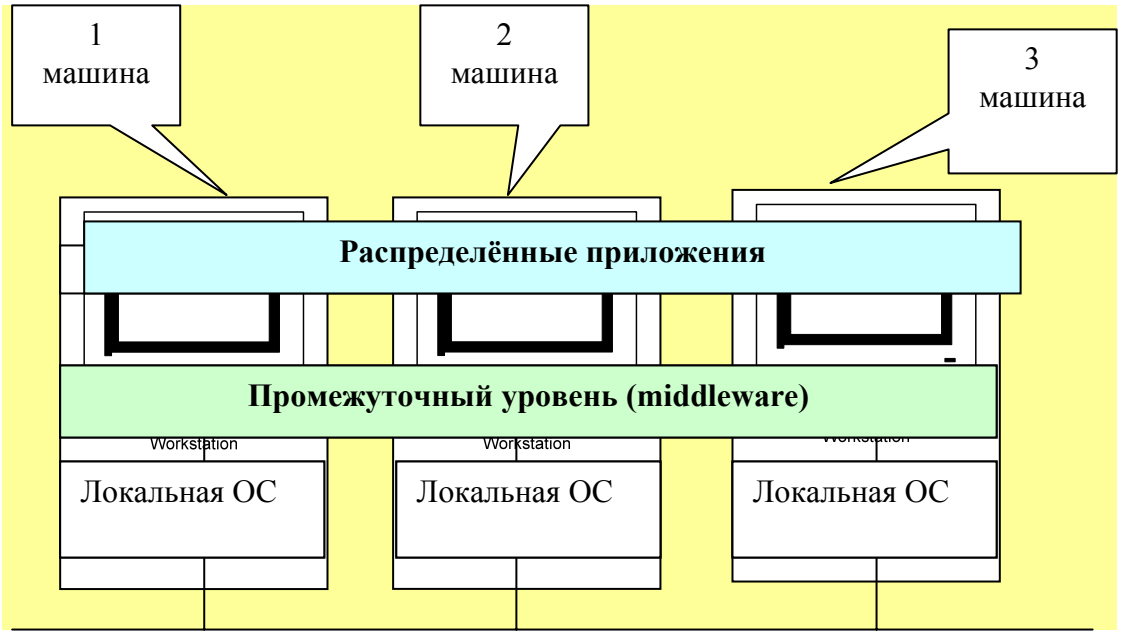


Рис. 4. Распределённая система, организованная в виде служб промежуточного уровня

Другие авторы[35] 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[35].

Каждая из этих типов распределённых систем обладает своими отличительными характеристиками, которые следует учитывать при разработке программного обеспечения. Так локальные и глобальные сети отличаются параметрами надёжности, временем коммуникации, однородностью вычислительных узлов:

- ***Параметры надежности***. Под параметром 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.
- ***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.

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

Глобальная сеть

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.

– ***Безопасность.*** 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.

Локальная сеть

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 распредёлённый 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

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.

Взаимодействующие процессы

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».

1.3. Распределённые системы имитационного моделирования

Исторически сложилось так, что термин «распределённая система имитации» относился к системам, выполняемым на вычислительной системе, узлы которой географически могли находиться на весьма отдалённом расстоянии друг от друга. В этих системах время, которое затрачивалось на передачу сообщений между узлами, было сравнительно велико, а производительность систем – низкая. Термин «параллельная система имитации» применялся к системе имитации, функционирующей на высокопроизводительных вычислительных системах. Вычислительные узлы этих систем могут находиться на незначительном расстоянии (например, в одном здании или одной комнате). В этих системах время, затрачиваемое на коммуникацию между узлами, незначительно по сравнению со временем, затрачиваемым на вычисления.

Однако в связи с появлением кластеров рабочих станций, корпоративных и GRID-систем, границы между параллельными и распределёнными системами имитации стираются. В дальнейшем все системы имитации, функционирующие на вычислительной системе с несколькими узлами, будем называть *распределёнными*.

1.3.1. Распараллеливание вычислений в распределённых системах имитации

Как происходит распределение вычислений между узлами?

За счёт чего можно сократить время проведения имитационного эксперимента?

В [3] авторы предлагают:

– Выполнять параллельно с ходом имитационного эксперимента специализированные функции, а именно: генерацию псевдослучайного числа,

управление списком будущих событий, сбор статистических данных (специализированные функции)[17]. Следует отметить, что выигрыш во времени в этом случае является небольшим.

– При декомпозиции иерархической модели происходит декомпозиция события на подсобытия (sub event). Эти подсобытия следует выполнить параллельно (иерархическая декомпозиция)[18]. Выигрыш во времени при декомпозиции модели зависит от самой модели. Для некоторых моделей (например, для моделей СМО) декомпозиция выполняется легко. Каждая очередь в модели СМО может быть выполнена на отдельном процессоре (компьютере). Для других имитационных моделей декомпозиция не является тривиальной.

– Несколько имитационных прогонов последовательной имитационной модели следует выполнять на нескольких процессорах (распараллеливание репликаций)[17] (см. рис 5). В этом случае вычислительные узлы должны обладать соответствующими вычислительными ресурсами, для выполнения имитационного прогона всей модели. В настоящее время мощность компьютеров достаточно велика, но возникает ещё одна проблема: если репликация может быть выполнена только при некотором условии, которое зависит от предыдущей репликации, то они не могут быть выполнены параллельно.

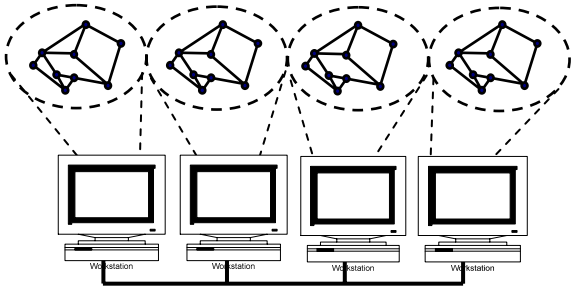


Рис. 5. Выполнение нескольких имитационных прогонов параллельно

– Распределить выполнение имитационной модели на несколько процессоров (компьютеров). Это распределение можно назвать ещё распределением на уровне объектов или процессов, в отличие от описанного выше распределения на уровне моделей. («Распределение» на уровне моделей – то есть модель является неделимой и должна полностью выполняться на той машине, на которой она начала выполняться, а разные модели могут выполняться на разных машинах). Это более простой, но неоптимальный вариант. «Распределение» на уровне объектов (процессов) модели означает, что часть объектов одной и той же модели выполняется на одном сервере, а часть на других. Сложность реализации такой системы заключается в том, что если в первом случае все объекты выполняются в контексте одного процесса и коммуникации между ними можно легко реализовать, во втором случае объекты должны взаимодействовать через сеть, что усложняет синхронизацию объектов, отладку моделей и саму реализацию этих объектов.

1.3.2. Два направления в развитии распределённого моделирования

Развитие распределённого моделирования идёт по двум направлениям:

Первое 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[52], SPEEDES[53], Task-Kit[54] и др[93]. В нашей стране тоже ведутся работы по созданию распределённых систем имитации (Мера (Новосибирск[116]), Диана (МГУ[117]). 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 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.

1.3.3. Технологии, используемые при реализации распределённых имитационных систем

Наиболее часто используемыми технологиями являются технологии точка-точка (peer-to-peer) и клиент-сервер (client-server).

Технология клиент-сервер (см. рис.6) применяется при выполнении системы имитации на нескольких компьютерах, объединённых локальной сетью. Большая часть программного обеспечения выполняется на сервере. Пользователи могут иметь удалённый доступ к серверу, на котором выполняется моделирование. Чаще всего этот подход применяется в распределённом моделировании, которое используется в играх.

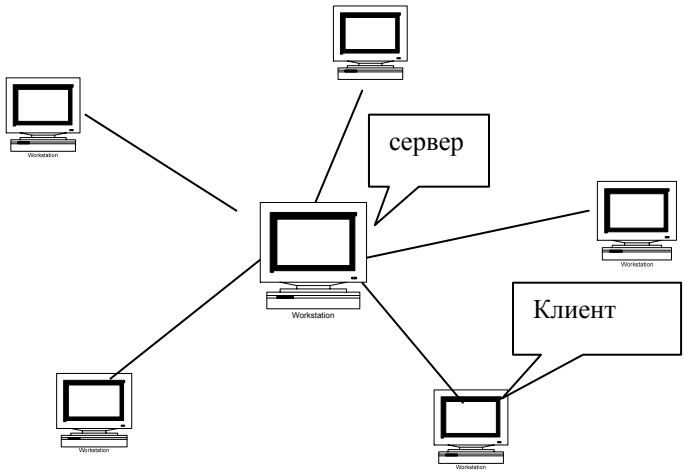


Рис. 6 Технология клиент-сервер

Технология точка-точка (рис.7.) реализуется на различных узлах вычислительной системы. Эти узлы могут объединяться глобальной сетью.

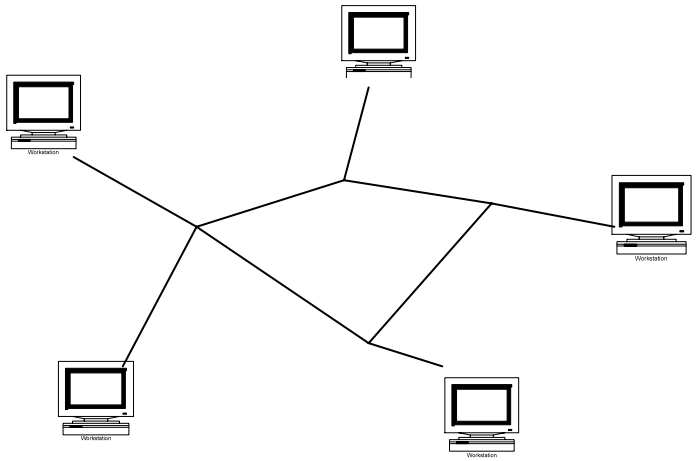


Рис. 7 Технология точка-точка

1.3.4. История создания распределённых систем имитационного 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 прошлого столетия. 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[22]) и (Bryant, 1977[20]). 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 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. Это системы моделирования для обучения, анализа и тестирования персонала (стандарт HLA будет подробно рассмотрен в лекции №3).

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

Лекция 2. **Управление временем в распределённых системах имитации**

Известно, что большую роль в имитационных моделях играет фактор времени. По определению имитационное моделирование является методом исследования динамических систем, в котором реальный объект (система) заменяются имитационной моделью. Процесс моделирования сопровождается отображением реального объекта (системы) в модель, которая выполняется, изменяя свое состояние с течением времени, причём время необратимо, оно не замедляется и не ускоряется. Состояние системы определяется состоянием её элементов, а каждый элемент обладает набором свойств (характеристик).

Прежде всего, следует определить, что следует понимать под термином «время» в имитационном моделировании. В работе Fujimoto [1] и других работах по имитационному моделированию[8] различают: физическое (physical), модельное (system time) и процессорное (wallclock time). Рассмотрим более подробно все эти 3 разновидности:

- ***Физическое время*** (physical-T_p) – это время, которое используется в реальной (физической) системе, которую моделируют. Например, мы моделируем работу некоторого предприятия в течение рабочего дня с 8.00 до 17.00.
- ***Модельное время*** (simulation time - T_s) – это представление физического времени в модели. Так работу предприятия в модельном времени можно представить отрезком времени [8.00,17.00], за единицу модельного времени (h) можно принять временной интервал в 1 минуту, в 10 минут, в 30 минут, в один час и т.д. $T_s = T_p/h$.
- ***Процессорное время*** (wallclock time - T_w) – время работы симулятора на компьютере. Так, например, моделирование предприятия может занять 1 час работы на компьютере. Иногда (при использовании тренажёров) продвижение модельного времени должно быть синхронизировано с процессорным. Такое моделирование называют *моделированием в реальном времени* (real time). Действительно, при использовании тренажёров человек погружается с виртуальную среду, которая должна выглядеть как можно более реалистичной. С другой стороны моделирование должно выполняться как можно скорее (as-fast-as-possible), т.е. *модельное* время продвигается с *гораздо большей скоростью*, чем процессорное. Например, работа некоторого физического процесса длится несколько суток, единицу модельного времени выбирают равной одному часу, а процесс моделирования на компьютере выполняется за 30 минут.

Время – частично упорядоченное множество $T = \{t_1, t_2, \ldots, t_n\}$.

Итак, одной из важнейших задач системы имитации является продвижение модельного времени. Управляющая программа, которая выполняла эти действия в последовательном моделировании, называлась симулятором. В распределённом моделировании используют иной термин: алгоритм синхронизации. Алгоритм синхронизации предназначен для синхронизации компонентов моделирования. В качестве таких компонентов в распределённом моделировании чаще всего фигурируют логические процессы (LP). Логические процессы (LP-*logical processes*) интерпретируют поведение физических процессов (PP-*physical processes*).

Вначале кратко рассмотрим алгоритм продвижения модельного времени в последовательном моделировании.

2.1. *Последовательное моделирование*

Известно, что существуют различные виды имитационных моделей, в основе которых лежит та или иная концепция:

- События (events).
- Процессы (processes).
- Объекты (objects) или агенты (agents).
- Непрерывные (continues) модели (системная динамика).

Как известно, событие – это изменение состояния системы, причём событие происходит мгновенно. В промежутке между двумя событиями модель остаётся неизменной. Процесс – это последовательность активностей, а активность – это элементарная работы по переводу системы из одного состояния в другое. Активность начинается и завершается событием (рис.8.).

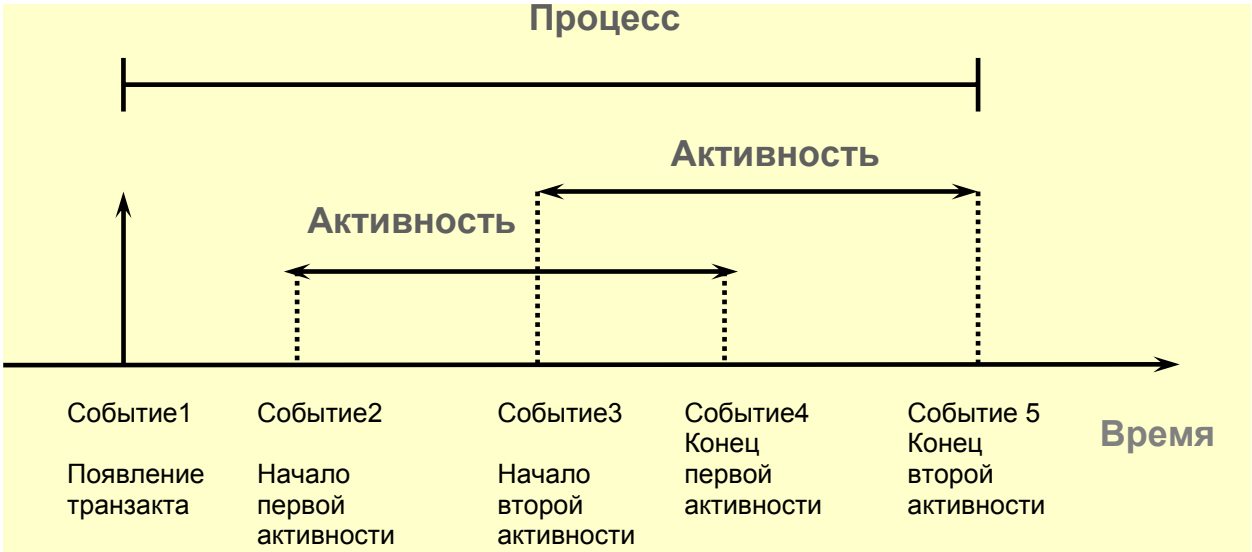


Рис. 8 Процессы, активности и события

Как уже говорилось ранее, система моделирования, управляющая выполнением модели, должна уметь продвигать модель из одного состояния в другое. Продвижение модели из одного состояния в другое выполняется по определённым правилам, эти правила определяют сценарий поведения модели во времени, причинно-следственные связи между активностями.

В зависимости от того, какая концепция лежит в основании имитационной модели, системы моделирования делятся процессо-ориентированные, событийно-ориентированные, объектно-ориентированные.

2.1.1. Событийно-ориентированное моделирование

В событийно-ориентированных системах моделирования приняты следующие соглашения:

а. модель продвигается во времени от события к событию, которые изменяют состояние модели,

б. логика наступления событий определяет последовательность смены состояний модели, которые связаны с наступлением этих событий;

в. время продвигается от события к событию;

г. эта парадигма широко использовалась в 60-70 годы.

Примером событийно-ориентированных систем является системы моделирования SIMSCRIPT, SMPL [56]и т.д.

2.1.2. Процессо-ориентированное моделирование

Для ***процессо-ориентированных*** систем моделирования характерно:

а. Модель представляет собой поток транзактов, который продвигают их от одного шага процесса к другому.

б. Состояние модели изменяется в дискретные моменты времени, каждый шаг процесса связан с запуском последовательности событий.

в. Шаги повторяются в течение всего времени моделирования.

г. Модель является такой же гибкой и эффективной, что и событийно-ориентированная, но менее абстрактна.

д. И в настоящее время процессо-ориентированные системы моделирования находят широкое применение.

Примером процессо-ориентированных систем моделирования являются GPSS[54,59,60], ASPOL[56]. Система моделирования GPSS находит широкое применение у российских исследователей [61,62,63, 115 и др.].

2.1.3. Объектно-ориентированное моделирование

Далее рассмотрим объектно-ориентированную имитационную модель, которая обладает следующими характеристиками:

а. Модель представляет собой совокупность объектов.

б. Объекты включают данные и операции над ними.

в. Объекты представляют собой модель «актора», который может выполнить работу, изменить своё состояние и взаимодействовать с другими объектами.

г. Объекты являются основой современных языков программирования.

д. Объекты являются естественным способом описания системы.

Примером объектно-ориентированного подхода является SIMULA [55,57].

2.1.4. Агентно-ориентированное моделирование

Пегден (Pegden) [91] выделяет ещё одну разновидность систем моделирования, ещё одну парадигму – ***агентно-ориентированную***, для которой характерны следующие свойства:

а. Агентный подход - это особый случай объектно-ориентированного подхода.

б. Поведение всей системы можно рассматривать как результат поведения большого числа объектов, называемых агентами.

в. Агенты являются автономными. Могут взаимодействовать друг с другом и преследуют свои цели.

г. Агентный подход является альтернативой систем СД (системной динамики).

Перечисленные выше подходы к моделированию представляли собой дискретное моделирование. Наряду с дискретными моделями большое применение находят ***непрерывные*** (системная динамика) и ***гибридные*** модели.

Отличительные особенности непрерывных моделей (Пегден[91], А.Прицкер [58]):

а. Состояние модели изменяется не в дискретные моменты времени, связанные с возникновением событий, а непрерывно в течение всего времени моделирования.

б. Непрерывное моделирование применяется для исследования непрерывных систем моделирования (к примеру, движение объектов, течение жидкости) или агрегированных моделей дискретных систем (например, рынок, цепочки поставок, население).

в. Модель описывается дифференциальными уравнениями.

В дальнейшем будем рассматривать только дискретное моделирование.

2.1.5. Основные компоненты имитационной модели

Итак, рассмотрим компоненты, из которых обычно состоит имитационная модель (чаще всего это относится к процессо-ориентированным и событийно-ориентированным моделям):

а. Часы
б. Список событий
в. Программные средства сбора статистики
г. Подпрограмма инициализации
д. Подпрограмма продвижения времени
е. Подпрограммы, реализующие события
ж. Библиотечные подпрограммы
з. Генератор статистических отчётов
и. Управляющая программа

Подробнее рассмотрим вопросы отображения времени – механизмы продвижения времени в различных системах моделирования (см.рис.9).

2.1.6. Принципы продвижения модельного времени

Итак, существует два принципа представления времени:

а. Δt;
б. особые моменты времени.

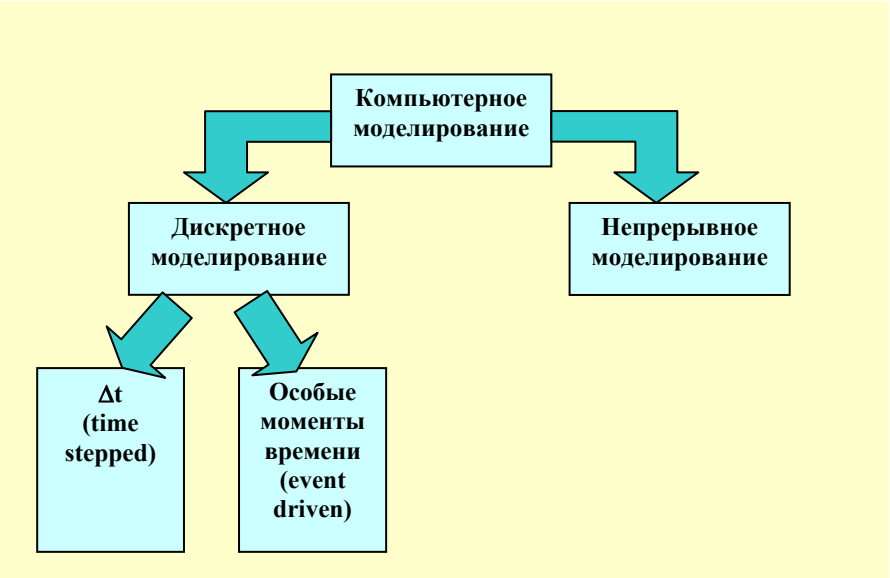


Рис. 9.Принципы продвижения времени в имитационном моделировании

Первый способ – это продвижение времени на фиксированные промежутки времени. Это принцип восходит к численным методам решения задач анализа (дифференцирования, интегрирования, дифференциальных уравнений). Продвижение на фиксированные промежутки времени очень часто применяется и в непрерывном моделировании. В методическом отношении этот способ достаточно прост. Состояние системы меняется через фиксированные промежутки времени. Тем не менее, возникает проблема выбора величины Δt.

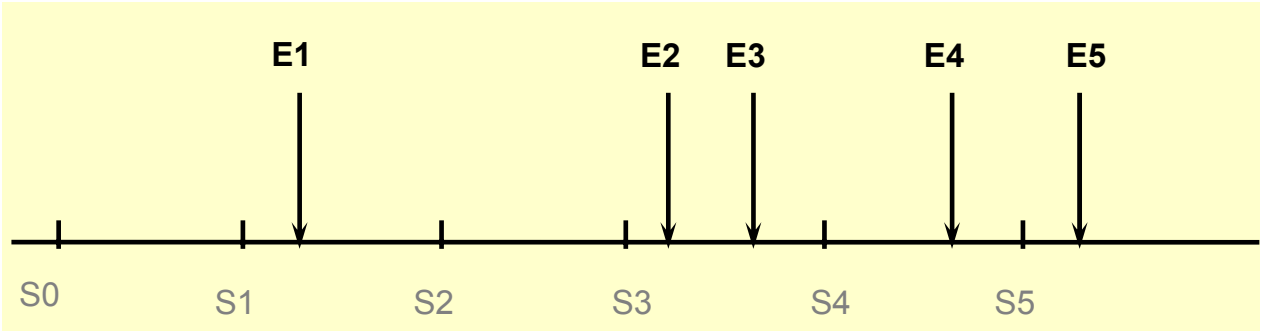


Рис. 10. Изменение времени через фиксированные промежутки

С уменьшением Δt уменьшается ошибка моделирования (оно становится более точным), но увеличивается объем вычислений (количество состояний растёт). Для достаточно гладких систем можно подобрать величину Δt и получить достаточно точные результаты. Но для многих систем использование фиксированного шага крайне неэффективно.

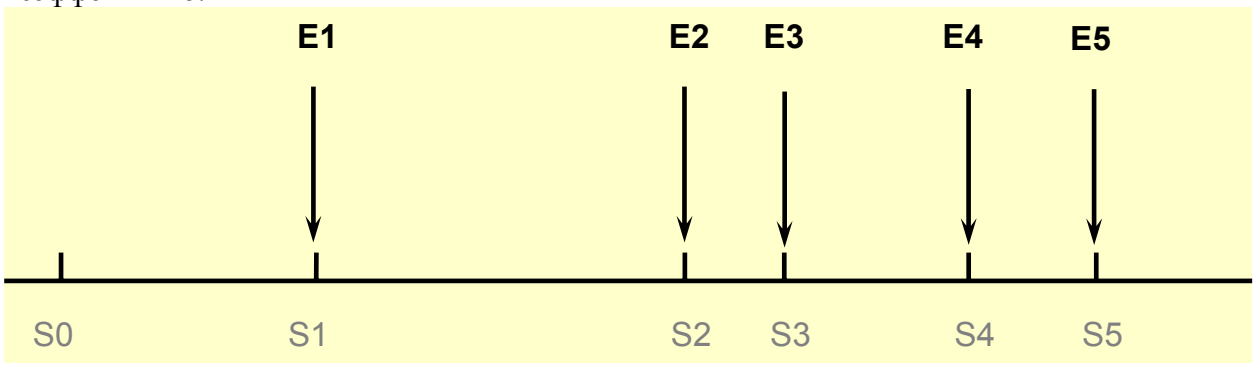


Рис. 11. Продвижение времени от события к событию

Если состояние медленно меняется на одних интервалах, но быстро на других, то целесообразно сделать этот шаг переменным. Именно таким образом организовано продвижение времени в системах с автоматическим изменением шага. Шаг увеличивается на участках, где состояние системы меняется редко и уменьшается на участках с частым изменением состояния системы. Тем не менее, если мы можем пропустить момент времени, когда состояние системы изменяется. Вследствие этого, ошибка может быть чрезмерно большой (рис.10 и 11). Необходимо использовать механизм, который позволял бы устранять такие ошибки.

Однако для многих дискретных систем, в которых состояния системы меняется скачкообразно, более предпочтительным является второй принцип продвижения модельного времени. При вычислении нового момента времени используется предыдущий момент времени.

Итак, фиксированные промежутки времени легко реализовать, но выбор величины шага в значительной степени влияет на точность и скорость моделирования. События должны быть упорядочены.

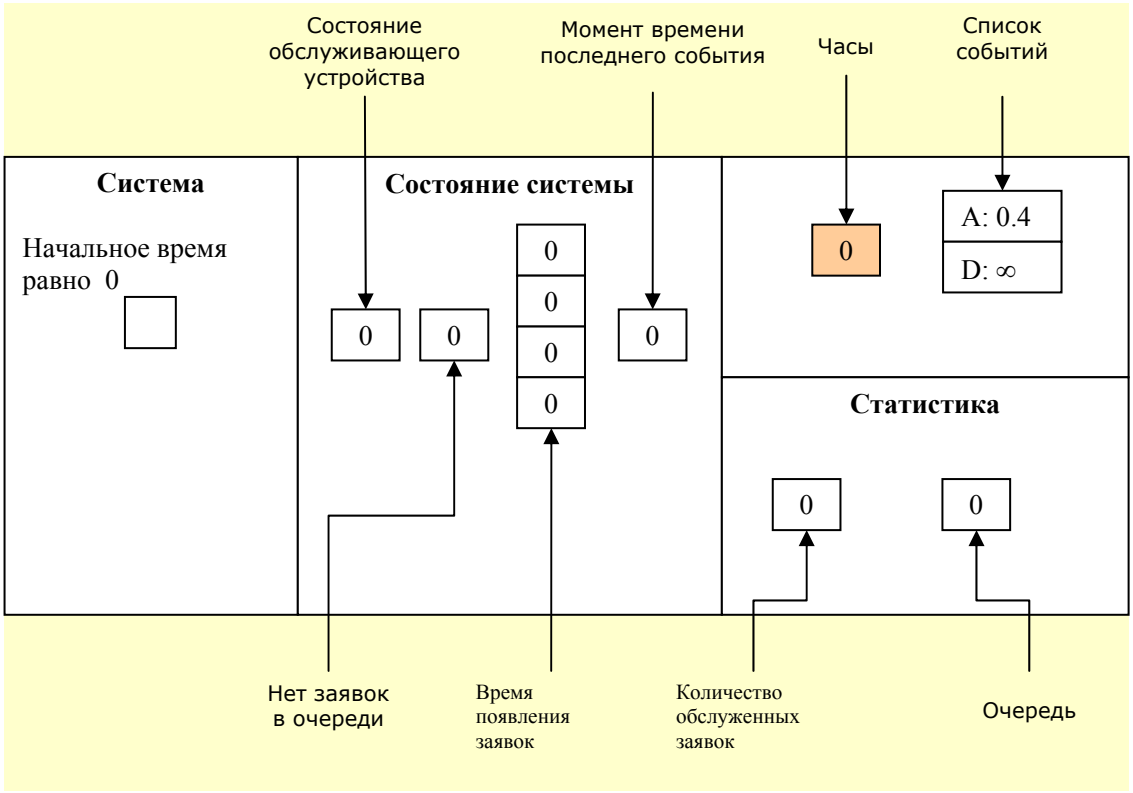


Рис. 12. Начальное состояние модели одноканальной СМО

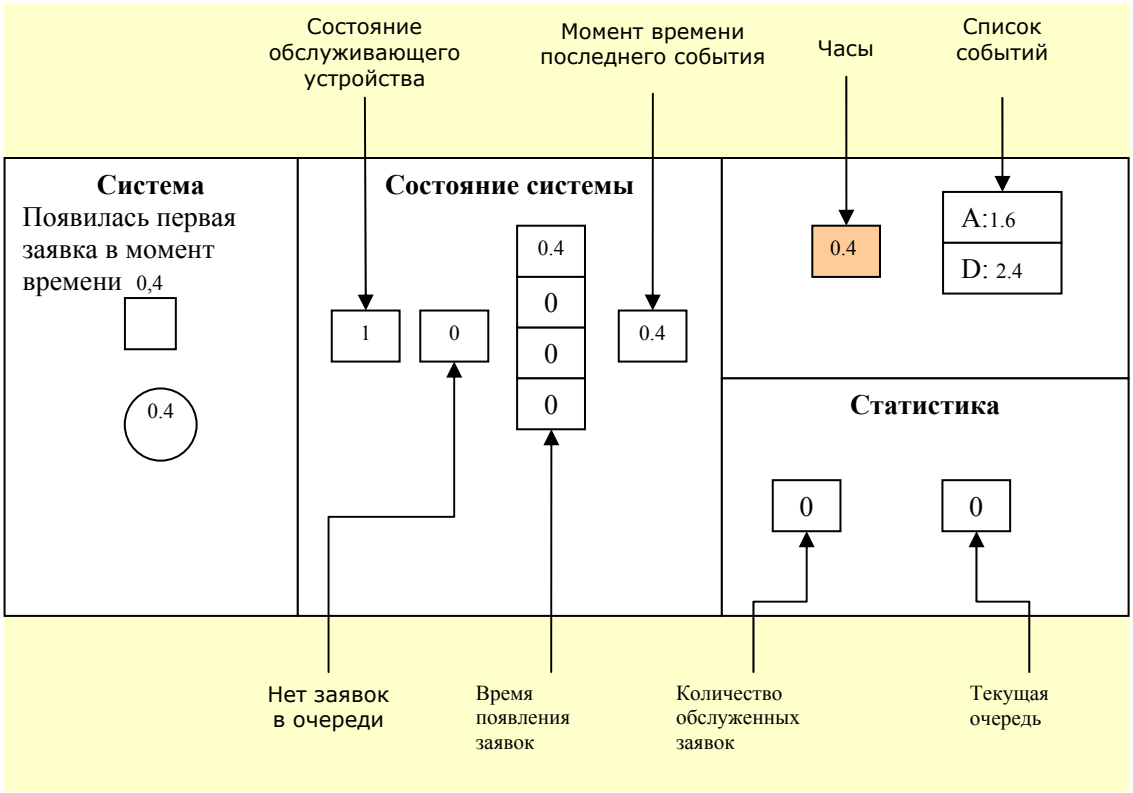


Рис. 13. Состояние системы в момент обслуживания первой заявки

При использовании принципа особых моментов времени события определяют длину промежутка времени. Длина шага определяется временным интервалом между двумя следующими друг за другом событиями. Таким образом, величина шага является величиной переменной.

А теперь на примере одноканальной системы массового обслуживания рассмотрим, каким образом продвигается модельное время.

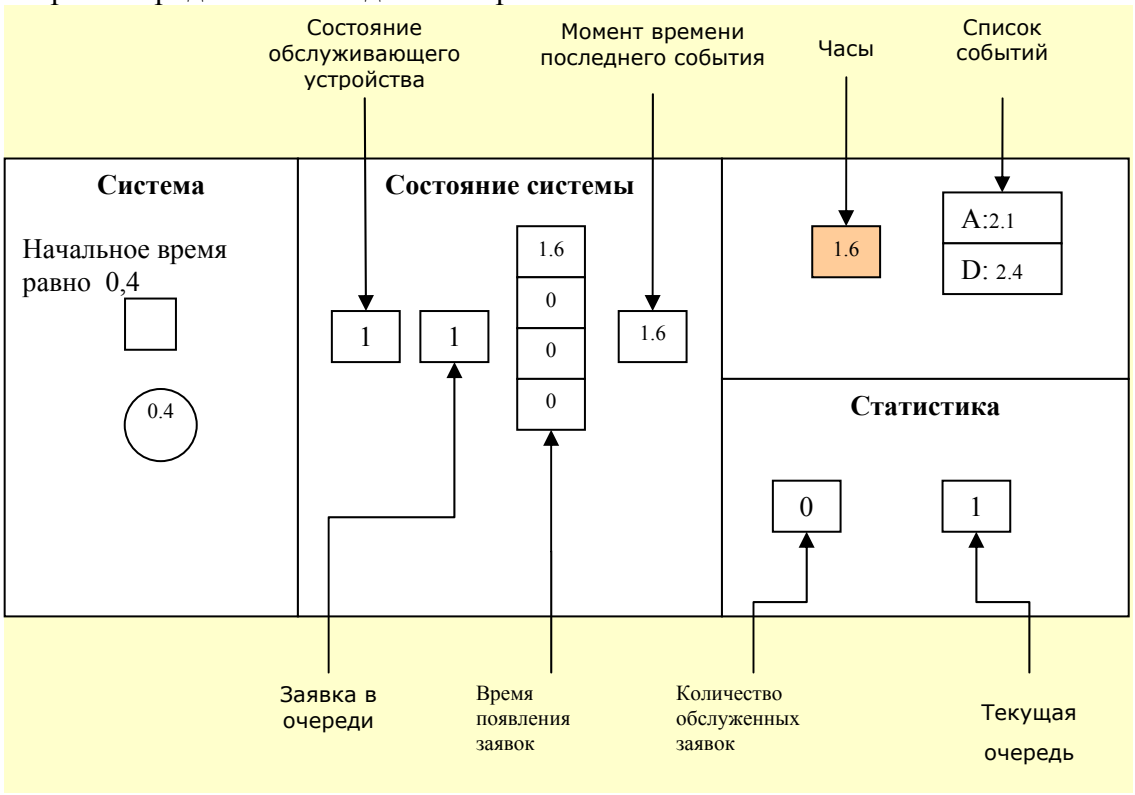


Рис. 14. Состояние системы в момент времени появления второй заявки

Пусть в системе появляются заявки (случайным образом, по некоторому закону распределения) и встают в очередь к обслуживающему прибору, если он занят и немедленно обслуживаются в противном случае.

В начальный момент времени состояние моделируемой системы таково: очередь пуста, обслуживающее устройство свободно. Все переменные, предназначенные для сбора статистики, имеют нулевое значение. В списке событий располагаются события, связанные с появлением заявки (A – arrival), и окончанием обслуживания (D – departure). Средства для сбора статистики накапливают информацию о количестве обслуженных заявок и о текущей длине очереди. Первоначально в списке событий A=0,4 (появление первой заявки запланировано на 0,4).

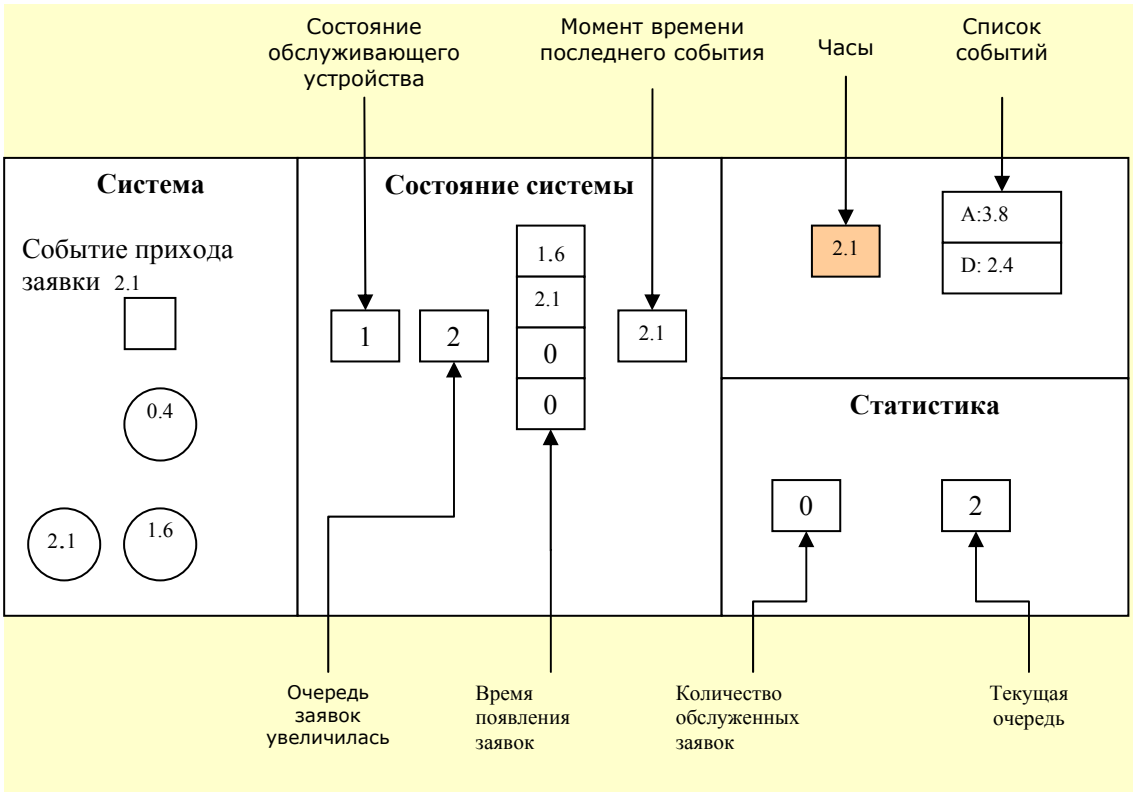


Рис. 15. Состояние системы в момент выполнения третьего события (появление третьей заявки)

Появление первой заявки сопровождается следующими действиями: поскольку обслуживающее устройство не занято, то оно начинает обслуживание заявки. Кроме того, выполняется планирование появления второй заявки (A=1.6) и планирование события, связанного с завершением обслуживания первой заявки (D=2.4). Очередь остаётся пустой, а состояние обслуживающего устройства становится равным 1, что означает, что статус устройства – «занято»(рис.13.).

Состояние системы в момент появления второй заявки представлено на рис.14. Часы переводятся на время, равное 1.6., планируется появление новой заявки (A=2.1), заявки появились в очереди, поскольку первая заявка ещё не обслужена.

Новая заявка появляется в момент времени, равный 2.1. Устройство занято, поэтому заявка направляется в очередь. Происходит планирование появления новой заявки на момент времени 3.8. Момент времени завершения обслуживания остаётся неизменным. Первая заявка так ещё и не обслужена. Очередь увеличилась на 1. Состояние системы в этот момент времени представлено на рис.15.

Далее управляющая программа извлекает очередное событие с минимальным временем из списка событий. Время наступления следующего события – окончание обслуживания первой заявки– 2.4. Итак, часы устанавливаются на момент времени = 2.4. Очередная заявка из очереди заявок занимает обслуживающее устройство. В очереди остаётся только одна заявка, о чём свидетельствует переменная «Текущая очередь», предназначенная для сбора статистики. Количество обслуженных заявок также увеличивается на 1 (рис. 16).

Итак, мы видим, что системное время имитации совпадает со временем последнего обслуженного события.

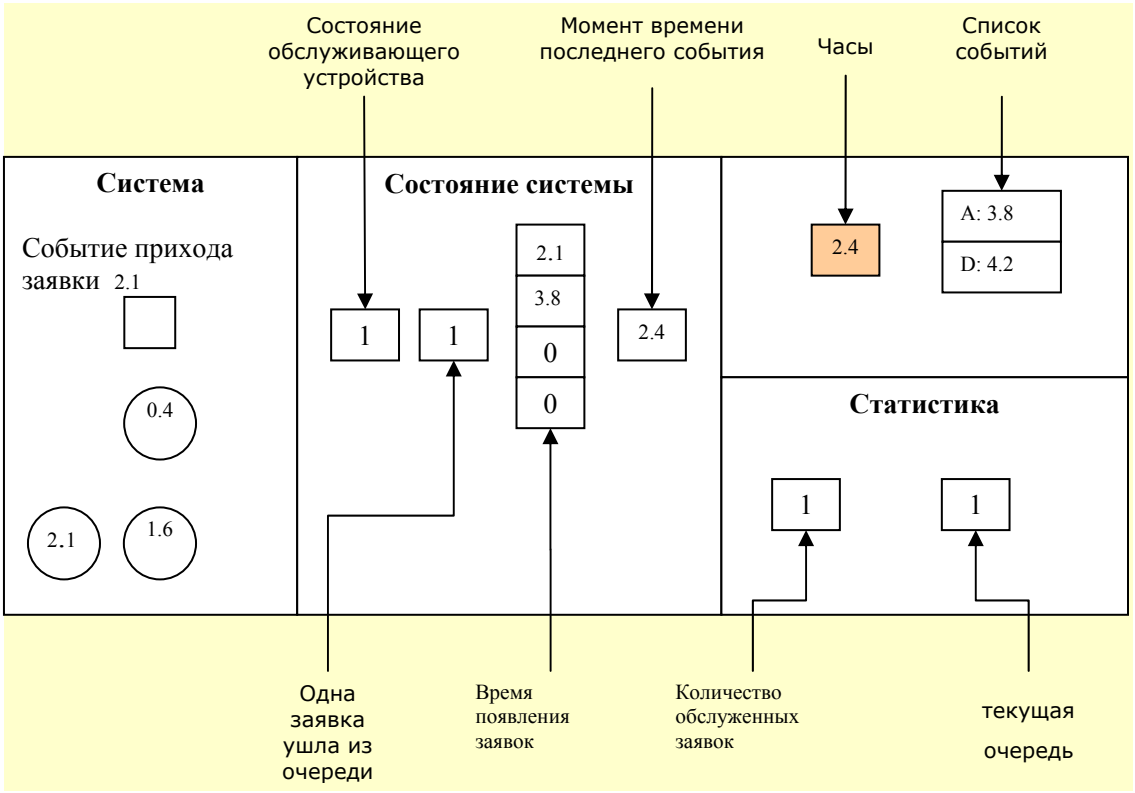


Рис. 16. Состояние системы после выполнения события, связанного с окончанием обслуживания заявки прибором.

И в событийно-ориентированном моделировании, и в процессо-ориентированном механизм продвижения времени связан со списком запланированных (или будущих) событий (GPSS или списком управляющих событий в Симула-67). Список событий (event list) ещё называют календарём событий.

2.1.7. Календарь событий и оптимизация работы с ним

Календарь событий состоит из элементов, каждый из которых содержит два поля:

- ссылку на запланированное событие (e_i);
- модельное время, на которое запланировано событие (t_i).

Таким образом, можно сказать, что календарь событий (обозначим его SE) – это список элементов l_i, где каждый элемент l_i представляет собой пару (e_i, t_i).

Управляющая программа (симулятор) должна выбрать из календаря событий событие с минимальным временем. Это время становится текущим модельным временем. Симулятор присваивает системной переменной с текущим модельным временем (назовём эту переменную именем Systemtime) минимальное значение времени, на которое запланировано событие из календаря событий, т.е. Systemtime = min (t_i). Далее симулятор передаёт управление событию e_i (с минимальным временем).

Цель: Необходимо выбрать очередное событие в календаре событий и передать ему управление:

WHILE (не конец моделирования)

Просмотреть список SE = {l_i}, где $l_i = (e_i,t_i)$, $0<=i<=n$ и выбрать элемент с минимальным временем t_i.

Systemtime = min(t_i)

Передать управление событию e_i из элемента $l_i=(e_i,t_i)$.

Обработать это событие. Если событие e_i запланировало новое событие e_j , то его следует поместить в список событий SE

END-LOOP

Если сразу несколько событий запланировано на одно и то же время, то они выполняются последовательно друг за другом (квазипараллельно), системное время systemtime не изменяется до тех пор, пока все эти события не будут обработаны.

Для того, чтобы поиск был эффективным обычно список запланированных событий упорядочивают по возрастанию.

В событийно-ориентированном механизме продвижения времени обработка очередного события связана с планированием нового события ej, выполнение которого запланировано на некоторый момент времени tj. Таким образом, в календаре событий появляется новый элемент lj=(ej,tj). Симулятор должен поместить новый элемент в календарь событий, причём упорядоченность во возрастанию должна быть сохранена.

Если число элементов в списке SE велико, поиск события с минимальным временем (и включение в список нового элемента) может занять много времени. Для оптимизации этого процесса исследователи предлагают усовершенствованные алгоритмы [24]:

- При моделировании замкнутых СМО предпринимается поиск с начала и конца списка.
- В Simscript2 для различных типов событий используют разные списки.
- Алгоритм, работающий со списком и дополнительным указателем на середину списка. Сначала происходит сравнение с нужным элементом, а потом поиск происходит в нужной половине списка.
- Алгоритм, работающий с множеством событий в виде бинарного дерева.
- Наряду с календарём событий хранится массив указателей на элементы списка. Таким образом. список событий делится на подсписки (между двумя соседними указателями). Эти подсписки соответствуют интервалам системного времени. Длины всех интервалов равны фиксированному значению. Это значение задаётся пользователем. При планировании нового события вычисляется его индекс в массиве указателей, после чего новый элемент календаря событий может быть размещён в соответствующем его подсписке.

2.2. Распределённое моделирование

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[1], 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 e_j 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.

Итак, m = {LP, ME}, где LP – это множество процессов, ME – коммуникационная среда для передачи сообщений от одного процесса другому. В свою очередь,

LP_i={Q, E, Sch, Ch, T), где

T- линейно-упорядоченное абстрактное множество с отношением порядка ≤. Множество T называют множеством моментов времени.

Q –множество состояний,

E – множество событий, $E = E_{in} \cup E_m$.

E_{in} – это множество внутренних событий процесса.

E_m –множество событий, полученных процессом i процессом от j-того процесса. Если процесс LP_i получит сообщение (событие, которое запланировано для него процессом LP_j), то он должен переупорядочить очередь локальных событий $C = ((e_1,t_1),(e_2,t_2),\ldots,(e_n,t_n))$, включив в неё пару (e_j,t_j). Очередь C упорядочена по возрастанию временных меток событий.

Sch-преобразование планирования событий процесса e_i, Sch: $E\times Q\times T \rightarrow E$;

Ch- преобразование изменения состояния Ch: $E\times Q\times T\rightarrow Q$,

Tloc – локальное время логического процесса (время последнего обработанного события из множества E).

Текущее модельное время time = min(Tloci), i= 1÷n.

(Обозначения заимствованы из монографии моего наставника А.И. Микова[24]).

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. R.Fujimoto[1] приводит такой пример (пример1): 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 другой 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 e1, произошло в момент времени t1=9). Магазин уведомил об этом банк (e2, t2=10) Сумма на счёте уменьшается S=S-N. Покупатель посещает банк с целью снять деньги со счёта (e3, t3=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.

Хронологический порядок событий: e1,e2,e3 (рис.17).

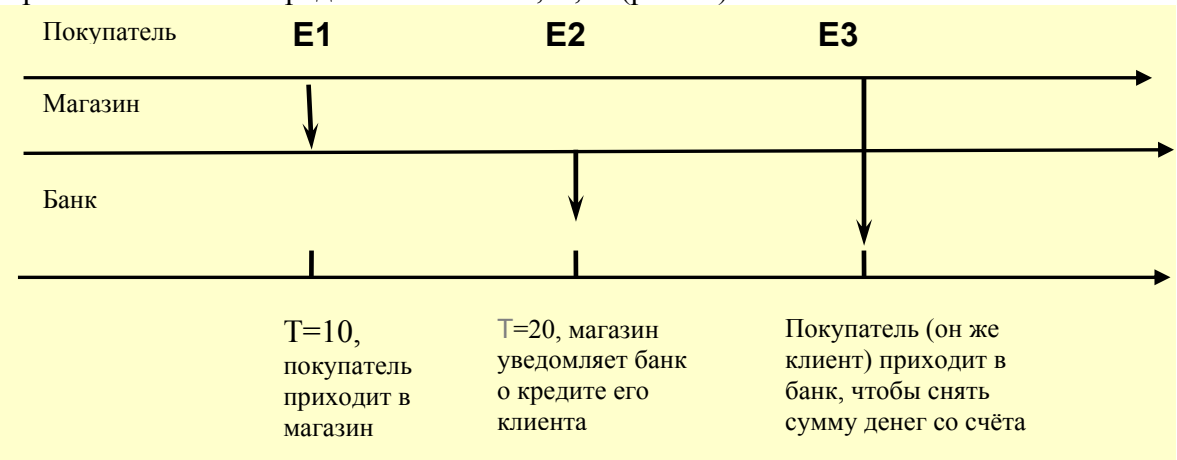


Рис. 17. Банк получает сообщения в хронологическом порядке

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 (рис.18).

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 (он выполняется на менее загруженном процессоре).

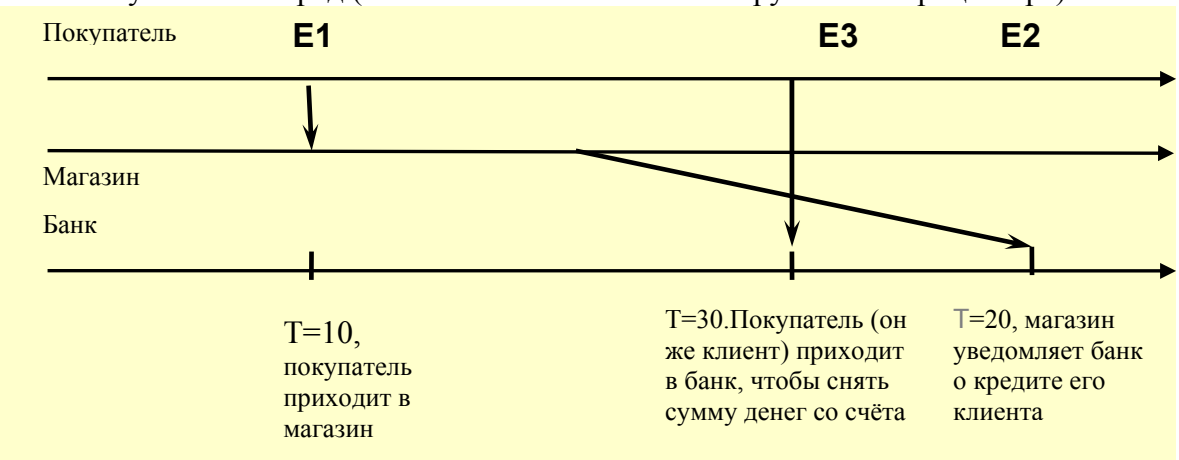


Рис. 18. Хронологический порядок событий нарушен

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[1]. В настоящее время все алгоритмы синхронизации делятся на: ***консервативные*** и ***оптимистические***.

Если мы вернёмся к примеру 1, то *консервативный алгоритм* не позволит логическому процессу к 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.

2.2.1. 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.

Большинства 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 очередной 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.

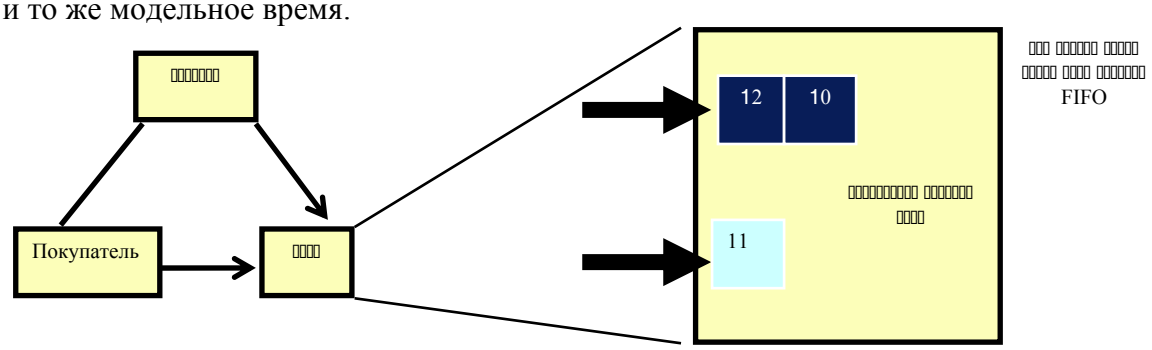


Рис. 19. Процесс «Банк» и очереди FIFO для каждой из линий связи

2.2.2. Алгоритм с нулевыми сообщениями

Первыми консервативными алгоритмами считаются алгоритмы, разработанные Bryant [20], Chandy[21], Misra[22].

Алгоритм предполагает:

- 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. 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 (рис.19). 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 (рис.20).

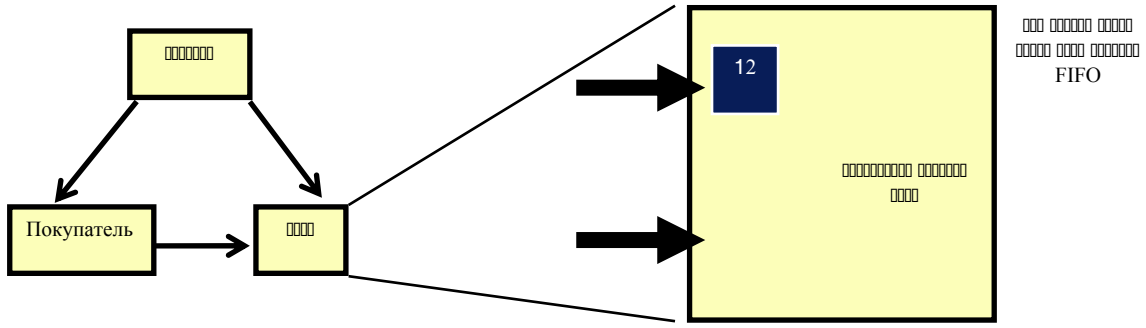


Рис. 20. Процесс блокируется, потому что на линии связи с процессом-покупателем нет сообщений.

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 3 процесса LP1,LP2,LP3. Процесс LP1 блокирован, т.к. ожидает прихода сообщения от LP3 и не может отослать сообщения процессу LP2. Процесс LP2 тоже блокирован, он ожидает сообщений от LP1. Процесс LP3 в свою очередь блокирован, поскольку в его входной очереди нет сообщений от первого процесса LP1. Теперь мы видим, что происходит циклическое ожидание процессов (тупик). Иллюстрацию примера см. на рис. 21.

Для того, чтобы избежать возникновение тупиков, Chandy и Misra предложили использовать нулевые сообщения. Процесс LP_a посылает сообщение с временной меткой T_{null} процессу LP_b. Этим процесс LP_a сообщает процессу LP_b, что он не пришлёт процессу

сообщение с временной отметкой, меньшей, чем T_{null}. Нулевые сообщения не соответствуют каким-либо физическим явлениям моделируемого объекта.

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

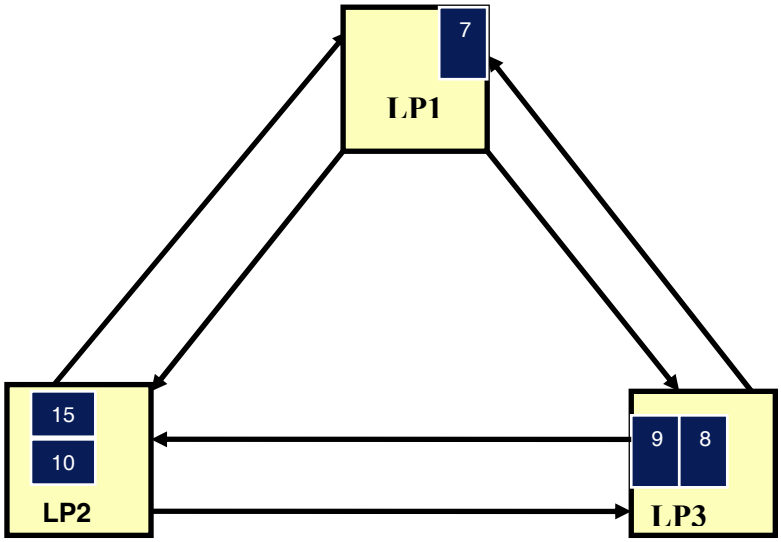


Рис. 21. Циклическое ожидание процессов.

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.

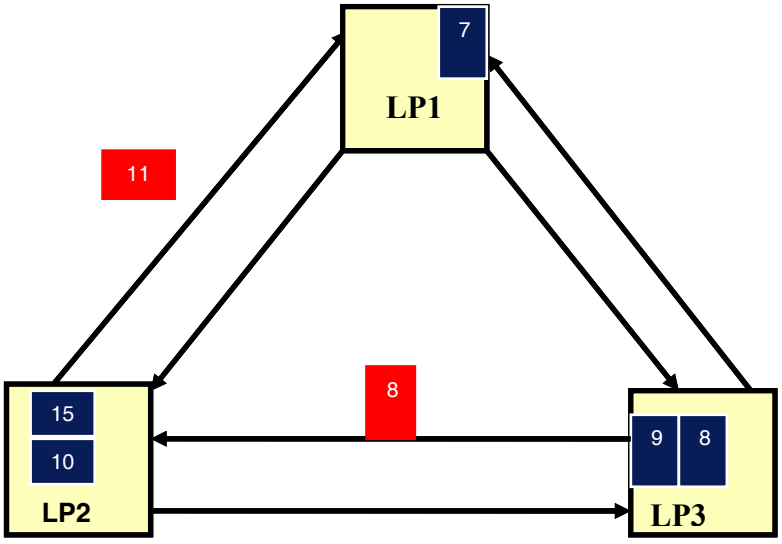


Рис. 22. Отсылка по выходным дугам нулевых сообщений.

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 (рис.22).

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.

Алгоритм с нулевыми сообщениями (выполняется каждым логическим процессом LP):

Цель: 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 + lookahead)

END-LOOP

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

2.2.3. Использование lookahead (Забегания вперёд)

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

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 lookahead («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, т.о. 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» lookahead может быть выбрана 1.

– ***Дискретное время***. При моделировании с дискретным временем любое событие может быть запланировано на время T + ΔT, где ΔT – это шаг дискретизации. В таких системах у любого процесса будет естественный lookahead 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, не взирая на возможность прихода отстающих сообщений, т.к. они всё равно не повлияют на эти события.

Обсуждаемый алгоритм допускает множество модификаций, таких как, например, подсчёт lookahead 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 lookahead возможна ситуация циклического ожидания нескольких LP с небольшими продвижениями локальных часов. В некоторых системах возможно, а зачастую и необходимо, использование нулевого предварительного просмотра (zero lookahead), однако обсуждаемый выше алгоритм не допускает нулевых предварительных просмотров.

2.2.4. Использование дополнительной информации о временной метке следующего события

Рассмотрим подробнее недостатки алгоритмы 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 (T_{locLP1} =100) и значение lookahead =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. Т.о., 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 проиллюстрирован на рис.23.

На рисунке видно, что взаимодействуют 3 логических процесса. 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 (см. рис.22). Итак, если lookahead 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 (lookahead) 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 (T_{locLP1} =200). Таким образом, мы можем избавиться от описанного выше недостатка алгоритма с нулевыми сообщениями.

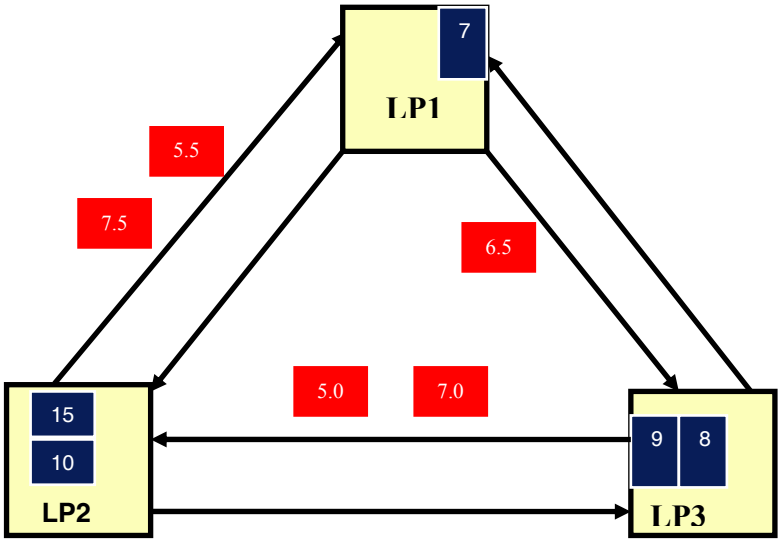


Рис. 23. Слишком много нулевых сообщений (lookahead = 0.5)

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» (lookahead).
- 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 (Ni + LAi);  Ni = время следующего события из LPi; LAi = lookahead  LPi
    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[1]. Этот подход возможен, если известна структура модели, т.е., если известны процессы, которые могут переслать сообщения конкретному процессу $LP_{i.}$

2.2.5. 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. 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 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.

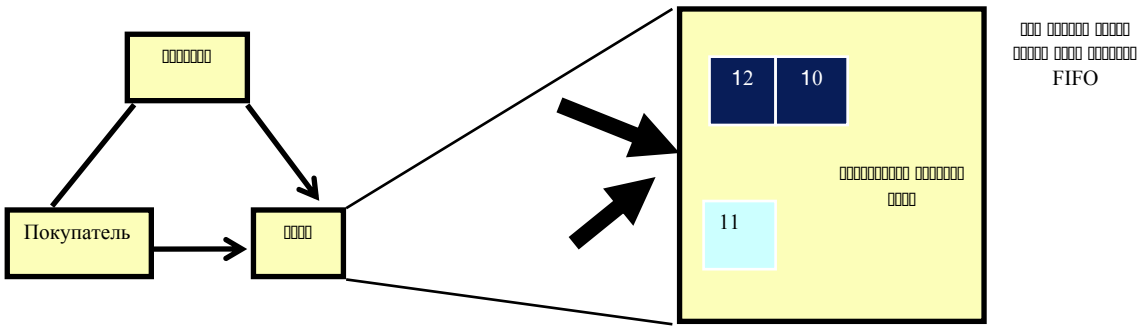


Рис. 24. В оптимистическом алгоритме все события, заппанированные на t1=10, t2=11 и t3=12 выполняются

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 2 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) [1]. 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 (Маттерн(Mattern)) 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)[1].

Существует подход, использующий «просмотр назад» (lookback). «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 (Jefferson).
- 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.

2.2.6. Выбор алгоритма для реализации системы моделирования

Алгоритмы 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[1] приводит следующие аргументы:

- Если приложение имеет хорошие характеристики для использования lookahead (заглядывания вперёд, предварительного просмотра) и вычисление этого lookahead в конечном счёте 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.

Лекция 3. **Технология HLA - High Level Architecture**

Введение

Как уже отмечалось в лекции №1, существует два направления в развитии распределённых систем имитационного моделирования:

- «Монолитные системы», которые обеспечивают пользователя всеми необходимыми сервисами для проведения распределённого моделирования. Имитационная модель в этом случае представляет собой совокупность логических процессов, которые обмениваются сообщениями друг с другом. Логические процессы распределяются по вычислительным узлам вычислительной системы и обмениваются сообщениями, используя линии связи между узлами ВС.
- Программные средства, которые позволяют объединить уже готовые системы имитации для проведения распределённого имитационного эксперимента.

Вопросы организации «монолитных» систем рассматривались в предыдущей лекции. В настоящей лекции будет подробно описана технология HLA (High Level Architecture, в переводе – архитектура высокого уровня), которая используется для проведения распределённого моделирования и объявлена в США стандартом распределённого моделирования.

HLA заменила существовавшие ранее системы DIS (Distributed Interaction System) и ALSP.

R.Fujimoto и его соавторы посвятили HLA большое количество научных работ, статей и докладов. Настоящая лекция основана на этих материалах.

3.1. Цели и задачи HLA

Технология HLA была разработана в середине 90-ых годов прошлого столетия. 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, 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 HLA.

Первоначально HLA была разработана 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.

3.2. История создания HLA

В середине 90-х прошлого столетия при министерстве обороны США был создан 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 HLA стандартам.

3.3. Архитектура HLA

3.3.1. Основные функциональные компоненты HLA

Ключевыми понятиями HLA являются федерация (Federation) и RTI (Runtime Infrastructure). Федерация – это объединение компонентов имитационного моделирования, называемых федератами (federates).

В качестве федератов могут рассматриваться:

- Имитационные модели и системы имитации (по терминологии HLA - ***"созданные"*** (constructive) участники).
- Тренажёры и модели, интерактивно управляемые людьми - по терминологии HLA **- *"виртуальные"*** (virtual) участники.
- Реальные образцы техники или системы связи и управления (по терминологии HLA - ***"живые"*** (live) участники).
- Специализированное программное обеспечение, предназначенное, например, для визуализации или для сбора и обработки информации.

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

3.3.2. Компоненты HLA

Итак, технология HLA включает компоненты(рис.24), 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 HLA***, к которым относятся десять базовых правил. Они определяют основные принципы разработки программного обеспечения имитационного моделирования в среде HLA.
- Специально разработанная для поддержки HLA ***операционная система*** RunTime Interface (RTI), которая включает шесть базовых групп по управлению интерфейсом:
  - **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

3.3.3. Правила HLA

В 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 – представления 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. Обеспечение того, чтобы верные данные обеспечились в верное время и что данные используются по существу, правильным способом – ответственность федератов. 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**: HLA позволяет в любой момент времени только одному федерату изменять значение атрибута объектного образца. HLA обеспечивает также механизм динамической передачи права монопольного использования атрибута объектного образца от одного федерата к другому. 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 HLA используются различные внутренние механизмы управления временем**.**

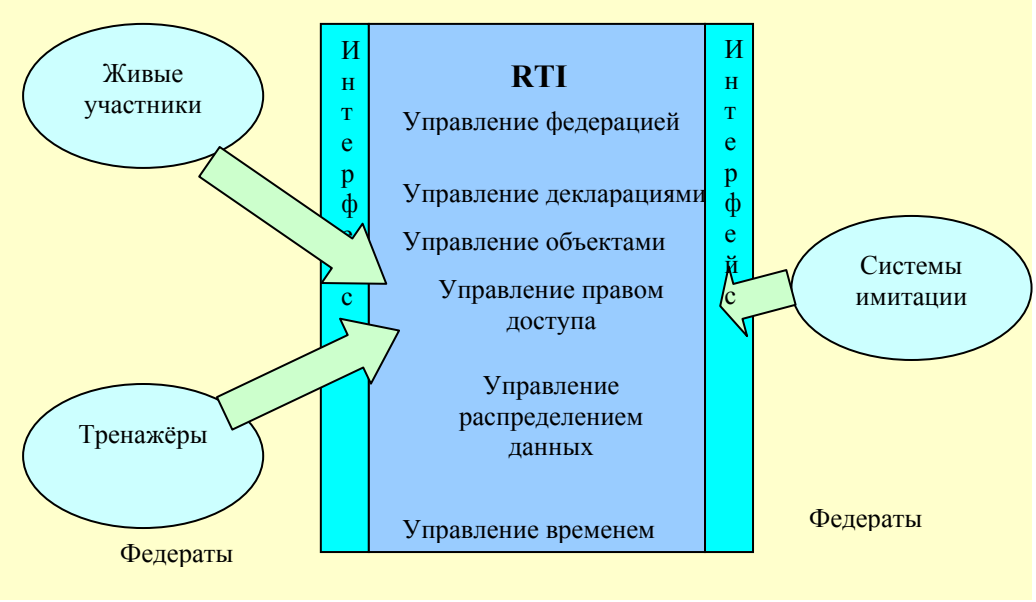


Рис. 25. Компоненты HLA

3.3.4. Шаблон объектных моделей (Object Model Template)

Стандартизированная структура (шаблон) моделей объектов HLA 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 индивидуальные моделирования, т.е. 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.

3.4. 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. А именно 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-as-possible» федерациях масштабный коэффициент берется равным «бесконечности».

3.4.1. Допущения и определения

В системах управления временем для федераций используются следующие допущения:

– 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.

– И для RTI, и для отдельных федератов допускается использование внешнего синхронизированного с некоторой точностью реального времени моделирования.

– 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.

– Федераты могут не генерировать события с временной меткой «в прошлом», то есть с временной меткой меньше текущего времени федерата.

– 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 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.

3.5. Управление временем в HLA

Управление временем в 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 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.

3.5.1. Сервисы передачи сообщений

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 HLA рассматриваются ниже.

3.5.2. Дисциплины очереди сообщений

Для сервисов управления временем в HLA основным механизмами являются механизмы упорядочивания сообщений, передаваемых федератам. Для поддержки возможности взаимодействия федератов с различными требованиями предоставляется множество сервисов. В настоящее время в HLA определено четыре механизма упорядочивания сообщений:

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

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 (first-in-first-out) 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-событийном 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 HLA одно событие каузально предшествует другому, если первое событие возникло перед вторым по данному выше определению.

Сервис каузального упорядочивания в 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.

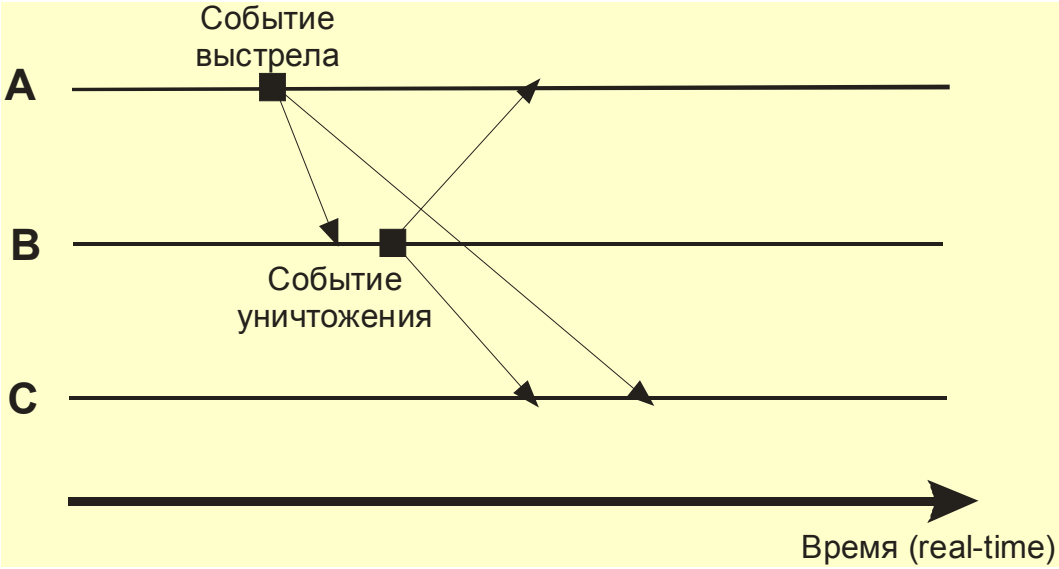


Рис. 26. Сценарий, демонстрирующий каузальное упорядочивание событий. RTI задерживает передачу первого сообщения, полученного C.

Например, рассмотрим пример, изображенный на рис.25. 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. В сценарии, изображенном на этом рисунке, сообщение об уничтожении достигнет федерата C раньше, чем сообщение о выстреле. Так как событие «выстрел» каузально предшествует событию «уничтожение», сервис каузального упорядочивания гарантирует, что событие выстрела будет передано федерату C 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 C. 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 [1].

3.5.3. 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 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 A собирается (?) поджечь первую цель, чтобы оказаться в области ее обстрела. Может получиться так, что танк B окажется в этой области раньше, чем танк C. Несмотря на это, так как обновления состояний танков B и C – это одновременные события, то каузальный порядок не дает гарантию того, что танк A получит сообщение обновления состояния танка B раньше, чем сообщение обновления от танка C. Это может вызвать неправильный поджог танка C. 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 B в момент входа в область танка A будет иметь меньшую временную метку, чем обновление состояния танка B. Таким образом сообщение танка B будет послано первым.

Скрытые зависимости

Рассмотрим пример боя во время военной кампании. Бой - это синхронизированная (рассчитанной по времени) последовательности 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 федератам. 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 (см. далее). 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.

3.5.4. 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, или
- 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.

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.

3.5.5. 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:

- 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:
  1. RTI послала все сообщения для событий категории IV системы моделирования с временной меткой меньшей или равной времени, указанном в сервисе **Time Advance Request**;
  2. нет сообщения для событий категории IV с временной меткой, меньшей или равной времени, указанном в последнем вызове сервиса **Next Event Request** с параметром «all»;
  3. 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.

Итак:

- 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.

Лекция 4. **Оптимизация времени выполнения распределённой имитационной модели.**

Введение

Балансировка нагрузки (Load Balancing) применяется для оптимизации выполнения распределённых (параллельных) вычислений с помощью распределённой (параллельной) ВС [31,32,33,34]. Балансировка нагрузки предполагает равномерную нагрузку вычислительных узлов (процессора многопроцессорной ЭВМ или компьютера в сети). При появлении новых заданий программное обеспечение, реализующее балансировку, должно принять решение о том, где (на каком вычислительном узле) следует выполнять вычисления, связанные с этим новым заданием. Кроме того, балансировка предполагает перенос (migration – миграция) части вычислений с наиболее загруженных вычислительных узлов на менее загруженные.

Балансировку необходимо выполнять и при проведении распределённого моделирования [28]. Первоначальная цель параллельного дискретно-событийного имитационного моделирования – быстрое выполнение больших и сложных моделей. В частности, PDES (Parallel Discrete Event Simulation) пытается увеличить скорость выполнения путём распределения модели на нескольких процессорах, которые работают параллельно. Однако распределение объектов моделирования оказывает большое влияние на скорость выполнения имитационного эксперимента, поскольку некоторые процессоры (или компьютеры в сети) оказываются сильно загруженными, другие слабо или вообще могут простаивать. Перераспределение нагрузки на менее загруженные процессоры (компьютеры) является выходом в этом случае.

Чаще всего в параллельном дискретно-событийном имитационном моделировании компоненты моделируемой системы представляют собой логические процессы(LP_i,$j=1\div n$) которые могут функционировать параллельно. Логические процессы распределяются между физическими процессорами, взаимодействие процессов осуществляется путём посылки сообщений от одного процесса другому. Возникает конфликт между сбалансированным распределением объектов по процессорам и низкой скоростью обменов сообщениями между процессорами. Если логические процессы распределены между процессорами таким образом, что издержки на коммуникацию между ними сведены к нулю, то некоторые процессоры (компьютеры) могут простаивать, в то время как остальные будут перегружены. В другом случае, «хорошо сбалансированная» система потребует больших затрат на коммуникацию. Следовательно, стратегия балансировки должна быть таковой, чтобы процессоры (компьютеры) были загружены достаточно равномерно, но и коммуникационная среда не должна быть перегружена.

Необходимо различать декомпозицию задач и проблему отображения задач (в нашем случае логических процессов). Декомпозиция задачи является этапом процесса создания параллельной программы. Декомпозиция предназначена для разделения приложения на модули. Модули исполняются на отдельных процессорах. В результате декомпозиции задачи появляется набор заданий, которые параллельно решают задачу. Эти задания могут быть независимыми или связанными друг с другом посредством обмена данными. Отображение (или «распределение задач») является отдельным этапом, позволяющим распределить задания, полученные на этапе декомпозиции, между процессорами.

Реализация распределённой системы имитации требует ***разработки алгоритмов синхронизации объектов (или процессов),*** функционирующих на различных узлах ВС. Алгоритмы синхронизации мы уже обсуждали во второй лекции. Эффективность реализации этих алгоритмов, в свою очередь, зависит от равномерности распределения (балансировки) вычислительной нагрузки по узлам ВС во время функционирования

распределённой программной системы, каковой является, в частности, распределённая система имитации.

4.1. Балансировка вычислительной нагрузки

4.1.1. Причины возникновения несбалансированной нагрузки

Проблема балансировки вычислительной нагрузки распределённой системы имитации возникает по той причине, что:

- структура имитационной модели неоднородна, различные её части требуют различных вычислительных мощностей;
- структура вычислительного комплекса (например, кластера), также неоднородна, т.е. разные вычислительные узлы обладают разной производительностью;
- структура межузлового взаимодействия неоднородна, т.к. линии связи, соединяющие узлы, могут иметь различные характеристики пропускной способности.

4.1.2. Статическая и динамическая балансировки

Следует различать ***статическую*** и ***динамическую*** балансировки.

Статическая балансировка выполняется ***до начала имитационного прогона***. При распределении логических процессов по процессорам *используется опыт предыдущих имитационных прогонов* (именно так происходит предварительное размещение процессов по компьютерам в SPEEDES[26]).

В подсистеме статической балансировки Triad.Net [66] анализируется структура имитационной модели. Объекты и их подобъекты (модель является иерархической) алгоритм располагает на одном вычислительном узле. Кроме того, выявляются клики в графе имитационной модели и они также отображаются на один вычислительный узел для того, чтобы сократить время на пересылку сообщений между узлами ВС.

Однако предварительное размещение логических процессов по процессорам (компьютерам) не даёт эффекта.

Это объясняется тем, что:

- Модель во время имитационного прогона может измениться (планирование новых событий, появление новых процессов, завершение работы процессов).
- Может измениться и вычислительная среда, в которой происходит моделирование (выход из строя компьютера или процессора).
- Компьютер (или процессор), на котором выполняется имитационная модель, занят ещё и другими вычислениями, вследствие этого доля работ не связанных с имитационной моделью, может возрасти.

Так или иначе, выигрыш от распределения логических процессов по компьютерам с целью выполнения параллельной обработки становится неэффективным.

Динамическая балансировка предусматривает перераспределение вычислительной нагрузки на узлы ***во время имитационного прогона***. Программное обеспечение, реализующее динамическую балансировку, определяет:

- загрузку вычислительных узлов;
- пропускную способность линий связи;
- частоту обменов сообщениями между логическими процессами имитационной модели и другие её свойства, влияющие на эффективность распределённого имитационного прогона и др..

На основании собранных данных (как о модели, так и вычислительной среде) принимается решение о переносе логических процессов с одного узла на другой.

4.1.3. Постановка задачи динамической балансировки

Цель балансировки загрузки может быть сформулирована следующим образом:

Исходя из набора задач, включающих вычисления и передачу данных, и сети компьютеров определенной топологии, найти такое распределение задач по компьютерам, которое обеспечивает примерно равную вычислительную загрузку компьютеров и минимальные затраты на передачу данных между ними.

Определим задачу балансировки более формально:

Задача балансировки ставится как задача отображения неизоморфных связных графов, B: TM → NG , где TM – множество графов моделей, NG – множество графов – конфигураций компьютерной сети. Граф G ∈ NG , G = {C, Ed}, определяется множеством вычислительных узлов C и множеством ребер Ed, обозначающих линии связи. Можно рассматривать NG как суперграф, содержащий все возможные (допустимые) графы G в качестве подграфов. Граф M ∈ TM, задает имитационную модель, M = {LPj}, j=1÷n.

.

Таким образом, множество графов моделей (в других литературных источниках [28] граф заданий) должны быть оптимальным образом отображено на множество графов компьютерной сети (графов процессоров).

4.1.4. Методология практического решения задачи балансировки

Обычно практичное и полное решение задачи балансировки загрузки состоит из четырех шагов:

- Оценка загрузки.
- Инициация балансировки загрузки.
- Принятие решений о балансировке.
- Перемещение объектов.

В последующих частях конкретизируется каждый шаг балансировки с рассмотрением различных методов решения.

Оценка загрузки

На этом этапе осуществляется приблизительная оценка загрузки каждого процессора. Полученная информация о загрузке используется в качестве базы данных для процесса балансировки, во-первых, для определения возникновения дисбаланса, во-вторых, для определения нового распределения объектов имитационной модели путем вычисления объема работ, необходимого для перемещения объектов. Отсюда, качество работы балансировки загрузки напрямую зависит от точности и полноты информации в базе данных.

В основном такая база данных состоит из ***двух типов*** данных:

- Данные о работе процессора (информация уровня процессора). Эти данные включают: загрузку процессора, время простоя процессора, фоновую загрузку процессора, скорость передачи информации по линиям связи и т.д.
- Данные о работе имитационной модели. Данные включают время выполнения объекта, время простоя, интенсивность обмена информацией и т.д.

Очень важно владеть такой информацией как ***коммуникационная модель***.

Коммуникационная модель содержит важную информацию для принятия решений о перемещении объектов при балансировке загрузки. При необходимости переместить объект с перегруженного процессора A на недогруженный процессор B целесообразно выбрать для перемещения тот объект, который наиболее интенсивно сообщается с объектами, уже расположенными на B.

Необходимо учитывать два типа связей между объектами: **двухточечные** коммуникации и **коллективные** коммуникации (широковещание, см. лекцию 1). При распределении объектов между процессорами производится оценка коммуникаций. Затраты на двухточечную связь между двумя объектами могут быть определены через объем передаваемых данных (**b** – объём сообщений в байтах) и частоту коммуникации (**n** – количество сообщений за единицу времени). Используя величины затрат процессора на каждое сообщение и каждый байт, можно оценить общие затраты на коммуникацию между двумя объектами: $\alpha * n + \beta * b$, где b – общий объем всех n сообщений.

Оценка загрузки процессора и объекта может быть произведена несколькими способами. Один из способов (**аналитический**), обычно используемый при статической балансировке загрузки и состоит в приблизительной оценке загрузки каждого объекта на основе знаний о приложении. К этим знаниям относятся:

- функция от размера объема данных, отражающая сложность алгоритма
- модель коммуникаций между объектами.

Преимущество аналитического метода состоит в том, что он достаточно точно может оценить трудоёмкость задачи с часто меняющейся загрузкой. В частности, вместо того, чтобы сообщать об изменении загрузки лишь после его возникновения, он может предсказать предстоящее изменение загрузки и отреагировать на него. Недостаток же этого метода состоит в том, что он требует огромных усилий со стороны программиста, знающего алгоритмы, и в том, что он может быть довольно неточным в случае, если модель для оценки скорости выполнения приложений неточна.

Другой способ сбора данных о загрузке состоит в измерении загрузки процессоров и задач. Большинство современных машин снабжено ***счетчиками времени*** (с точностью до микросекунд), которые могут быть использованы для измерения времени выполнения каждой задачи. Также этот метод потенциально предоставляет автоматическое решение задачи оценки стоимости загрузки. Преимущество метода состоит в том, что он является точным и не требует больших усилий программиста. К недостаткам можно отнести следующее: стратегии балансировки, основанные на этом методе (измерение) учитывают прошлое распределение нагрузок. Если загрузка задач меняется ***непредсказуемым*** образом, то метод будет неточным.

Приведенные два метода сбора данных о загрузке можно сочетать, дополняя метод, основанный на измерении производительности предсказывающей способностью аналитического метода оценки.

Инициализация балансировки загрузки

Слишком частое выполнение балансировки загрузки может привести к тому, что выполнение имитационной модели только замедлится. Затраты на саму балансировку могут превзойти возможную выгоду от ее проведения. Следовательно, для продуктивности балансировки необходимо каким-то образом определять момент ее инициализации.

Для этого следует:

- Определить момент возникновения дисбаланса загрузки.
- Определить степень необходимости балансировки путем сравнения возможной пользы от ее проведения и затрат на нее.

Дисбаланс загрузки может определяться ***синхронно*** и ***асинхронно***.

При ***синхронном*** определении дисбаланса все процессоры (компьютеры сети) прерывают работу в определенные моменты синхронизации и определяют дисбаланс загрузки путем сравнения загрузки отдельного процессора с общей средней загрузкой.

При ***асинхронном*** определении дисбаланса каждый процессор хранит историю своей загрузки. В этом случае момент синхронизации для определения степени дисбаланса отсутствует. Вычислением объема дисбаланса занимается фоновый процесс, работающий параллельно с приложением.

Принятие решений в процессе балансировки

Большинство стратегий динамической балансировки загрузки можно отнести к классу централизованных или к классу полностью распределенных.

При ***централизованной стратегии*** специальный компьютер собирает глобальную информацию о состоянии всей вычислительной системы и принимает решение о перемещении задач для каждого из компьютеров.

При ***полностью распределенной стратегии*** на каждом процессоре выполняется алгоритм балансировки загрузки, обменивающийся информацией о состоянии с другими процессорами. Перемещение происходит только между соседними процессорами.

Перемещение объектов

После принятия решений о балансировке происходит перемещение объектов среди процессоров для достижения нового баланса загрузки. При перемещении объекта должна обеспечиваться целостность его состояния. Для перемещения данных объекта обычно используются вспомогательные функции приложения, особенно когда речь идет о сложных структурах данных, таких как списки ссылок или указателей.

Архитектура подсистемы балансировки

Из всего вышесказанного можно сделать вывод о том, что для проведения балансировки во время имитационного моделирования необходимо разработать специальное программное обеспечение. Это программное обеспечение включает:

- Программные средства, обеспечивающие оценку состояния распределённой имитационной модели и вычислительной среды (подсистема анализа).
- Управляющую программу, принимающую решение о моменте проведения балансировки, и о том, какие логические процессы следует переместить с одного процессора на другой.
- Программные средства, реализующие перемещение объекта с процессора на процессор.
- Подсистему визуализации, отображающую распределение компонентов имитационного моделирования по вычислительным узлам, коммуникационную среду, изменение состояния имитационной модели и вычислительной среды.
- Базу данных, которая хранит информацию о компонентах имитационной модели и о вычислительной среде.

Разработаны ряд стратегий для балансировки PDES. Существует большое количество подходов к решению этой проблемы, реализованы алгоритмы балансировки и программные средства для его выполнения.

4.2. Пример алгоритма динамической балансировки в SPEEDES

В большинстве случаев алгоритм балансировки разрабатывается для конкретного класса задач. В других случаях балансировка предполагает большие изменения в коде авторских программ. В SPEEDES алгоритм балансировки предполагает, что программные средства для моделирования охватывают самые разнообразные области знаний, а вносимые изменения в код программы пользователя тоже незначительны.

Алгоритм динамической балансировки в SPEEDES предполагает пересылку (перенос, миграцию) объектов с процессора на процессор во время моделирования.

В SPEEDES рассматривается универсальная балансировка, которую можно применить к задачам различных типов, причём требуется лишь незначительное изменение пользовательского кода.

SPEEDES - это объектно-ориентированное программное обеспечение, реализующее дискретное моделирование, управляемое событиями. Поскольку данная система была

разработана для распределённого моделирования, оно реализует три стратегии синхронизации (TIME WARP, Breathing Time Buckets, и Breathing Time Warp). Стратегии синхронизации могут быть выбраны пользователем при выполнении имитационного прогона.

В SPEEDES пользователь полностью отвечает за отображение логических процессов на компьютеры. Поскольку пользователю предоставлена полная свобода в выборе отображения, он априори не знает, как решать данную задачу в конкретном случае. В общем случае, построение оптимального отображения достаточно сложная задача, поэтому автоматизированное решение системой данной задачи значительно бы увеличило ее привлекательность для пользователей.

Статическая балансировка не может дать хороших результатов, поскольку характеристики моделирования могут периодически меняться (об этом уже говорилось ранее).

4.2.1. Динамическая балансировка и перенос нагрузки

Алгоритм динамической балансировки использует характеристики состояния системы и принимает решение о том, с какого компьютера и на какой следует перенести работу во время моделирования. Это подход позволяет реагировать на изменение состояния вычислительной машины или моделируемой системы и выполнить балансировку, если время, затраченное на имитационный прогон, растёт. Однако динамическая балансировка влечёт за собой дополнительные расходы на сбор статистических данных о состоянии вычислительной среды и модели, на анализ данных и на принятие решений

Перенос нагрузки (или ***перенос объекта*** или ***процесса)***, - это механизм, который используется для переноса работы с одного компьютера на другой. Если балансировка связана с равномерной загрузкой процессоров, то перенос нагрузки связан с перемещением части работы на другой компьютер, а балансировка является целью переноса нагрузки.

Балансировку и перенос нагрузки используют для повышения производительности распределённой системы моделирования. Из-за разнородности вычислительной среды, один алгоритм может хорошо работать в распределённой системе и плохо в другой. RCL - стратегия переноса нагрузки

Рассмотрим три алгоритма динамического переноса нагрузки, предложенные разработчиками SPEEDES:

- случайный алгоритм (random,R);
- алгоритм, основанный на коммуникациях (communication,C);
- алгоритм, основанный на вычислении нагрузки (load,L).

Будем в дальнейшем называть эти алгоритмы балансировки RCL-стратегией.

Алгоритм переноса объекта или процесса с одного компьютера достаточно сложен. Введение некоторых разумных допущений только незначительно снижает сложность этого алгоритма. Итак, сделаны следующие допущения:

- Распределённая система является гетерогенной.
- Пользователь может интерактивно взаимодействовать с машиной в любой момент времени.
- Количество вычислительных узлов в сети ограничено.
- Возможно изменение сети.
- Задержки на коммуникацию в сети остаются неизменными.
- Издержки на перенос нагрузки с одного на другой процессор соизмерим с физическими размерами этой нагрузки.

RCL стратегия использует двухуровневый процесс принятия решений, который объединяет ***централизованный*** и ***децентрализованный*** подходы. Двухуровневый процесс принятия решений предполагает:

– Центральный процесс-координатор принимает решение о переносе нагрузки,
– Каждый отсылающий сообщения процесс ответственен за выбор нагрузки, которую следует перенести.

На первом уровне выбирается центральный координатор для того, чтобы среди всех рабочих станций именно он мог принимать решения. Во время процесса переноса нагрузки все рабочие станции приостанавливают свою работу на некоторый промежуток времени, пока центральный координатор собирает информацию о загрузке процессоров. Координатор анализирует информацию, которая включает сведения о загрузке компьютеров и о распределении нагрузки в системе. Координатор, на основании этой информации решает, следует ли предпринимать действия по переносу нагрузки с одного компьютера на другой. Если состояние системы таково, что перенос нагрузки необходим (состояние соответствует критерию переноса нагрузки), центральный координатор начинает процедуру перераспределения нагрузки, руководствуясь скоростью работы компьютеров.

Если в переносе нагрузки нет необходимости, компьютер должен разослать сообщение об этом всем другим компьютерам. В этом случае действия второго уровня игнорируются, и каждый компьютер продолжает обрабатывать свои локальные события. Если, находясь на втором уровне, посылающие данные рабочие станции получат от центрального координатора сигнал о необходимости миграции нагрузки, они начинают процесс выбора части нагрузки для переноса её на другой хост. При выборе используют RCL стратегию. После того, как части нагрузки выбраны на всех пересылающих данные рабочих станциях, начинается процесс миграции. Последовательность шагов, предпринимаемых при миграции нагрузки между двумя компьютерами, включает упаковку данных и их отправку с посылающего компьютера, и приём и распаковку на принимающем компьютере. В конце каждого шага миграции принимающий компьютер рассылает всем остальным сообщение о завершении процедуры миграции и о новом размещении данных. После этого каждый компьютер продолжает обработку. Рассмотрим более детально действия на каждом уровне.

Действия первого уровня

В начале действий по переносу нагрузки информации все компьютеры прекращают свою работу, и каждый получает информацию о локальной нагрузке в текущий момент времени. Информация о локальной нагрузке включает:

– Количество нагрузки (Ld).
– Количество времени, потраченное процессором на обработку приложений SPEEDES (TAppS).
– Количество нагрузки, которая была обработана с момента последнего переноса(LdM).

Как только координатор получает информацию от всех компьютеров сети, он начинает анализировать общую информацию о нагрузке на компьютеры. Координатор вычисляет следующие характеристики:

– Дисперсию коэффициента загрузки компьютера (VarCLd);
– Общее количество нагрузки, ожидающей обслуживания (TotalWLd);
– Дисперсию нагрузки, ожидающей обслуживания(VarWLd).

Дисперсия коэффициенты загрузки компьютера может свидетельствовать о том, насколько они загружены. Действительно, если какие-либо компьютеры выполняют несколько приложений вместе с приложениями SPEEDES в то время, как другие простаивают, значение дисперсии может быть велико. Чем больше величина дисперсии использования компьютера, тем менее эффективно используются ресурсы процессоров.

Общее количество нагрузки, ожидающей обслуживания – это сумма всех невыполненных работ на каждом из компьютеров. Если их количество мало, то выигрыша

от балансировки может и не быть, поскольку накладные расходы на перенос нагрузки могут превысить выигрыш от балансировки. Это особенно ярко выражено, если нагрузка невелика.

Дисперсия нагрузки, ожидающей обслуживания(VarWLd). Дисперсия нагрузки, ожидающей обслуживания вычисляется для каждого компьютера. Большое значение этого показателя свидетельствует об отсутствии равномерной нагрузки на компьютеры. Этот показатель используется в качестве вторичного критерия, если дисперсия общей нагрузки ниже порогового значения (T). Указанную метрику (VarWLd) используют в процедуре (назовём её *Decide()*), которая выполняется перед процедурой переноса (*Migrate()*). Метрика нужна для принятия решения о переносе нагрузки.

На первом шаге процедуры *Decide()* выполняют сравнение между общим количеством ожидающей обслуживания нагрузки(TotalWLd) и указанным пороговым значением(T). Если значение TotalWLd выше, то вычисляется коэффициент загруженности компьютера. Если дисперсия коэффициента загрузки выше соответствующего порога (VarCLd > TLd), то выполняется процедура (Migrate()). В противном случае Migrate() на этом шаге не выполняется. В конце выполнения процедуры Decide(), центральный координатор посылает сообщение всем компьютерам о том, что они могут продолжить обработку своих локальных нагрузок, если в процедуре Migrate() нет необходимости.

Если необходимость в процедуре Migrate() есть, центральный координатор посылает сообщение о том, что ожидается выполнение этой процедуры. Далее происходит определение посылающего и принимающего компьютеров(Csender и Creceiver). Опираясь на показатель скорости обработки, (например, отношение количества нагрузки, обработанной на локальном компьютере к общему количеству обработанной нагрузки на всех процессорах со времени последнего переноса.) центральный координатор перераспределяет оставшуюся необработанную нагрузку. Например, если компьютер обрабатывал 10% общего количества нагрузки после последнего выполнения Migrate(), он возьмёт 10% процентов ещё необработанной нагрузки. Следовательно, те рабочие станции, которые обработали большее количество нагрузки, получат опять большее её количество. Если слишком большой нагрузки нет и скорость работы компьютеров остаётся неизменной, все компьютеры закончат свою работу примерно в одно и то же время.

Центральный координатор отмечает отложенные действия компьютеров – посылка, получение или отсутствие действия. Получающий компьютер будет проинформирован о количестве нагрузки, которую ему передадут, и идентификаторы посылающих компьютеров.

И, наконец, центральный координатор выполняет подготовку к переносу объектов, если этот перенос необходим. Таким образом, процедуры второго уровня выполняются лишь при необходимости выполнения процедуры Migrate().

Действия второго уровня

Действия второго уровня охватывают все рабочие станции распределённой системы. Конкретной количество нагрузки посылается с одной рабочей станции на другую. Основные действия связаны с выбором нагрузки, её упаковкой, инициализацией переноса, переносом нагрузки, распаковкой, изменением глобальной информации. Каждый логический процесс в SPEEDES состоит из количества объектов моделирования (N) и очереди событий (e_i, $i=1\div n$). События помещаются в очередь событий в неубывающем порядке их временных меток.

Нагрузка определяется как вся работа, которая должна быть выполнена для обработки запланированных событий, находящихся в очереди. Поскольку SPEEDES должна модифицировать нагрузку для переноса её части, каждому типу событий назначается коэффициент сложности (k_i) в интервале от 1 до 10. Пользователь должен

назначить коэффициент сложности для каждого определённого им события. Чем ближе коэффициент сложности к 10, тем больше времени требуется на обработку события. Нагрузка, ожидающая обслуживания, вычисляется исходя из количества событий, находящихся в очереди и их коэффициентов сложности ($LocalWLd = \Sigma e_i * k_{i,}\ i=1 \div n$).

К примеру, предположим, что моделируемый объект содержит два события типа 1 и два события типа 2 в очереди событий. Если коэффициент сложности равен 4 для события типа 1и 8 для события типа 2, то нагрузка может быть определена следующим образом: (2*4)+(3*8)= 32. Перенос нагрузки предполагает перенос объектов с одного компьютера на другой. Для выбора объектов, которые следует перенести на другой компьютер, используют один из трёх алгоритмов. Это случайный алгоритм, основанный на коммуникациях и основанный на вычислении нагрузки.

Случайный алгоритм

Случайный алгоритм (R) заключается в случайном выборе объектов моделирования на посылающем компьютере (Csender). Выбор продолжается до тех пор, пока количество выбранных объектов не будет соответствовать заданному числу.

К преимуществам использования случайного алгоритма следует отнести: лёгкость реализации, небольшие накладные расходы и сравнительно небольшое время выбор объектов для переноса.

Алгоритм, основанный на коммуникациях

Посылающий компьютер выбирает локальные объекты, которые наиболее часто обмениваются информацией с принимающим компьютером. Этот подход может сократить время на коммуникацию между двумя логическими процессами. Но он требует больших накладных расходов, поскольку на каждом компьютере должна быть таблица коммуникаций, в которой отмечается частота обменов каждого объекта, находящегося на Csender с другими компьютерами. Таблица должна всё время обновляться. Кроме того, для выбора объекта с целью его переноса следует использовать алгоритмы сортировки и поиска. Известно, что алгоритмы линейного поиска и сортировки при большом количестве объектов время выполнения этих алгоритмов может быть значительным. .

Алгоритм, основанный на вычислении нагрузки

Алгоритм, основанный на вычислении нагрузки, пытается минимизировать количество выбранных для миграции объектов. Во время миграции объекты моделирования сортируются в зависимости от их нагрузки (количество событий каждого типа, умноженное на коэффициент их сложности). Первым выбирают объект с максимальной нагрузкой.

Алгоритм, основанный на вычислении нагрузки, предпочтительнее алгоритма, основанного на коммуникациях, поскольку требует меньших временных затрат.

Итак, перед тем, как начать имитационный эксперимент, пользователь выбирает один из алгоритмов переноса нагрузки. По окончании шага выбора объектов, посылающий компьютер уже имеет список объектов для пересылки.

4.2.2. Реализация

Стратегия динамического переноса нагрузки RCL была разработана для SPEEDES с целью повышения её производительности. Были проведены эксперименты для выявления конкретных параметров, которые влияют на скорость выполнения имитационного эксперимента. В качестве такого параметра может быть рассмотрен интервал между переносами нагрузки (между процедурами Migrate()). SPEEDES поддерживает несколько синхронизирующих алгоритмов для выполнения распределённого моделирования: Breathing Time Warp (BTW), который совмещает черты алгоритма деформации времени Time Warp и протокола Breathing Time Buckets.

В результате исследований было обнаружено, что процедуру Migrate() следует выполнять в конце цикла GVT (Global Virtual Time). Действительно, события удаляются из системы во время вычисления GVT. Поэтому нет опасности, что возникнет необходимость в откате, который затронет только что перенесённую нагрузку. Более того, необходимо перенести только события и переменные мигрирующих объектов, а данные, связанные с откатом, не надо переносить, поскольку очередь откатов очищается в конце цикла GVT.

Каждый из алгоритмов был протестирован на трёх наборах входных данных. При этом варьировалась внешняя нагрузка. Результаты экспериментов показали, что процедуры миграции могут дать положительный эффект в различных условиях моделирования – при различных нагрузках.

Результаты можно интерпретировать следующим образом:

- Для всех трёх стратегий время выполнения сокращается, если интервал между процедурами Migrate() невелик.
- Частота выполнения процедуры Migrate() влияет на время моделирования существенным образом.
- При сравнении параллельного выполнения с переносом нагрузки и без него видно, что процедура переноса существенно влияет на скорость выполнения эксперимента.
- Стратегия, основанная на вычислении нагрузки, стабильно даёт возможность сократить время моделирования.

Лекция 5. **Валидация и верификация имитационной модели**

Введение

Качество информации является одним из важнейших параметров для потребителя информации. Оно определяется следующими характеристиками:

- **репрезентативность** – правильность отбора информации в целях адекватного отражения источника информации.
- **содержательность** – семантическая емкость информации.
- **достаточность** (полнота) – минимальный, но достаточный состав данных для достижения целей, которые преследует потребитель информации.
- **доступность** – простота (или возможность) выполнения процедур получения и преобразования информации.
- **актуальность** – зависит от динамики изменения характеристик информации и определяется сохранением ценности информации для пользователя в момент ее использования.
- **своевременность** – поступление не позже заранее назначенного срока.
- **точность** – степень близости информации к реальному состоянию источника информации.
- **достоверность** – свойство информации отражать источник информации с необходимой точностью.
- **устойчивость** – способность информации реагировать на изменения исходных данных без нарушения необходимой точности.

Вопросы получения качественной информации в результате имитационного эксперимента встают и перед специалистами в области имитационного моделирования. Имитационные модели получают все большее применение в процессе решения задач и принятия решений. В том, что модель и полученные с ее помощью результаты являются верными, в полной мере заинтересованы как разработчики модели, ее пользователи и лица, принимающие решения, так и люди, на которых оказывают влияние решения, принятые на основе данной модели. Эта заинтересованность в первую очередь относится к верификации и валидации модели. Под ***верификацией*** чаще всего понимают проверку правильности преобразования концептуальной имитационной модели в программную модель, под ***валидацией*** – проверку правильности её поведения и представления концептуальной модели. В Министерстве Обороны США широко применяются имитационные модели. В последние годы Министерство Обороны проявляет интерес к верификации, валидации и концепции, известной как аккредитация (VV&A- Validation, Veryfication and Accreditation). ***Аккредитация*** определяется как «официальное засвидетельствование того, что модель, симуляция, или объединение моделей и симуляций является допустимым для использования для определенной цели». *Аккредитация* – это официальное свидетельство (спонсора проекта) того, что имитационная модель применима для данной задачи. Министерство Обороны поддержало концепцию аккредитации, так как кто-то должен нести ответственность за принятие решения о возможности использования модели для данной задачи - от этого зависит большие суммы денег и жизни людей.

По одному из принципов тестирования полное тестирование систем имитационного моделирование невозможно (Balci). Исчерпывающее (полное) тестирование требует тестирования систем имитационного моделирования при всех возможных значениях входных параметров. Комбинации возможных значений входных параметров для систем имитационного моделирования в ходе исполнения программы могут привести к миллионам логических цепочек. Но в силу временных и денежных ограничений

тестирование правильности такого большого количества логических цепочек невозможно. Поэтому можно сказать, что «единственный существующий способ исчерпывающего тестирования – это тестирование до тех пор, пока тестеры не исчерпают все свои силы». Следовательно, ***целью тестирования*** систем имитационного моделирования является ***увеличение уверенности в правильности*** системы в той мере, как это диктуется планируемым использованием системы и целями проекта, а не попытка полного тестирования системы имитационного моделирования.

Несмотря на то, что существуют более 100 методов верификации и валидации [Balci,1998], в связи с временными и ресурсными ограничениями, для тестирования систем имитационного моделирования используется только очень ограниченный набор методов. Ограниченное тестирование не позволяет доказать достаточную точность систем имитационного моделирования. Поскольку использование только лишь валидации имеет хорошо известные ограничения, некоторые исследователи предлагают использовать ***оценку правильности*** вместе с ***характеристиками качества имитационных моделей***[82].

Вопросам достоверности имитационных моделей посвящено много работ, как в нашей стране, так и за рубежом. Исследования в этой области проводились очень давно. Свидетельством тому служат труды Р.Шеннона, Н.П.Бусленко, И.Н.Коваленко и т.д.[67,68,69,70,71]. И в настоящее время этим проблемам посвящено много исследований. Это исследования О. Балси, Р.Сарджента, А.Лоу и Д.Кельтона, [111,112], Б.В. Соколова, Г. Яцкив, и т.д.[113,114, 116]

5.1. Этапы имитационного моделирования

Процесс построения имитационных моделей представляет собой последовательное выполнение этапов имитационного моделирования. Эти этапы процесса моделирования приведены в книге А.Прицкера[58]:

- ***Формулирование проблемы***-описание исследуемой проблемы и определение целей исследования.
- ***Разработка модели*** - логико-математическое описание моделируемой системы в соответствии с формулировкой проблем.
- ***Подготовка данных -*** идентификация, спецификация данных.
- ***Трансляция модели -*** перевод модели на язык, приемлемый для используемой ЭВМ
- ***Верификация модели -*** Установление правильности машинных программ
- ***Валидация модели*** - оценка требуемой точности и соответствия имитационной модели реальной системе.
- ***Стратегическое и тактическое планирование*** - определение условии проведения машинного эксперимента с имитационной моделью.
- ***Экспериментирование*** - прогон имитационной модели на ЭВМ для получения требуемой информации.
- ***Анализ результатов*** - изучение результатов имеет моделирование для подготовки выводов, для решения проблемы.

Названные выше этапы имитационного исследования редко выполняются в строго заданной последовательности, начиная с определения проблемы и кончая документированием. В ходе имитационного исследования могут быть сбои в прогонах модели, ошибочные допущения, от которых в последствии 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.

Рассмотрим более подробно этапы верификации и валидации имитационной модели. Упрощенный процесс разработки имитационной модели[80] приведён на рис. 27.

Рис. 27. Упрощенный вариант процесса моделирования

Рассмотрим более подробно этапы верификации и валидации имитационной модели. Эти этапы связаны с оценкой функционирования имитационной модели. На этапе верификации определяется, соответствует ли ***запрограммированная*** модель замыслу разработчика. ***Установление адекватности*** имитационной модели выполняется на этапе валидации. Валидация модели обычно выполняется на различных уровнях (например, на уровне входных данных, элементов модели, подсистем и их взаимосвязи). Проверка адекватности модели включает сравнение её структуры со структурой системы, сравнения того, как реализованы элементарные функции и рушения в модели и системе. Существуют специальные методы валидации (например, путём оценивания чувствительности выходных данных к изменению значений входных), различные парадигмы, подходы и методики. Рассмотрим некоторые из них. Но прежде постараемся дать основные определения, а именно, более подробно рассмотрим различные подходы к валидации, а затем и алгоритм построения валидной модели, предложенный Лоу.

5.2. Валидация

Итак,. *валидация* – это *процесс* определения того, является ли имитационная модель точным представлением данной системы *для конкретной задачи.* 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*. Например, валидация модели смежного банка относительно проста, эту модель смежного банка можно хорошо изучить. А вот полная валидация модели системы

морского оружия в 2025 году фактически невозможна ввиду того что, неизвестно ни место проведения сражения, ни оружие противника. Так же обычно для построения и валидации модели можно собирать данные о существующей системе.

– Имитационная модель сложной системы может быть лишь *аппроксимацией* реальной системы, не зависимо от того, как много времени и средств потрачено на ее создание. Не существует абсолютно точных моделей, как бы того не хотелось. Модель – это абстракция, упрощение реальной системы. Чем больше времени (а следовательно и финансовых затрат) тратится на разработку модели, тем более валидная в целом будет модель. Но наиболее валидная модель не обязательно является и наиболее выгодной. Например, так как для улучшения валидности модели до определенного уровня может потребоваться сбор подробных данных, то такое улучшение может быть достаточно затратным. Но в то же время такое улучшение валидности может и не привести к принятию решений, которые значительно лучше существующего.
– Имитационная 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 имитационной модели, при условии наличия времени и средств. К сожалению, на практике эта рекомендация не всегда выполняется.
– Каждый раз, когда имитационная модель применяется к другой задаче, необходимо перепроверять валидность данной модели. Данная задача может существенно отличаться от первоначальной, либо параметры модели могут измениться.

Имитационная модель и результаты ее выполнения *надежны*, если лицо, принимающее решение и другие ведущие специалисты проекта приняли ее как «точная». Заметим, что надежная модель не всегда является валидной, и наоборот, валидная модель не всегда является надежной. Для упрощения установления надежности модели необходимо следующее:

– Понимание и принятие принимающим решение лицом допущений модели.
– Демонстрация того, что была осуществлена валидация и верификация модели (то есть того, что программа отлажена).
– Вовлеченность и ответственность за проект лица, принимающего решения.
– Репутация разработчиков модели.
– Убедительная анимация.

5.3. Подход к управлению успешным исследованием системы методами имитационного моделирования

На рис.28 представлены этапы построения имитационной модели (они уже были приведены ранее). Далее более подробно рассматриваются рекомендации А. Лоу[74] по проведению каждого из этапов. Приведённая ниже информация используется А. Лоу при чтении курса лекций.

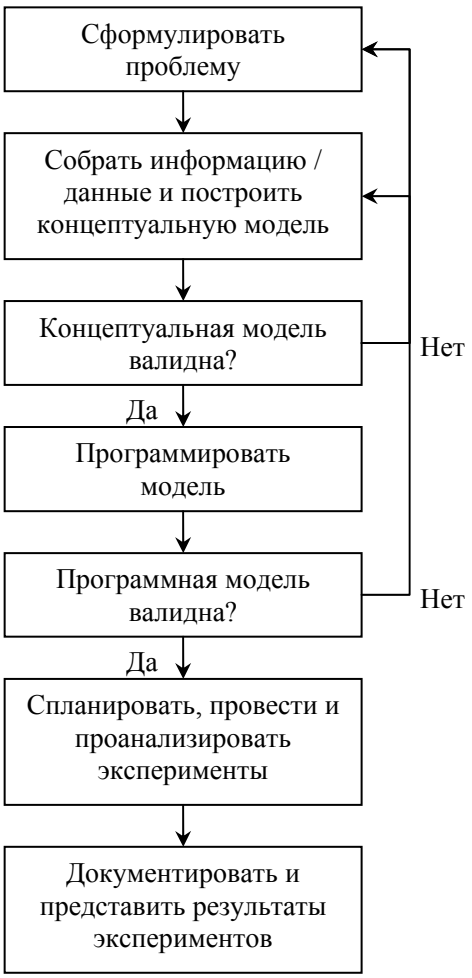


Рис. 28. Этапы исследования имитационной модели

Далее приводится описание действий на каждом из семи шагов.

Шаг 1. Формулировка задачи

- Задача формулируется лицом, принимающим решение
  - Задача может быть сформулирована нечетко либо только на качественном уровне.
  - Обычно задача формулируется итеративно.
- Организационное совещание таких проектов возглавляются руководителем проекта, в присутствие аналитика в области имитационного моделирования и эксперта в данной предметной области. На собрании обсуждаются следующие положения:
  - Общие цели исследования.
  - Специфические вопросы, на которые необходимо ответить во время исследования (без такой специфики невозможно определить необходимый уровень детализации).
  - Критерии качества, используемые для определения эффективности различных конфигураций системы.
  - Размеры системы.
  - Моделируемая конфигурация системы.
  - Изучаемый временной кадр и необходимый ресурсы (люди, компьютеры и т.д.)

Шаг 2. Сбор данных и создание концептуальной модели

- Сбор информации о макете системы и способе эксплуатации.

- Сбор данных для определения параметров модели и распределении вероятностей (например, для времени отказов и времени восстановления машины).
- Документация допущений модели, алгоритмов, краткое изложение данных на письменной *концептуальной модели.*
- Уровень детализации модели должен зависеть от следующего:
    - Цели проекта
    - Критерий решения задачи
    - Доступность данных
    - Технические ограничения
    - Мнения экспертов в данной предметной области
    - Временные и финансовые ограничения
    - Между моделью и системой не должно быть соотношения один-к-одному.
    - Степень достоверности.
    - Сбор данных о рабочих характеристиках (выходных) на основе существующей системы (если таковая существует) для последующей валидации модели на шаге 5.

Шаг 3. Определение валидности концептуальной модели

- Структурированный просмотр концептуальной модели в присутствии руководителя проекта, аналитика и эксперта. Этот просмотр называется *валидацией концептуальной модели.*
- Если в концептуальной модели выявлены ошибки или упущения, которые есть практически всегда, то до того как приступить к этапу программирования необходимо обновить модель.

Шаг 4. Программирование модели

- Программирование модели на коммерческих пакетах для имитационного моделирования или на универсальных языках программирования (например, C, C++ или Java).
- Проверка (откладка) программы.

Шаг 5. Определение валидности запрограммированной модели

- При наличии реальной системы необходимо сравнить выходные данные имитационной модели с соответствующими выходными данными реальной модели (см. шаг 2). Этот процесс называется *валидацией результатов*.
- Независимо от того, существует ли реальная система или нет, аналитик по имитационному моделированию и эксперт в данной предметной области должны просмотреть результаты моделирования на корректность. Если результаты согласуются с тем, какими они должны быть в реальной системе, то говорят, что имитационная модель имеет *внешнюю (лицевую) валидность.*
- Для определения параметров модели, более всего влияющих на критерии качества, необходимо произвести анализ чувствительности. Полученные параметры требуют более тщательного моделирования.

Шаг 6. Проектирование, управление и анализ экспериментов

- Для каждой исследуемой конфигурации системы необходимо выбрать временные параметры (выходы) такие как время работы, время разогрева системы и количество независимых репликаций модели.
- Проанализировать результаты и решить, нужны ли дополнительные эксперименты.

Шаг 7. Документирование и представление результатов моделирования

- Документация модели (и связанных с ней исследования) должна включать в себя концептуальную модель (необходима для дальнейшего переиспользования модели), детальное описание программы и результаты данного исследования.
- Для повышения надежности модели окончательное представление исследования должно включать в себя анимацию и описание обсуждений процесса построения/валидации модели.

5.4. Методы разработки валидных и надежных моделей

В данной главе представлены *практические* методы разработки валидных и надежных моделей. В конце названия каждой подглавы в квадратных скобках указаны номера шагов, в которых (как минимум) должен быть применен рассматриваемый метод.

5.4.1. Точное формулирование задачи [шаг 1]

Необходимо точно сформулировать исследуемую проблему. Для этого необходимо сформулировать все решаемые задачи, список специфических вопросов, на которые должна ответить модель, а также критерии для оценки эффективности конфигураций системы. Без окончательных формулировок специфических вопросов невозможно выбрать необходимый уровень детализации модели. Критерии также должны быть четко сформулированы, так как для различных критериев требуется различный уровень детализации [58].

Когда лицо, принимающее решения, начинает исследование, решаемая задача еще четко не сформулирована или даже не определена. Таким образом, во время исследования, когда появляется более глубокое понимание проблемы, эта информация должна передаваться лицу, принимающему решения. После этого он может переформулировать проблему.

5.4.2. Проведение интервью с экспертом в данной предметной области [1, 2]

Никогда не существует одного человека, знающего всю информацию, необходимую для создания имитационной модели. Поэтому для глубокого понимания моделируемой системы аналитику необходимо провести переговоры со многими различными экспертами в данной предметной области. Заметим, что некоторые данные, предоставленные экспертами, постоянно будут некорректными. Если какая-то часть системы особенно важна, то необходимо провести интервью, по крайней мере, с двумя экспертами. В части 2.6 обсуждается методика, которая позволяет гарантировать, что допущения, принятые в модели, верны. Эта методика также помогает разрешать противоречия между мнениями разных экспертов.

5.4.3. Постоянное взаимодействие с лицом, принимающим решения [1-7]

Один из основных принципов разработки валидной и надежной модели – постоянное взаимодействие аналитика и лица, принимающего решения, и других членов проектной группы. Этот подход имеет следующие основные преимущества:

- Он помогает обеспечить корректность решаемой проблемы.
- Необязательно изначально знать точную структуру системы.
- Во время исследования лицо, принимающее решение, может менять задачи.

– Необходимо поддерживается заинтересованность и вовлеченность в исследование лица, принимающего решения.
– Так как лицо, принимающее решения, понимает и принимает допущения модели, то сама модель получается более надежной.

Пример 2. Один военный аналитик работал над созданием имитационной модели в течение нескольких месяцев без взаимодействия с генералом, заказавшим данную разработку. После пяти минут предоставления окончательной формулировки задачи в Пентагоне генерал вышел со словами: «Эта не та задача, которая меня интересует».

5.4.4. Использование количественных методов для валидации компонентов модели [2]

Каждый раз, когда возможна проверка валидности различных компонентов всей системы, аналитик должен использовать количественные методы. Далее приводятся примеры таких методов.

Если данные наблюдения имеют какое-либо теоретическое распределение вероятностей (например, экспоненциальное или нормальное), то адекватность их представления может быть оценена с помощью графика и критерия согласия.

Как обсуждается ниже, для построения модели важно использовать соответствующие данные. Однако также важно и тщательно структурировать эти данные. Например, для исследования нескольких наборов данных на однородность распределения при их объединении можно использовать тест Крускала-Валиса на однородность популяции. Если наборы данных однородны, то их можно объединить и объединенные набор данных можно использовать для построения имитационной модели.

Пример 3. Рассмотрим производственную систему. Предположим для нее были собраны данные об отказах и восстановлении системы после отказов для двух «идентичных» машин одного производителя. Тем не менее, тест Крускала-Валиса показал, что распределения этих двух наборов данных различны. Поэтому в имитационной модели для этих машин были использованы разные законы распределения.

5.4.5. Документирование концептуальной модели [2]

Основной причиной некорректных допущений в модели является ошибки при проведении интервью. Документация концепций, допущений, алгоритмов и кратких данных может уменьшить эту проблему, а так же увеличит надежность системы. Модель (концептуальная) не должна быть точным описанием работы системы. Она должна описывать работу системы относительно решаемых моделью задач. Это описание является основной документацией модели. Оно должно быть понятным для аналитика, эксперта в данной предметной области и технического директора. В концептуальную модель должно входить следующее:

– Обзор обсуждаемых целей проекта, специфичных задач, решаемых моделью, и основные критерии.
– Диаграммы операций и схема системы
– Подробное описание всех подсистемы (*в виде списка* для наглядности) и их взаимодействия
– Какие упрощения были сделаны и почему
– Краткое описание входных данных модели (для того, чтобы упростить чтение модели лицом, принимающим решение, технический анализ следует поместить в приложение)
– Источники важных или спорных данных

Детализация концептуальной модели должна быть достаточной для создания имитационной программы (на шаге 4).

5.4.6. Структурированный просмотр концептуальной модели [3]

Как говорилось выше, аналитику необходимо собирать данные от разных экспертов. Кроме того, эксперты обычно очень заняты своей непосредственной работой. Вследствие чего они не уделяют требуемого внимания вопросам, поставленным аналитиком. В результате существует большая вероятность того, что аналитик не получит полного и корректного описания системы. Эффективным способом решения этих потенциальных проблем является проведение *структурированного просмотра концептуальной модели* в присутствие экспертов и лиц, принимающих решения. С помощью проектора аналитик представляет концептуальную модель по пунктам. При этом аналитик не переходит к следующему пункту, пока все присутствующие не убедятся в правильности данного пункта на выбранном уровне детализации. Структурированный просмотр увеличивает и валидность, и надежность имитационной модели. (Как уже говорилось выше, этот этап называется валидацией концептуальной модели.)

В идеальном случае структурированный просмотр должен быть выездным (например, в конференц-зале отеля), так чтобы участники могли полностью сосредоточиться на рассматриваемой проблеме. Более того, если во время просмотра не были рассмотрены основные вопросы, то просмотр должен проводится до начала этапа программирования. До начала просмотра необходимо раздать описание концептуальной модели всем участникам просмотра с последующим получением их комментариев. Однако, не предполагается, что это заменяет сам структурный просмотр, так как у участников может не быть времени и стимулов самостоятельно тщательно просматривать документацию. К тому же, необходимо взаимодействие во время проведение реального совещания.

Пример 4. Во время структурного просмотра транспортной системы, по мнению экспертов, оказались неверными основные процентные соотношения, предоставленные спонсорами. (Из-за удаленности главных офисов спонсоров и экспертов, эксперты не могли присутствовать на организационном совещании по проекту.) В результате для сбора данных о различных частях системы были выбраны разные люди. Собранные данные были использованы для обновления концептуальной модели. После чего второй просмотр был удачным.

5.4.7. Использование анализа чувствительности для определения важных параметров модели [5]

Анализ чувствительности – это метод определения параметров модели, оказывающих наибольшее влияние на критерий оценки работы. Если определенный параметр оказывается важным, то его следует более тщательно рассматривать. С помощью анализа чувствительности можно рассматривать следующее:

- Значение параметра (см. пример 5)
- Выбор распределения вероятностей (закона распределения).
- Объекты, проходящие через моделируемую систему.
- Уровень детализации подсистем.

Пример 5. При исследовании новой системы после разговора с экспертами значение вероятности было принято равным 0,75. Важность «абсолютно» точного значения этой вероятности можно определить прогоном этой имитационной системы с параметром 0,75 и, например, с 0,7 и 0,8. Если результаты этих прогонов будут примерно одинаковы, то выходные данные не чувствительны к выбору значения

вероятности в диапазоне от 0,7 до 0,8. Иначе необходимо более точное значение вероятности. (Собственно говоря, для определения влияния значения вероятности на результаты моделирования, необходимо выполнить несколько независимых репликаций модели с различными случайными значениями данной вероятности.)

При определении чувствительности результатов моделирования к изменениям двух или более параметров в общем случае *нельзя* один параметр, при этом беря произвольные значения других параметров. (Такой опасный подход называется подходом *однофакторного эксперимента*.) Правильнее использовать статистическое экспериментальное проектирование. Формально можно оценить влияние каждого параметра. Если количество параметров не очень большое, то также можно оценить и взаимосвязь этих параметров.

5.4.8. Валидация результатов общей имитационной модели [5]

Наиболее точный тест валидности имитационной модели это оценка того, какие выходные данные имеют большое сходство с выходными данными реальной системы. Если система схожа с предполагаемой существующей, то имитационная модель существующей системы разработана и ее выходные данные сравнимы с выходными данными реальной системы. Если эти 2 набора данных не сильно различаются, то модель *реальной* системы считается «валидной». (Требуемая точность модели зависит от предполагаемого использования и функций лица, принимающего решения.) Затем модель модифицируется так, что она представляет рассматриваемую систему. Чем больше общего между моделью и реальной системой, тем больше уверенности в модели рассматриваемой системы. Не существует универсального (полностью определенного) подхода к валидации модели рассматриваемой системы. Если бы такой поход существовал, то не было бы необходимости в имитационной модели. Если вышеописанное сравнение успешно, то к тому же обеспечивается надежность использования имитационной системы. (Как говорилось выше, идея сравнения выходных данных модели и системы называется валидацией результатов.)

Пример 7. Для увеличения производительной мощности производитель продукции из огнеупорного алюминия хотел поменять используемую печь периодического действия на новую печь непрерывного действия. При моделировании существующей системы было обнаружено, что производительность модели отличается от реальной производительности менее чем на 1%. Таким образом, получилось, что модель существующей системы была обоснованно «валидна».

Для сравнения выходных данных имитационной модели и реальной системы в литературе по валидации предлагается несколько статистических тестов (t, тест Манна-Витнея и т.д.). Тем не менее, так как выходные данные (процессы) практически всех реальных и имитационных систем являются *нестационарными* (распределение успешных наблюдений постоянно меняется) *и автокореллированными* (все наблюдения связаны друг с другом), то такое сравнение не является простой задачей, как это может показаться на первый взгляд. Таким образом, невозможно применять напрямую классические статистические тесты, основанные на независимых, одинаково распределенных наблюдениях. Более того, возникает вопрос, являются ли критерии проверки гипотез более подходящими по сравнению с построением доверительных интервалов. Так как модель является лишь аппроксимацией реальной системы, то нулевая гипотеза о том, что модель и система «идентичны» ложна. Лучше спрашивать, является ли разница между моделью и системой достаточной для того, чтобы на основании этой модели можно было сделать какие-либо выводы.

Для сравнения выходных данных модели и системы, кроме статистических процедур, можно использовать *тест Тьюринга* [9]. Людей, хорошо знающих систему, (например, экспертов) просят проанализировать наборы данных системы и модели. При этом, они не знают какие данные были получена на основе модели, а какие из самой системы. В случае, если эксперты находят различия между этими наборами данных, то для улучшения модели используется их объяснения.

Пример 8. При валидации имитационной модели потока машин на автостраде использовался анимированный тест Тьюринга. Анимация пассажиропотока имитационной модели отображалась на большой монитор параллельно с изображением реальной ситуации. Данные с автострады были получены с помощью видеокамеры, установленной на самолете.

Независимо от того, существует ли реальная система, аналитики и эксперты должны проверить выходные данные имитационной системы (численные результаты, анимацию и т.д.) на корректность. (Так как желаемые выходные данные точно неизвестны (иначе бы не было бы необходимости в создании модели), то такая проверка должна проводиться очень тщательно.) Если результаты имитационного моделирования не противоречат работе системы, то, как говорилось выше, говорят, что модель имеет экспертную (или face) валидность.

Пример 9. Описанная выше идея была удачно применена при разработке имитационной модели системы кадров ВВС США. (Эта модель была разработана для того, чтобы обеспечить экономических аналитиков информацией в масштабе все системы о влиянии различных кадровых политик.) Результаты работы модели с основной кадровой политикой были предоставлены аналитикам ВВС США и лицам, принимающим решения. После чего они нашли некоторые различия между моделью и реальным поведением системы. Эти данные были использованы для усовершенствования модели. После нескольких дополнительных оценок и усовершенствования была получена модель, довольно точно аппроксимирующая используемую кадровую политику ВВС США. Благодаря этому была увеличена не только валидность модели, но и ее надежность.

5.4.9. Использование графиков и анимаций выходных данных имитационной модели [5-7]

Для демонстрации того, что модель не является валидной, а также для увеличения надежности модели можно использовать графики (статические и динамические) и анимации (динамические). Ниже приводится несколько примеров использования графиков:

- Гистограмма (графическая оценка плотности вероятности и функции массовости)
- График корреляции (показывает, являются ли выходные данные автокореллированными)
- График временной зависимости (для отображения поведения системы в течение *длительного периода* значения одной или нескольких переменных модели изображаются вдоль оси времени работы модели)
- Столбчатая диаграмма (гистограмма) и секторная диаграмма

При обсуждении сущности модели лицом, принимающим решения, и другими участниками проекта, которым нет необходимости вникать в технические подробности модели, удобно использовать анимацию поведения системы во время *краткосрочного* периода. Это увеличивает надежность модели. Также анимацию удобно использовать для

верификации окончательной имитационной программы, рассмотрения усовершенствованных процедур и для обучения.

5.5. Статистические методы сравнения выходных данных модели и системы [5]

В данной части обсуждается возможные варианты использования статистических процедур сравнения выходных данных модели и системы.

Допустим, что R_1, R_2, …, R_k – наблюдения реальной системы, а M_1, M_2, …, M_l – выходные данные соответствующей имитационной модели (см. пример 10). Для определения того, является ли модель точным представлением реальной системы, необходимо сравнить эти наборы данных. Тем не менее, в большинстве классических статистических подходов, таких как доверительные интервалы и проверка гипотез, предполагается, что данные реальной системы и модели независимы и одинаково распределены, что на практике обычно не так (см. часть 2.8). Таким образом, для решения данной задачи такие классические статистические подходы нельзя применять *напрямую*.

Пример 10. Рассмотрим производственную систему, в которой выходными данными является время выполнения успешно завершенных операций. Эти данные зависят от реальной системы (а не от соответствующей имитационной модели). Например, если в данный момент времени система занята, то в системе для всех выполняемых операций будет выделяться как можно больше времени (то есть, время *положительно коррелировано*).

Для сравнения выходных данных модели и системы можно использовать наблюдения, доверительные интервала, а также другие подход, основанные на использование временных рядов.

5.6. Основные принципы получения хороших данных

Модель является валидной для определенной задачи, если корректна логика ее работы *и* используются соответствующие данные. В данной части предлагаются методы получения хороших данных.

5.6.1. Основные принципы

Если модель схожа с существующей системой, то данные для построения модели берутся из системы. Эти данные можно получить из архивных записей, а также собрать во время исследования системы. Так как мнения людей, предоставляющих данные, и аналитиков могут различаться, то необходимо соблюдать следующие принципы:

- Необходимо, чтобы предоставляли лицам, предоставляющим данные, точное описание требований к данным (тип, формат, количество, для чего необходимы, условия, при которых данные собираются, и т.д.)
- Необходимо, чтобы аналитик понимал сам процесс получения данных, а не только абстрактную обработку наблюдений. Например, предположим, что в нескольких наблюдениях были получены данные, значительно отличающиеся от данных в большинстве наблюдений. Без глубокого понимания основных принципов работы системы невозможно являются ли эти данные результатом ошибки или это допустимые значения, которые появляются с малой вероятностью.

5.6.2. Препятствия для получения хороших данных

Ниже приводятся четыре препятствия, которые не позволяют получить хорошие данные:

- Данные не отражают требуемых характеристик модели.

Пример 11. Данные, полученные при исследовании поля сражения, могут отличаться от реальной ситуации из-за различий в поведении исследовательской группы и войск, а так же из-за недостаточного учета каких-либо обстоятельств, например, появления дыма во время сражения.

- Используются данные неверного (неподходящего) формата или типа.

Пример 12. При моделировании производственной системы случайный характер носят простои машин. В идеальном случае нам необходимы данные о времени сбоев и времени восстановления машин. Иногда эти данные собирают во время сбоев, но зачастую они не представлены в требуемом формате. Например, время сбоя может быть основано на физическом времени и включать в себя периоды простоя машины.

- Данные могут содержать ошибки измерений, записи или округления.

Пример 13. Данные, представляющие время выполнения некоторого задания, иногда округляется до ближайших 5 или 10 минут. Так как данные дискретны, то усложняется использование непрерывных теоретических законов распределения.

- Данные могут быть «приукрашены»

Пример 14. Административно-хозяйственный отдел автомобильного завода намеренно предоставил преувеличенные данные о надежности машины.

Заключение

Необходима валидация *всех* имитационных моделей, иначе решения, принятые на основе этих моделей, будут неверными. Ниже приводятся наиболее важные идей разработки валидных и надежных моделей:

- Точная формулировка проблемы.
- Проведение интервью с экспертами в данной предметной области.
- Постоянное взаимодействие лица, принимающего решения с участниками проекта, что гарантирует корректность решаемой задачи, а также увеличивает надежность модели.
- Разработки письменной 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.

Лекция 6. **Использование языка XML в имитационном моделировании**

Введение

В последнее время появились публикации, в которых описывают активное применение языка XML в имитационном моделировании. Известно, что XML обладает рядом достоинств. XML является удобным средством стандартизации представления информации и реализации обмена между компонентами распределённой системы. Кроме того, достоинством языка XML является его "бизнес-гибкость", проявляющаяся в том, что XML постоянно находит новое практическое применение.

6.1. Особенности применения языка XML

XML [92](eXtensible Markup Language – расширяемый язык разметки) 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.

Рассмотрим язык разметки HTML (HyperText Markup Language – гипертекстовый язык разметки). HTML, 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. В чем же заключаются его достоинства?

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.

Документ XML состоит из элементов XML. Элементы XML иногда называют узлами, так как структура документа XML имеет вид дерева. Элемент XML состоит из открывающего тега, закрывающего тэга и расположенных между ними данных. Для обозначения тэга используются символы “<” и “>”, внутри которых помещаются название элемента и его атрибуты, содержащие дополнительную информацию об элементе или уточняющие его описание. Примеры элементов XML:

- **<Author>Richard Fujimoto</Author>**
  Этот элемент XML называется **Author**, а его содержимым является **Richard Fujimoto**
- **<Title year = “2002”>Disrtributed simulation</Title>**
  Этот элемент XML называется **Title**, атрибутом его является **year** со значением **2002**, заключенным в кавычки, а содержимым элемента является **Using XML for simulation modeling**

Любой документ XML начинается с уникального первого элемента, который служит для этого документа корневым узлом. Элементы могут содержать вложенные элементы, но это не является обязательным условием. Вложение элементов создает определенные связи между ними и определяет структуру документа. В приведенном ниже примере представлен код очень простого документа XML:

```
<Works>
    <Author>
        <Name> Richard Fujimoto </Name>
        <Title year = “2002”>Distributed simulation</Title>
    </Author>
    <Author>
        <Name>Dean C. Chatfield</Name>
        <Title year = “2004”>XML-based supply chain simulation modeling</Title>
    </Author>
</Works>
```

6.1.1. Правила для создания XML-документов

Язык XML является высоко структурированным языком, и поэтому важно, чтобы все XML-документы были синтаксически правильными. Под этим подразумевается, что при создании документа необходимо следовать определенному набору правил. Приведем полный список этих правил:

- В заголовке документа помещается объявление XML вида <?xml?>, в котором указывается язык разметки документа, номер его версии и дополнительная информация.
- Любому открывающему тэгу должен соответствовать закрывающий тэг.
- Тэги XML чувствительны к регистру.
- В любом документе XML должен содержаться один корневой элемент для всего документа.

– Элементы не должны перекрываться. Перекрывание элементов возникает, когда закрывающий тэг внешнего элемента располагается перед закрывающим тэгом внутреннего элемента.
– Можно определять пустые элементы. Это делается двумя способами: <tag/> и <tag></tag>.
– В XML имеется несколько зарезервированных символов, которые используются как элементы синтаксиса XML и, следовательно, при употреблении этих символов в тексте нужно заменять их специальными последовательностями других символов, называемых сущностями. Следует заметить, что сущности также чувствительны к регистру.

Зарезервированный символ	**Замена**
<	&lt;
&	&amp;
>	&gt;
“	&quot;
‘	&apos;

Состоятельные документы XML должны быть не только правильными, но и удовлетворять ряду дополнительных присоединенных к документу правил, которые описывают документ в целом и отношения между его элементами. Эти правила помогают приложению понять компоновку данных в том случае, когда отсутствует встроенное описание формата. В настоящее время существует два способа создания таких правил, сопровождающих документ: DTD и схемы. Схемы основаны непосредственно на XML и, следовательно, так же способны к расширению и позволяют разработчику выйти за пределы тех функциональных возможностей, которые предусматриваются DTD.

6.2. XML и имитационное моделирование

Язык XML применяется в 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)[96] и Lu, Qiao и McLean (2003)[97] 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)[94] 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)[93]. 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.

6.2.1. 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.

Общая структура SDF-файла имеет следующий вид:

```
<shop-data type="" identifier="" number="">
    <name />
    <description />
    <reference-keys />
    <revisions />
    <units-of-measurement />
    <organization-directory />
    <calendars />
    <resources />
    <skill-definitions />
    <setup-definitions />
    <operation-definitions />
    <maintenance-definitions />
    <layout />
    <parts />
    <bills-of-materials />
    <inventory />
    <procurements />
    <process-plans />
    <work />
    <schedules />
    <time-sheets />
    <references />
    <probability-distributions />
</shop-data>
```

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.

6.2.2. 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, который будет обсуждаться в этом разделе, 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.29):

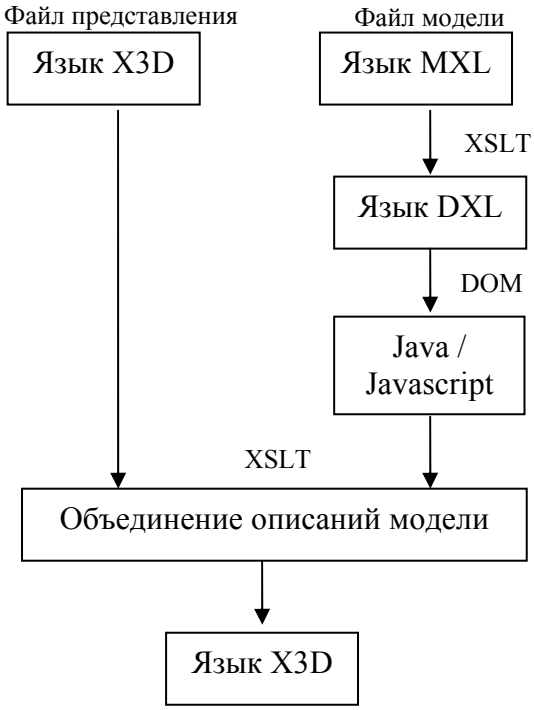


Рис. 29. Структура проекта Rube

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.

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. Ниже в качестве примера приведен неполный текст файла модели:

```
<MXL>
<model id="FSM2S2T" type="FSM">
    <topology type="GRAPH">
    <node id="S1" type="STATE" start="TRUE">
    <script id="S1.js" func="S1_func"/>
    </node>
    <node id="S2" type="STATE">
    <script id="S2.js" func="S2_func"/>
    </node>
    <edge id="T12" type="TRANSITION" begin="S1" end="S2"
    data_type="INTEGER">
    <script id="T12.js" func="T12_func"/>
    </edge>
    </topology>
    …
```

В примере описывается полный двухвершинный граф.

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. Ниже приведен отрывок текста на языке DXL:

```
    <DXL>
    <block id="IN">
    <port id="IN.OP1" type="OUTPUT" target="S1.IP1" data_type="INTEGER">
    </port>
    <definition id="IN.js" func="IN_func"> </definition>
    </block>
    <block id="S1">
    <port id="S1.IP1" type="INPUT" source="IN.OP1" data_type="INTEGER">
    </port>
    <port id="S1.OP1" type="OUTPUT" target="T12.IP1" data_type="INTEGER">
    </port>
    <port id="S1.OP2" type="OUTPUT" target="OUT.IP1" data_type="INTEGER">
    </port>
    <definition id="S1.js" func="S1_func"> </definition>
    </block>
    …
```

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 плюсам 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 минусам 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.

6.2.3. 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 для определения базового синтаксиса языка SML использовать XML. Использования 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, как показано в следующем примере:

```
<object name="Product11">
        <attribute name="Color">
        <attribute name="Priority">
<process>
        <actions>
                <event name="OnInit">
                        set m=APP("Simulation").system("SILK").model("Testmodel")
                        m.object("Product11").att("Priority")=10
                </event>
                <event name="DuringSimulation">
                        generate 1 object("ThisObject") every 120 seconds
                        m.object("Buffer1").store("ThisProduct")
                        m.object("Machine1").workon("ThisProduct").fortime(10)
                </event>
        </actions>
        <options output="tracelist" animation="ON">
</process> </object>
```

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 ниже:

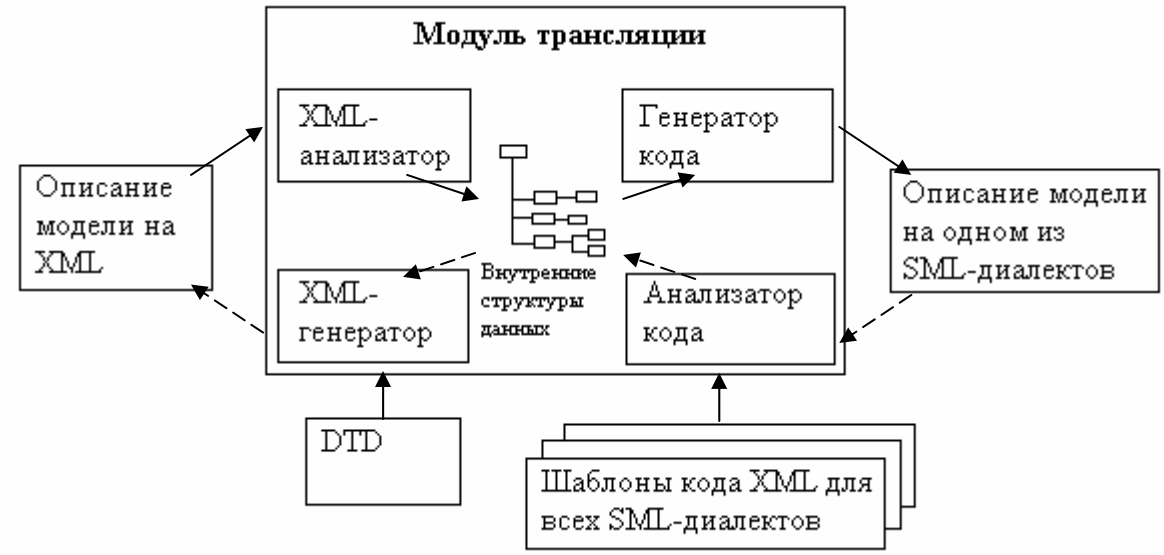


Рис. 30. Схема трансляции

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.

6.2.4. 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. В данном разделе обсуждается, 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.

В таблице приведены основные конструкции языка SRML и описание их назначения.

Таблица 1. Назначение атрибутов

Элемент или атрибут	Назначение
Элемент <srml:Simulation>	Для инкапсуляции структуры и поведения отдельной имитационной модели
Элемент <srml:Simulations>	Для инкапсуляции отдельных имитационных моделей в имитационную систему
Элемент <srml:ItemClass Name=”classname” SuperClasses=”class1 class2”>	Для описания классов
Элемент <srml:Script Type=”language”>	В описании элемента: для описания поведения элемента; в описании класса: для описания поведения всех элементов, принадлежащих данному классу
Элемент <srml:Link Target=”XPath”/>	Для определения ассоциативной связи между элементами
Атрибут <… srml:Quantity=”value”>	В описании элемента: для представления составных элементов
Атрибут <… srml:Source=”target”>	Для дополнительной модуляризации документа BOM путем указания ссылок на внутренние элементы, которые бывают внедренными, связанными или присоединенными

Для обработки BOM и Mega-BOM, написанных на SRML, то есть для создания и выполнения отдельных имитационных моделей и имитационных систем, был разработан универсальный имитатор под названием Simulation Reference Simulator. Использование SRML и универсального имитатора SR Simulator значительно сокращает затраты, связанные с построением имитационных моделей.

Лекция 7. **Использование онтологий в имитационном моделировании**

Введение

Важной проблемой имитационного моделирования является возможность переиспользования 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 [103]:

- 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.

По мнению автора[103] для успешной автоматизации достраивания моделей не обязательно иметь общие знания о мире в целом. 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.

7.1. 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. Это 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.

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[104].

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. IF BOMs используются для:

- Обеспечения общего описания уровня метаданных, называемого идентификацией модели
- Описание активностей, используемых в шаблоне
- Определение 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.

OWL

OWL (Web Ontology Language) – язык для представления онтологий и связанной информации в виде семантической сети. OWL был создан WWW консорциумом в 2004м 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[105].

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 [106]. 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 [107] 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 [108] 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) [110] 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. 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 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.

Лекция 8. **Агентное моделирование**

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.[40,41,42, 43,44,45].

Выясним более подробно, что такое агенты, их классификацию, рассмотри архитектуру мультиагентных систем, сферы применения мультиагентных систем, различие между дискретным и агентным моделированием.

8.1. 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 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.*)[46] 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 ***другими агентами***. Агенты ***децентрализованы***[43], а ***глобальное состояние*** моделируемой системы можно вывести из взаимодействия агентов, зная индивидуальную логику поведения каждого из них.

Программной реализацией агента является объект (в смысле ООП) 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.

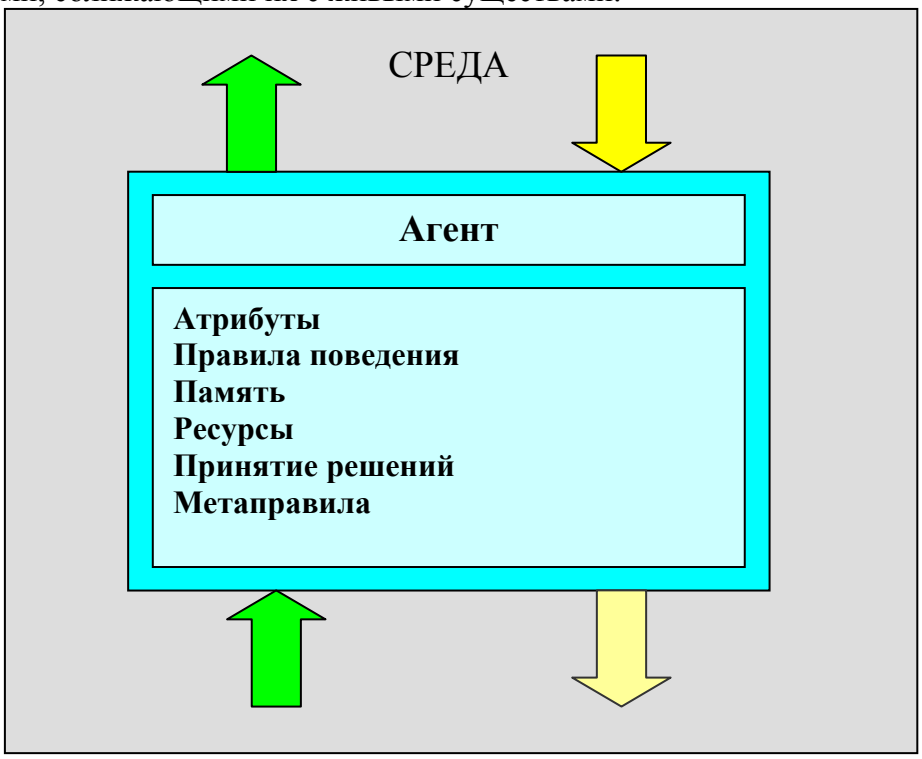


Рис. 31. Агент и его характеристики

Итак, про агента можно сказать [40], что:

– 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.
– 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.

– Агент целеустремлён, имеет цели, которые необходимо достичь, исходя из его поведения.
– Агент автономный и самонаводящийся. Агент может функционировать в своей среде независимо от сделок с другими агентами, по крайней мере в небольшом числе ситуаций.
– Агент гибкий и может обучаться и адаптировать своё поведение по времени, основываясь на опыте. Агент может иметь метаправила, правила, изменяющие другие правила поведения.

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!

По замечанию автора [45] существуют разные термины для обозначения агентного моделирования. Это и индивидуумное моделирование (ИМ), 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 ОО 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

8.2. Истоки 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.

8.3. Агентное моделирование и другие науки

АМ 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.

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 - это:

- **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 (ГА) при исследовании САС. ГА это общая процедура поиска, основывающаяся на генетических механизмах и естественной селекции, и является одной из основ для оптимизационных алгоритмов.

8.4. Использования простых правил для проявления организации и сложного поведения

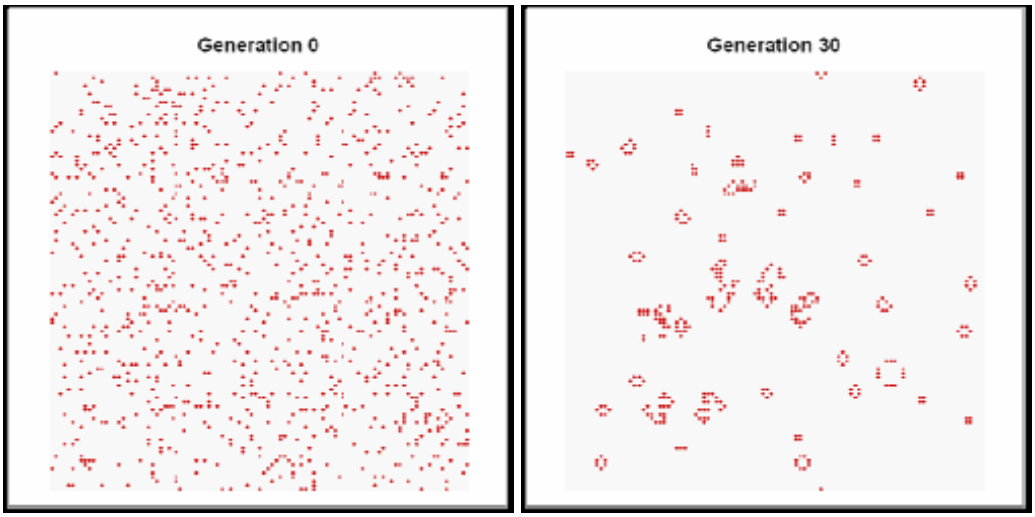
Обсуждение истоков АМ начинается с простой игры, разработанной математиком John Conway, игры «Жизнь». «Жизнь» основывается на клеточном автомате (КА). Пожалуй, это самый простой способ проиллюстрировать базовые идеи агентного моделирования. Первоначально идея КА была разработана физиком Уламом (Stanislaw Ulam) в ответ на вопрос, поставленный известным математиком 20-го века John von Neumann. Вопрос заключался в том, может ли машина быть запрограммирована для создания её собственной копии. На самом деле, этот вопрос заключался в том, возможно ли разработать логическую структуру (машину), достаточно сложную, чтобы полностью содержать в себе все необходимые инструкции для дублирования себя самой. Ответом, в итоге, оказалось что да, и в итоге было найдено абстрактное математическое представление для этой машины в форме клеточного автомата.

Типичный КА – это двумерная сетка или решетка, состоящая из ячеек. Каждая ячейка имеет конечное число состояний в любой момент времени. Набор простых правил

определяет значение каждой ячейки, основываясь на предыдущем состоянии ячейки. Каждая ячейка обновляется каждый период времени согласно правилам. Следующее значение ячейки зависит от текущего значения и значений непосредственных соседей в восьми окружающих ячейках. Правила обновления значений в каждой ячейке одинаковы. КА детерминирован в том смысле, что для одного и того же состояния ячейки и её соседей применение правил обновления приводит к одному и тому же обновленному состоянию. «Жизнь» руководствуется тремя правилами, для определения следующего состояния (включено или выключено) каждой ячейки:

– Ячейка будет находиться в состоянии включено. в следующем поколении, если ровно три из восьми ее соседей сейчас в включено.
– Ячейка останется в ее текущем состоянии, если ровно два из ее соседей находятся в состоянии включено.
– Ячейка будет выключено. в любом другом случае.

На рис. 32 - снимки из программы, моделирующей игру «Жизнь». Изначально ячейки, находящиеся в состоянии «включено» распределены случайно. После нескольких обновлений всех ячеек на сетке, появляются различимые закономерности, и в некоторых случаях эти закономерности поддерживают себя неопределенно долго на всем продолжении моделирования. Область взаимодействия агентов и локально доступную информацию для каждой ячейки, необходимую для обновления ее состояния, определяет предположение о существовании восьми соседей.



(a) Начальное состояние (b) Состояние после 30 обновлений

Рис. 32 Моделирование игры «Жизнь»

Итак, можно сделать следующие заключения относительно правил игры «Жизнь»:

– Эти правила просты.
– Эти правила используют только локальную информацию.

Состояние каждой ячейки базируется только на текущих состояниях самой ячейки и ячеек, непосредственно соприкасающихся с ней.

Таким образом, можно констатировать:

– Устойчивые закономерности могут появляться в системах, которые полностью описываются простыми правилами, основывающимися лишь на локальной информации, и
– Эти закономерности могут быть очень чувствительными к начальным условиям.

Wolfram продемонстрировал, что варьирование правил в клеточном автомате может привести к удивительному диапазону возникающих закономерностей, и эти закономерности полностью соотносятся с широким диапазоном алгоритмов и логических

систем. Wolfram утверждает, что простые правила могут быть использованы для понимания большей части сложностей, наблюдаемых в реальном мире.

Основываясь на простых правилах поведения и взаимодействия агентов, естественные системы явно проявляют групповой интеллект. Естественные системы способны не только выживать, но и адаптироваться. Каким образом муравьиная колония может самоорганизоваться для того, чтобы осуществить сбор еды и строительство колонии, кроме того, муравьиная колония в большой степени устойчива к разрушениям. Оптимизационные процессы в муравьиной колонии повлияли на развитие оптимизационных методов и на решение конкретных задач (например, для решения практической проблемы расписаний и проблемы маршрутизации).

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. (См. рис.33).

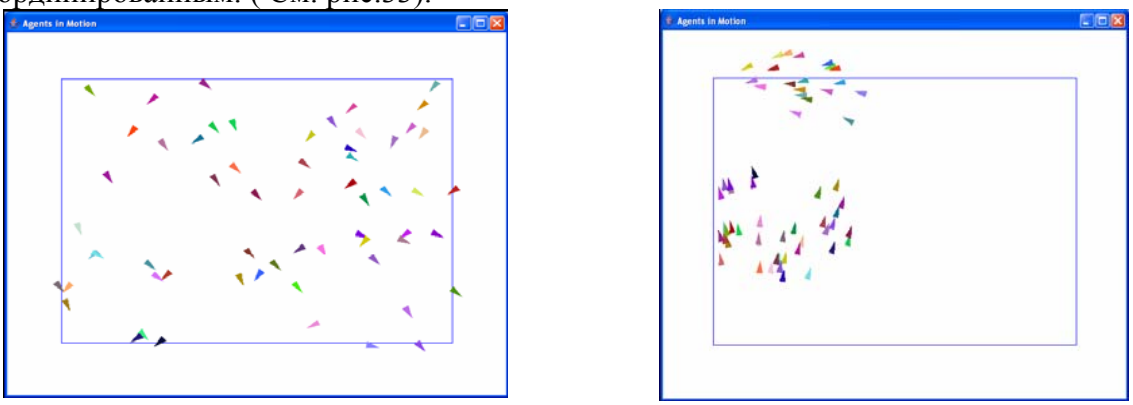


Рис. 33. Boids моделирование (начальная конфигурация и колония после 500 обновлений)

8.5. Агентное моделирование в науках

Рассмотрим более подробно применение агентного моделирования в различных областях науки.

Исследования в социальных науках. В приложениях АМ к социальным процессам, агенты представляют людей или группы людей, взаимоотношения агентов интерпретируют процессы социального взаимодействия. Агентное моделирование позволяет с достаточной степенью достоверности смоделировать взаимодействия людей в обществе на некотором уровне абстракции. Эти взаимодействия могут быть достоверно смоделированы, по крайней мере, для характерных и хорошо определенных целей. Эта ограниченная область для представления поведения агентов в АМ расходится с общими целями искусственного интеллекта. С точки зрения АМ некоторые важные вопросы становятся тотчас очевидными:

– Насколько много мы знаем о достоверном моделировании человеческого поведения?
– Насколько много мы знаем о моделировании социального человеческого взаимодействия?

Эти вопросы породили базовые исследовательские программы в социальных науках, которые обещали информировать АМ о теории и методах представления агентов и их поведения.

Томасу Шеллингу (Thomas Schelling) приписывается разработка первой социальной модели, основанной на агентах, в которой агенты представляли людей, а взаимодействия агентов - социальный процесс (агенты адекватно отображали этот процесс). Шеллинг применил понятие клеточного автомата, чтобы изучить закономерности жилищной сегрегации (разделения). Он сформулировал вопрос о том, возможны ли закономерности в разделении поселений, если большинство индивидуумов на самом деле не имеют расовых предрассудков? Модель Шеллинга продемонстрировала, что гетто могут развиваться спонтанно. Более того, Шеллинг показал, что закономерности, которые могут появляться, не обязательно следуют или даже не согласуются с целями (стремлениями) отдельных индивидуумов. Это важное наблюдение вызвало интерес и дало толчок для применения АМ в этой области. (Отметим, что первоначальные модели Шеллинга даже не были компьютерными: агенты представляли собой монеты, передвигаемые по шахматной доске). Эксперименты в области моделирования людей в растущих искусственных сообществах с помощью агентных моделей было продолжено Эпштейном (Epstein) и Акстеллем (Axtell) (модель Sugarscape (плантация сахара)). В многочисленных вычислительных экспериментах, агенты Sugarscape появлялись с разнообразными характеристиками и поведением, делая их похожими на реальное, хотя и элементарное общество. Исследовались новые процессы, включающие смерть, болезни, торговлю, богатство, борьбу и войны, и прочие вещи, вроде загрязнения.

Исследования в экономических процессах. Агентное моделирование используется и в экономике. Вот некоторые классические предположения стандартной микроэкономической теории:

– ***Экономические агенты рациональны***, что означает, что агенты имеют хорошо определенные цели и способны оптимизировать свое поведение. (основа модели «рациональный агент» широко используется в экономике и во многих других социальных дисциплинах),
– ***Экономические агенты однородны***, что означает, что агенты имеют одинаковые характеристики и правила поведения,
– Существуют ***понижающие факторы*** для регулирования экономических процессов, например, уменьшение предельной полезности, уменьшение предельной производительности, и т.д.
– ***Долгосрочное равновесие*** системы- это наиболее интересная информация.

Каждое из этих предположений в приложениях АМ для экономических систем несколько ослаблено.

Во-первых, действительно ли организации и индивидуумы оптимизируют? Герберт Саймон (Herbert Simon), Нобелевский лауреат (пионер в области искусственного интеллекта) разработал теория «удовлетворения», чтобы описать наблюдаемое поведение людей и организаций в реальном мире. Поведенческая экономика – это относительно новая область, которая объединяет экспериментальные результаты психологии и когнитивных аспектов принятия решений агентами, чтобы определить настоящее экономическое поведение людей.

Во-вторых, разнообразие агентов всюду встречается в реальном мире и это предмет исследование многих наук.

В-третьих, Саймон обнаружил «позитивные циклы обратной связи» и «увеличивающие факторы» как основу динамических процессов быстрого экспоненциального роста в экономике. Позитивные циклы обратной связи могут создавать самоподдерживающиеся процессы, которые быстро переводят системы с начальной точки (стартовой позиции) в другое состояние.

В-четвертых, долгосрочное равновесие системы – это не единственные интересующие исследователей результаты. Более того, не все системы приходят к состоянию равновесия. Область деятельности Агентной Вычислительной Экономики возникла из приложений АМ для экономических систем.

Исследования в антропологии

Антропологи также часто разрабатывают крупные системы моделирования на агентах для моделирования древних цивилизаций, чтобы помочь объяснить их развитие(рост) и упадок, основываясь на данных археологии. АМ было применено, чтобы помочь понять социальные и культурные факторы, ответственные за исчезновение Anasazi на юго-западе США и падение (упадок) древней Мессопатамской цивилизации.

Агентное моделирование в социологии

Агентное моделирование также используется в социологии. Macy и Willer провели обзор агентного подхода как основы для моделирования социальной жизни. Социальная жизнь рассматривается как взаимодействия между адаптивными агентами, которые влияют друг на друга в ответ на влияние, оказанное на них. Последний выпуск Американского Социологического Журнала посвящен в значительной степени агентному моделированию в социологии. Таким образом, вычислительные социологические науки становятся одним из направлений деятельности.

Агентное моделирование в политологии. Агентное моделирование начинает использоваться в политических науках. Например, Cederman использует агентное моделирование, чтобы понять основные процессы, вовлеченные в национальную идентификацию и государственный строй.

Агентное моделирование в когнитологии. Когнитология, наука о мышлении, имеет свое представление об агентном подходе, а социальная когнитология расширяет эти идеи до социальных установок. Учёные в области когнитологии разрабатывают агентные модели эмоций, познания, а социальное поведение базируется на идее, что человеческое эмоциональное состояние влияет на поведение, также как на социальное взаимодействие. Цель – создать искусственных агентов, которые бы реализовывали взаимодействие между эмоциями, сознанием и социальным поведение.

Исследования в биологии и физике. В дополнение к социальным наукам, АМ также начинает использоваться в физике и биологии. В физических науках АМ применяют для моделирования всевозможных возникающих структур на молекулярном уровне. В биологии, агентное моделирование применяется для моделирования поведения бактерий, их взаимодействия и самоорганизации колоний бактерий.

Топологические особенности

Клеточный автомат представляет закономерности взаимодействия агентов и локально доступную информацию, используя для этого сетку или решетку и ячейки, непосредственно окружающие агента в качестве окрестности. Другие топологии взаимодействия агентов, как сети, позволяют определять окрестность агента более гибко. Сети более точно описывают закономерности социального взаимодействия агентов.

Анализ сети социальных связей (АССС) – это область науки, которая занимается определением параметров и анализом социальной структуры (взаимодействие определяется сетевым представлением). Традиционно, АССС занималась статическими сетями, т.е. сетями, которые не меняли своей структуры во времени или в результате поведения агентов. В последнее время было обнаружено большое количество масштабируемых сетей, таких как Всемирная паутина, членство в корпоративных исполнительных комитетах, и экологических естественных средах.

Анализ динамических сетей – это новая область, которая заключает в себе механизмы роста и изменения сетей, основываясь на процессах взаимодействия агентов. Понимание основополагающих правил агентов, которые управляют структурой сетей, тем, насколько быстро информация передается через сеть, и типами отношений, которые реализованы в сети – это все важные аспекты моделирования агентов и «сетей АМ».

Рассмотрим пример: в этом моделировании присутствует два вида агентов, красные и синие. Первоначально, синих агентов в два раза больше чем красных. Каждая сторона пытается убедить другую сторону в правильности своих позиций. Агент, окруженный большим числом агентов одного цвета чем другого, переходит на позиции этого доминирующего агента. Следует сказать, что каждый красный агент вдвое убедительнее, чем синий агент. Количество красных агентов в окружении синего должно быть вдвое меньше, чем синих, синий агент может с тем же успехом перейти на позицию красных агентов. В этом примере исследуется: будет ли какой-нибудь тип агентов (мнения, позиции) доминировать или будет образована устойчивая смесь позиций в долговременной перспективе? Результаты моделирования приведены на рисунке 34.

Рис. 34. Моделирование социального влияния.

8.6. АМ приложения

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, рассмотренные в [45].

Адаптивная система рынка электроэнергии

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 (рис.35).

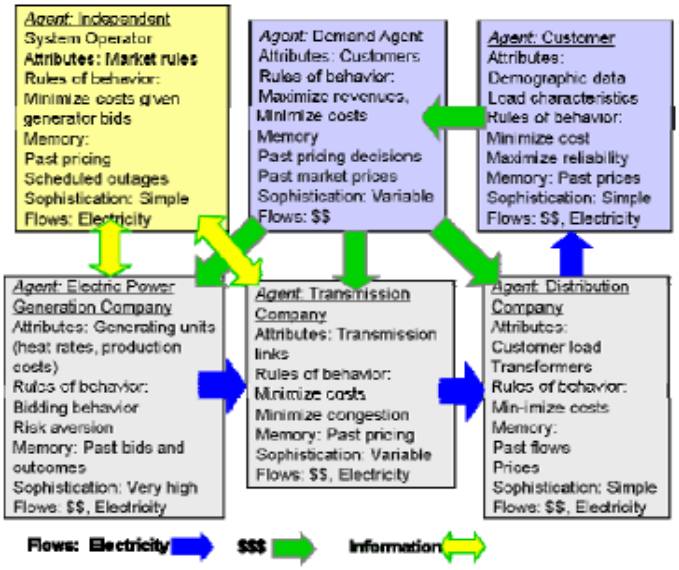


Рис. 35. Агенты и их взаимодействие в модели EMCAS.

Модель цепочки 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. 36). 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. Розничный торговец 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.

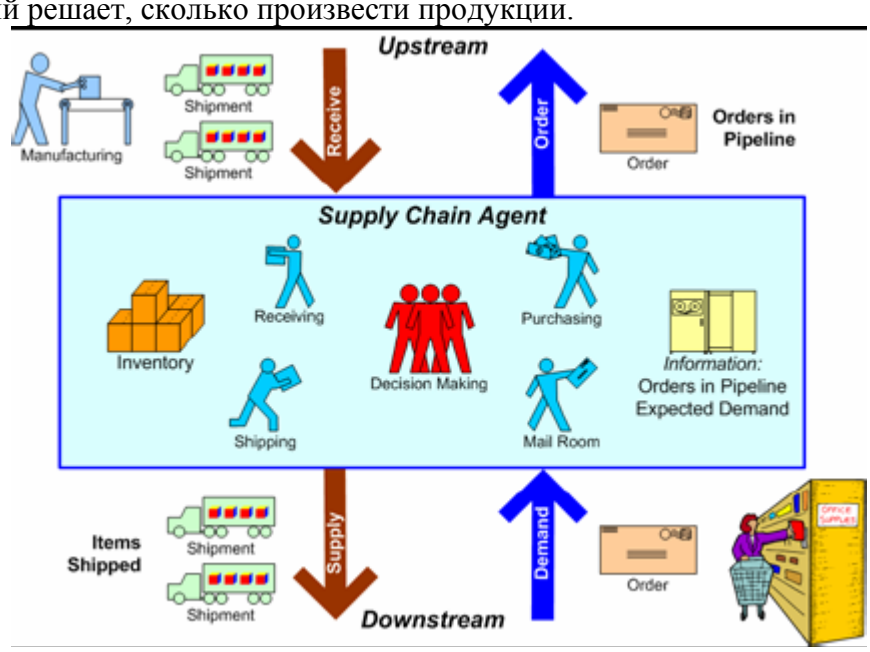


Рис. 36. Цепочка поставок

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.

8.7. Построение агентных моделей

Построение агентных моделей осуществляется по большей части также как и построение любой другой модели в моделировании (см. лекцию 5).

Авторы [45] отмечают характерные особенности в построения агентных моделей. В дополнение к стандартным задачам построения модели, практическое АМ требует следующего:

- Определить агентов и теоретические основы поведения агентов;
- Определить взаимоотношения между агентами и теоретические основы таких отношений;
- Найти платформу для АМ и разработать стратегию АМ модели;
- Получить необходимые данные для агентов,
- Проверить (этап валидации) модели поведения агентов (в дополнение ко всей модели в целом);
- Запустить модель и проанализировать выходные данные с точки зрения связи между поведением агентов на микроуровне и поведением всей системы в целом.

Рассмотрим каждый из этапов более подробно.

Определение агентов

Определение агентов с точным заданием их поведения и взаимодействия с другими агентами – это основа для разработки достоверных агентных моделей. Агенты – это те, кто обычно принимают решения в системе (лицо, принимающее решение – ЛПР). Они представляют собой:

- традиционных ЛПР;
- менеджеров;
- нетрадиционных ЛПР (сложные компьютерные системы со сложным поведением);
- группы (если это продиктовано целями агентного моделирования).

Как только агенты определены, следующей главной задачей становится определение поведения агентов. Здесь можно рекомендовать следующее:

- Найти теоретические основы поведения агентов. Например, можно начать с нормативной модели и использовать эту модель в качестве отправной точки для разработки простой и наглядной эвристической модели поведения
- Можно также начать с поведенческой модели, если есть подходящая поведенческая теория и результаты ее применения выглядят адекватными. Например, имеется большое число теорий для моделирования поведения покупателя, основанных на эмпирических знаниях.
- Когда поведение отдельных личностей (людей) используется в качестве основы для агентных моделей существующих или гипотетических систем, полезные техники для применения можно позаимствовать у инженерии знаний и общего моделирования.

Общее АМ сочетает парадигму агентного моделирования с идеями из организационной теории, чтобы определить модель, управляемую целями, которая полностью состоит из участников-людей, играющих роли, сродни игре, но намного более

структурировано. Общее моделирование полезно использовать на начальной стадии разработки агентной модели. Используя правильную структуру, инструкции и дисциплину, люди в общем АМ могут обнаружить намного больше информации о поведении агентов, а именно:

- Насколько много информации люди способны обработать за заданное время для принятия решений?
- Какие факторы и признаки люди считают самыми важными для принятия решений?
- Каким образом прошлый опыт людей влияет на процесс принятия решений?
- Какие стратегии из тех, которые сформулированы людьми, являются наиболее эффективными?

Общее агентное моделирование может быть использовано для улучшения понимания и проверки правильности функционирования агентных моделей, для определения, насколько эти можели правдоподобны, для демонстрации концепций агентного моделирования всем заинтересованным сторонам и для тестирования идей о поведении агентов в нестандартных ситуациях.

8.8. Средства разработки систем АМ

Агентное моделирование может быть реализовано на небольших, настольных компьютерах, или с использованием крупных кластеров компьютеров, или на любом другом варианте между первыми двумя.

Настольное АМ

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 АМ

Крупномасштабное АМ 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 Repast, Swarm, NetLogo и MASON.

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 Repast Py спроектирован 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.37. Линии на нижнем экране отображают взаимодействие между соединенными агентами.

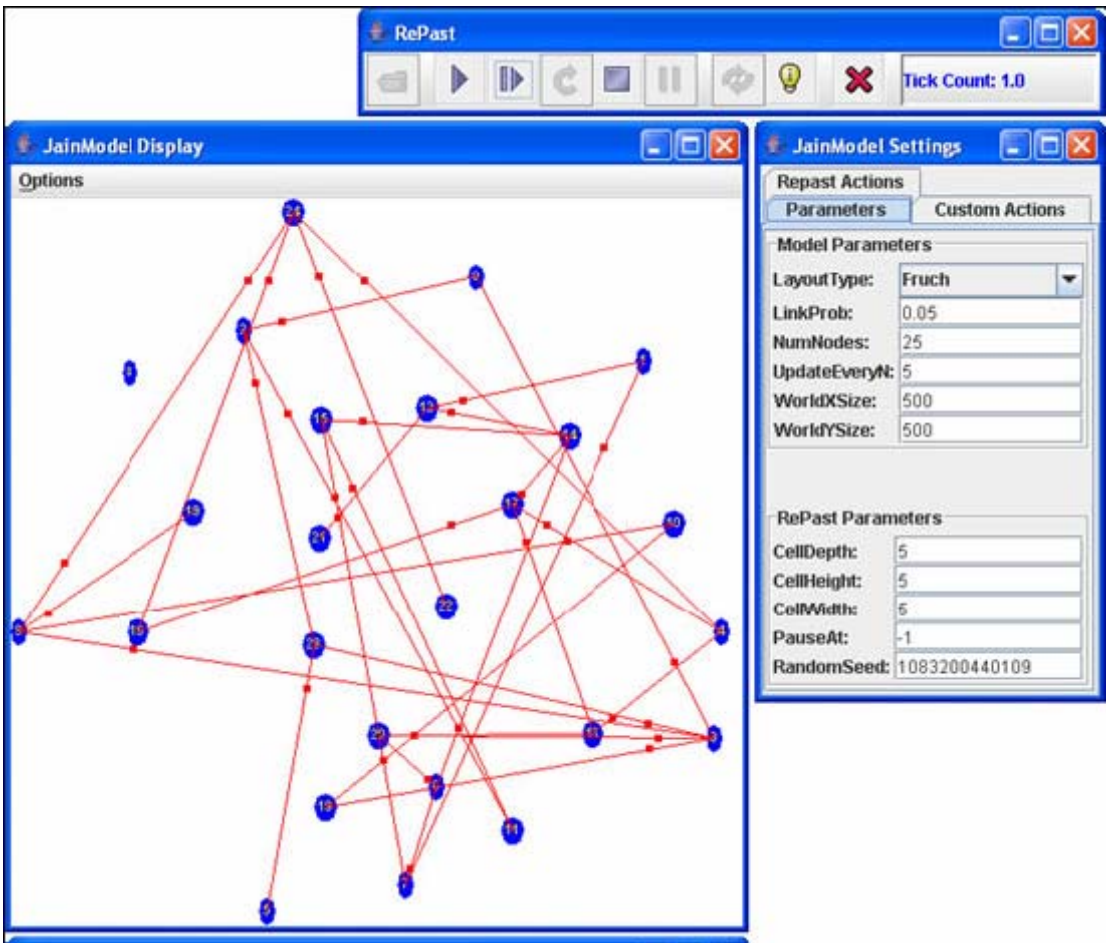


Рис. 37 Интерфейс Repast J: отображение модели сети “Jain”

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

Swarm (стая, рой) был 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 компонент 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.

AniLogic

AniLogic – это отечественная разработка. В настоящее время она нашла широкое применение среди отечественных и зарубежных пользователей.

8.9. Применение АМ

Агентное моделирование **используют** для изучения возникновения закономерностей и исследования того, как появляются структуры в изучаемой системе, которые не проявлялись в поведении отдельных агентов.

Можно порекомендовать применять агентное моделирование в следующих ситуациях:

- Когда представление агентов естественно.
- Когда существуют сценарии поведения и решения, которые могут быть определены дискретно.
- Когда важно, чтобы агенты могли адаптироваться и менять свое поведение.
- Когда важно, чтобы агенты обучались.
- Когда важно, чтобы агенты имели динамические связи с другими агентами, и эти связи могли формироваться и исчезать.
- Когда важно, чтобы агенты формировали организации, и адаптация и обучение были важны на уровне организации.
- Когда важно, чтобы агенты имели пространственную составляющую своего поведения и взаимодействий.
- Когда в прошлом нет предсказания для будущего.
- Когда важны пропорциональные увеличения до произвольного уровня.
- Когда процесс структурных изменений должен быть результатом модели, а не результатом входных данных модели.

Библиографический список

1. Richard M. Fujimoto. Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. pp. 124-134
2. Richard E. Nance. Distributed Simulation With Federated Models: Expectations, Realizations And Limitations. In Proceedings of the 1999 Winter Simulation Conference. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds. pp. 1026-1031.
3. Enver Yucesan, Yah_Chuyn Luo, Chun-Hung Chen, Insup Lee. Distributed Web-based Experiments for optimization. Simulation Practice and Theory 9 (2001) pp.73-90.
4. Richard A. Meyer, Rajive Bagrodia. Parsec User Manual. Release 1.1., UCLA Parallel Computing Laboratory, 1998 pcl.cs.ucla.edu/projects/parsec
5. Briane J.Premore, David M.Nicol. Parallel Simulation of TCP/IP Using TeD. In Proceedings of the 1997 Winter Simulation Conference S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson S., eds. pp. 436-447
6. Ferenci S, Perumalla K., and Fujimoto R. An Approach to Federating Parallel Simulators //Workshop on Parallel and Distributed Simulation, May 2000 (http://www.cc.gatech.edu/ computing/ pads/papers.html)
7. Вознесенская Т.В. Математическая модель алгоритмов синхронизации времени для распределённого имитационного моделирования.//В кн. Программные системы и инструменты. Тематический сборник факультета ВМиК МГУ им. Ломоносова №1., стр.56-66
8. В.В. Окольнишников. Представление времени в имитационном моделировании. Вычислительные технологии. Т. 10, №5, Сибирское отделение РАН, 2005, стр. 57-77
9. Основные классы современных параллельных компьютеров, Лаборатория НИВЦ МГУ. http://www.parallel.ru/computers/classes.html
10. Э. Таненбаум. Распределенные системы: принципы и парадигмы. – СПб: Питер, 2003. – 877 с.
11. Россия и грид. Информационный Интернет-канал НАУКА и ИННОВАЦИИ Доступно на сайте: http://www.rsci.ru/rt/?action=more&id=59
12. А. Лацис. Как использовать и построить суперкомпьютер.-М.:Бестселлер,2003.-240с.
13. С.Немнюгин, О.Стесик. Параллельное программирование для многопроцессорных вычислительных систем.-СПб.:БХВ-Петербург, 2002.-400с.
14. В.В.Воеводин., Вл.В.Воеводин. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002.-608с.
15. Корнеев В.В. Вычислительные системы. М.:Гелиос АРВ, 2004-512с.
16. Jalal Nikoukaran, Vlatka Hlupic, Ray J. Paul. Criteria For Simulation Software Evaluation. In Proceedings of the 1998 Winter Simulation Conference D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.pp. 399-406.
17. J.C.Comfort. The Simulation of master-slave event set processor. Simulation.42.3 (1984) pp.117-124
18. A.I.Concepcion. A hierarchical computer architecture for distributed simulation, IEEE Transactions on Computing C-38.2 (1989) 311-319.
19. Terry L. Wilmarth, Laxmikant V. Kal´ e. Getting Over Grainsize in Parallel Discrete Event Simulation. Pp/
20. Bryant, R. E. (1977). Simulation of packet communications. Architecture computer systems. MIT-LCS-TR-188.
21. Chandy, K. M. and J. Misra (1978). “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs.” IEEE Transactions on Software Engineering SE-5(5): 440-452.
22. Chandy, K. M. and J. Misra (1981). “Asynchronous Distributed Simulation via a Sequence of Parallel Computations.” Communications of the ACM 24(4): 198-205.

23. Fujimoto, R. M. (2001). Parallel and Distributed Simulation. In Proceedings of the Winter Simulation Conference.
24. А.И.Миков. Автоматизация синтеза микропроцессорных управляющих систем.- Иркутск, Иркут.ун-т.,1987-288с.
25. Л.Н.Лядова. Имитационное моделирование. Методические указания по курсу «Системное и прикладное программное обеспечение» /Перм.ун-т; Сост. Л.Н.Лядова.- Пермь, 2003.-60 с.
26. Linda F. Wilson and Wei Shen. Experiments In Load Migration And Dynamic Load Balancing In Speedes. Proceedings of the 1998 Winter Simulation Conference.D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds, pp.487-490
27. Wilson, L. F., and D. M. Nicol. 1995. Automated Load Balancing in SPEEDES. Proceedings of the 1995 Winter Simulation Conference, pp 590–596.
28. Gengbin Zheng. Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing; in Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005,165p. Доступно на сайте: http://charm.cs.uiuc.edu/
29. Terry Lynn Wilmarth Pose: Scalable General-Purpose Parallel Discrete Event Simulation.. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005, pp.152 - Доступно на сайте: charm.cs.uiuc.edu/research/pose
30. Миков А.И., Замятина Е.Б., Фатыхов А.Х. Система оперирования распределёнными имитационными моделями сетей телекоммуникаций. Труды Второй Всероссийской научной конференции «Методы и средства обработки информации». М.: Изд-во МГУ, 2003 г., сс. 437-443.
31. Foster I. Designing and building Parallel Program: Concepts and Tools for Parrallel Software Engineering. Addison-Wesley Publishing Co,1995.
32. Lyadova L.N., Vodovoz N.A. Mapping Problem in Network Time Optimization. //Математика программных систем: Межвуз. сб. науч. тр./ Перм. ун-т, - Пермь, 2001.- сс. 48-56.
33. Копысов С.П. Динамическая балансировка нагрузки для параллельного распределённого МДО. Труды Первой Всероссийской научной конференции «Методы и средства обработки информации». М.:Изд-во МГУ, 2003 г., сс.222-228.
34. Курилов Л.С. Прогностическая стратегия балансировки загрузки для невыделенных кластерных систем. Труды Первой Всероссийской научной конференции «Методы и средства обработки информации». М.:Изд-во МГУ, 2003 г., сс. 413-418.
35. Gerard Tel. Introduction to Distributed Algorithms. Cambridge University Press, 1994, 534 p.
36. Defense Modeling and Simulation Office (DMSO). 1996. HLA Time Management Design Document, Version 1.0, dated 15 August 1996. Available online at the HLA Homepage (DMSO 1997).
37. Defense Modeling and Simulation Office (DMSO). 1998. High Level Architecture Rules, Version 1.3 (Draft 2), dated 5 February 1998. Available online at the HLA Homepage.
38. Defense Modeling and Simulation Office (DMSO). 1998. High Level Architecture Interface Specification, Version 1.3 (Draft 9), dated 5 February 1998. Available online at the HLA Homepage.
39. Defense Modeling and Simulation Office (DMSO). 1998. High Level Architecture Object Model Template, Version 1.3, dated 5 February 1998. Available online at the HLA Homepage.
40. Charles M. Macal, Michael J. North. Tutorial On Agent-Based Modeling And Simulation. Proceedings of the 2005 Winter Simulation Conference pp.
41. Patrick F. Riley, George F. Riley.Spades — A Distributed Agent Simulation EnvironmentWith Software-In-The-Loop Execution. Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds., pp.

42. Т.А.Гаврилова, В.Ф.Хорошевский. Базы знаний интеллектуальных систем. СПб.:Питер, 2001,-384 с.
43. А.Борщёв. От системной динамики и традиционного ИМ –к практическим агентным моделям: причины , технология , инструменты. Доступно на сайте: www.gpss.ru
44. Andrei Borshchev, Yuri Karpov, Vladimir Kharitonov Distributed Simulation of Hybrid Systems with AnyLogic and HLA Доступно на сайте www.gpss.ru.
45. A.M.Uhrmacher. Simulation for Agent-Oriented Software Engineering/ www.thesimguy.com/GC/papers/WMC02/G067_UHRMACHER.pdf
46. Michael Wooldridge and Nick R.Jennings. Intelligent agents: Theory and Practice. / Knowledge Engeneering Review. Vol 10, no. 2. ,pp. 115-152, Jun 1995.
47. Brian Logan and Georgios Theodoropolous. A Distributed Simulation of Multi-Agented Systems. Proceedings of IEEE, Vol 89, No 2, February 2001, pp. 174-185
48. Les Gasser and Kelvin Kakugava. MACE3J: Fast Flexible Distributed Simulation of Large, Large-Grain Multi-Agent Systems/www.isrl.uiuc.edu/~gasser/papers/mace3j-aamas02-pap.pdf
49. А.И.Миков, Е.Б. Замятина, А.Н.Фирсов. Инструментальные средства удалённого параллельного моделирования. В кн. Proceedings of XXII International Conference “Knowledge-Dialogue-Solution”.- FOI-COMMERCE, Sofia, 2006, pp. 280-287.
50. Ferenci S., Perumalla K. , and Fujimoto R. An Approach to Federating Parallel Simulators // Workshop on Parallel and Distributed Simulation, May 2000 (http://www.cc.gatech.edu/ computing/ pads/papers.html)
51. Syrjakow M., Syrjakow E., Szczerbicka H. Towards a Component-Oriented Design of Modeling and Simulation Tools// Proceedings of the International Conference on AI, Simulation and Planning in High Autonomy Systems (AIS 2002), Lisbon, Portugal, April 7-10, 2002
52. Миков А.И. Моделирование вычислительных систем. Учебное пособие по спецкурсу.-Пермь, Перм. ун-т.,1982, 96 с.
53. Е.Киндлер. Языки моделирования. - М.:Энергоатомиздат.,1985.-288с.
54. Шрайбер Т.Дж.Моделирование на GPSS.-М.:Маширостроение, 1980.-
55. Дал У.,Мюрхауг Б., Нюгорд К. Симула -67.-М..:Мир, 1969.-100с.
56. Брейер М. Автоматизация проектирования вычислительных систем. Языки, моделирование и базы данных.-М.:Мир, 1979.-464с.
57. Андрианов А.Н., Бычков С.П.,Хорошилов А.И. Программирование на языке Симула-67.-М.:Наука, 1985.-370с.
58. Прицкер. А. Введение в имитационное моделирование и язык СЛАМ II.-М.Мир,-1987 г., 646с.
59. Томашевский В., Жданова Е. имитационное моделирование в среде GPSS.-М.:Бестселлер, 2003-416 с.
60. Т. Дж. Шрайбер, С. Кокс, Дж. О. Хенриксен, П. Лоренц, Дж. Рейтман, И. Стол. GPSS 40 лет: перспективы развития. (Труды конференции WSC-2001, 9-12 декабря 2001 года). Доступно на сайте: www.gpss.ru
61. Якимов И. М., Девятков В. В. Развитие методов и систем имитации в СССР и России. (Казань, ноябрь 2001 год). Доступно на сайте: www.gpss.ru
62. Лычкина Н. Н. Опыт применения GPSS в Государственном Университете Управления. (40-летию GPSS посвящается). (Москва, 2002 год). Доступно на сайте: www.gpss.ru
63. Яковлев С. А. Методические основы использования GPSS в учебном процессе при подготовке дипломированных специалистов по направлениям 654600 – Информатика и вычислительная техника и 654700 – Информационные системы (формат MS Word, архив zip 20 кБ). (Санкт-Петербург, 2003). - опубликовано 17 февраля 2003. Доступно на сайте: www.gpss.ru

64. Мир имитационного моделирования - мир GPSS. (Казань, 2003). - опубликовано 18 февраля 2003. Доступно на сайте: www.gpss.ru
65. Замятина Е.В., Осмехин К.А. Принципы проектирования алгоритма динамической балансировки в распределенной системе имитационного моделирования. Рецензируемый научно-технический журнал «Известия белорусской инженерной академии», 2005. - №1(19)/2 2005. - С.54-55.
66. Миков А.И.., Замятина Е.Б., Осмехин К.А. Метод динамической балансировки процессов имитационного моделирования. В кн. «Материалы Всероссийской научно-технической конференции «Методы и средства обработки информации МСО-2005».г.Москва,3 октября 2005,стр.472-478 г.
67. Бусленко Н.П. Моделирование сложных систем. – М.: Наука, 1978. – 395 с.
68. Клейнен Дж. Статистические методы в имитационном моделировании. – М.: Статистика, 1978, вып.1. – 204 с.
69. Коваленко И.Н. Анализ редких событий при оценке эффективности и надёжности систем. – М.: Советское радио, 1980. – 208 с.
70. Калашников В.В. Организация моделирования сложных систем.- М.:Знание, 1982.- 64 с.
71. Е.Ф.Аврамчук, А.А.Вавилов, С.В.Емельянов и др.; Под общ. ред. С.В.Емельянова и др. Технология системного моделирования/М.: Машиностроение; Берлин: Техник, 1988.- 520 с.
72. Balci O. Verification, validation and accreditation // Proceedings of the 1998 Winter Simulation Conference.- 1998. – pp. 41–48.
73. Sargent R.G. Some approaches and paradigms for verifying and validating simulation models//Proceedings of the 2001 Winter Simulation Conference.- 2001.-pp. 106–114.
74. Law A.M., McComas, M.G. How to build valid and credible simulation models // Proceedings of the 2001 Winter Simulation Conference.- 2001. – pp. 22–29.
75. Carson J.S. Model verification and validation// Proceedings of the 2002 Winter Simulation Conference. – 2002. – pp. 52–58.
76. Balci O. Credibility Assessment of Simulation Results // Proceedings of the 1986 Winter Simulation Conference. – 1986. –pp. 39–44.
77. Christopher D. Carothers, Kalyan S. Perumalla, Richard M. Fujimoto. Efficient Optimistic Parallel Simulations Using Reverse Computation. //Accessed at www.cc.gatech.edu
78. Kevin Jones and Samir R. Das. Combining Optimism Limiting Schemes In Time Warp Based Parallel Simulations. *Proceedings of the 1998 Winter Simulation Conference D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds*
79. Robert G. Sargent. Some Approaches And Paradigms For Verifying And Validating Simulation Models. *Proceedings of the 2001 Winter Simulation Conference B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds*.
80. Patrick F. Riley, George F. Riley**.**Spades — A Distributed Agent Simulation Environment With Software-In-The-Loop Execution *Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice.*
81. Osman Balci,Richard E. Nance,James D. Arthur Expanding Our Horizons In Verification, Validation,And Accreditation Research And Practice *Proceedings of the 2002 Winter Simulation Conference E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.*
82. John A.Miller,Gregory Baramidze. Simulation And The Semantic Web. *Proceedings of the 2005 Winter Simulation Conference M.E.Kuhl,N.M.Steiger,F.B.Armstrong,and J.A.Joines,Eds*
83. Paul A. Fishwick. Using XML for simulation Modeling. Proceedings of the 2002 Winter Simulation Conference E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds
84. Richard M. Fujimoto, Richard M. Weatherly. HLA Time Management and DIS. citeseer.ist.psu.edu/fujimoto95hla.html

85. Perakath Benjamin, Kumar V. Akella, Kaiser Malek, Ronald Fernandes. An Ontology-Driven Framework For Process-Oriented Applications. *Proceedings of the 2005 Winter Simulation Conference M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds*
86. В. А. Пепеляев, Ю. М. Чёрный (Киев). О современных подходах к оценке достоверности имитационных моделей. Доступно на сайте: www.gpss.ru
87. Jefferson, D. (1985). "Virtual Time." ACM Transactions on Programming Languages and Systems 7(3): 404-425.
88. Jefferson, D. R. (1990). Virtual Time II: Storage Management in distributed Simulation. Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing: 75-89.
89. Jha, V. and R. Bagrodia (2000). "Simultaneous Events and Lookahead in Simulation Protocols." ACM Transactions on Modeling and Computer Simulation 10(3):
90. C.D.Pegden. Future Directions in Simulation Modeling. . Proceedings of the 2005 Winter Simulation Conference. 35p.
91. Bagrodia, R., R. Meyer, M. Takai, Y. Chen, Y., X. Zeng, J. Martin and H. Song. 1998. PARSEC: A Parallel Simulation Environment for Complex Systems. IEEE Computer, 3(10): 77-85.
92. Д. Хефлин, Т. Ней. Разработка Web-скриптов. Библиотека программиста. – СПб.: Питер, 2001.
93. S.W. Reichenthal. The Simulation Reference Markup Language (SRML): a foundation for representing BOMs and supporting reuse. – Boeing, 2002.
94. P.A. Fishwick. Using XML for simulation modeling. Из *Proceedings of the 2002 Winter Simulation Conference,* ред. E. Yucesan, C.-H. Chen, J. L. Snowdon, J. M. Charnes, 616-620, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2002.
95. T. Wiedemann. Next generation simulation environments founded on open source software and XML-based standard interfaces. Из *Proceedings of the 2002 Winter Simulation Conference,* ред. E. Yucesan, C.-H. Chen, J. L. Snowdon, J. M. Charnes, 623-640, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2002.
96. G. Qiao, F. Riddick, C. McLean. Data driven design and simulation system based on XML. Из *Proceedings of the 2003 Winter Simulation Conference,* ред. S. Chick, P. J. Sanchez, D. Ferrin, D. J. Morrice, 1143-1148, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2003.
97. R.F. Lu, G. Qiao, C. McLean. NIST XML simulation interface specification at Boeing: a case study. *Proceedings of the 2003 Winter Simulation Conference,* S. Chick, P. J. Sanchez, D. Ferrin, D. J. Morrice ed, 1230-1237, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2003.
98. D.C. Chatfield, T.P. Harrison, J.C. Hayya. XML-based supply chain simulation modeling. Из *Proceedings of the 2004 Winter Simulation Conference,* ред. R .G. Ingalls, M. D. Rossetti, J. S. Smith, B. A. Peters, 1485-1493, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2004.
99. R.Bagrodia, R.Meyer. Parsec: A Parallel Simulation Environment for Complex Systems. Доступно на сайте: doi.ieeecomputersociety.org/10.1109/2.722293
100. Distributed Simulation via a Sequence of Parallel Computations."Communications of the ACM 24(4): 198-205.
101. Chen, G. and B. K. Szymanski (2002). Lookback: A New Way of Exploiting Parallelism in Discrete Event Simulation. Proceedings of the 16th Workshop on Parallel and Distributed Simulation: 153-162.
102. J A. Miller. Simulation And The Semantic Web/ John A. Miller, Gregory Baramidze // Proceedings of the 2005 Winter Simulation Conference. – pp. 2371-2377
103. P. Gustavson. Using Xml And Boms To Rapidly Compose Simulations And Simulation Environments/ Paul Gustavson, Tram Chase // Proceedings of the 2004 Winter Simulation Conference. – pp. 1467-1475

104. Lee Lacy. Potential Modeling And Simulation Applications Of The Web Ontology Language – Owl / Lee Lacy, William Gerber // Proceedings of the 2004 Winter Simulation Conference. – pp. 265-270
105. Dean, M., D. Connolly, F. van Harmelen, et al. 2002. Web Ontology Language (OWL) Reference Version 1.0. W3C. <http://www.w3.org/TR/2002/owl-ref/>
106. Vei-Chung Liang. A Port Ontology For Automated Model Composition / Vei-Chung Liang, Christiaan J.J. Paredis // Proceedings of the 2003 Winter Simulation Conference, - pp. 613-622
107. Zeigler B. P. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems / Zeigler B. P., H. Praehofer, T. G. Kim // 2000 2nd. Academic Press.
108. Mohamed Fayez. Ontologies For Supply Chain Simulation Modeling / Mohamed Fayez, Luis Rabelo, Mansooreh Mollaghasemi // Proceedings of the 2005 Winter Simulation Conference, - pp. 2364-2370
109. Perakath Benjamin. A Model-Based Approach For Component Simulation Development /Perakath Benjamin, Dursun Delen, Richard Mayer, Timothy O.Brien // Proceedings of the 2000 Winter Simulation Conference, - pp. 1831-1839
110. Scott Christley. An Ontology For Agent-Based Modeling And Simulation / Scott Christley, Xiaorong Xiang, Greg Madey
111. Кельтон В. Д., Лоу А. М. Имитационное моделирование. Классика CS. 3-е изд..-СПб.:БХВ-Питер, 2004.-887 с.
112. Карпов Ю.Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 (+CD) ..-СПб.:БХВ-Питер, 2005.-400 с.
113. Б.В.Соколов, Р.М.Юсупов. Концептуальные и методические основы квалиметрии моделей и полимодельных комплексов. Труды СПИИРАН./ Институт информатики и автоматизации, стр.10-35.- СПб.: Наука, 2004,-348 с.
114. Яцкив И. В. Проблема валидации имитационной модели и ее возможные решения. http://gpss.ru/immod'03/043.html
115. Рыжиков Ю.И. Имитационное моделирование. Теория и технологии.-М.:Альтекс, 2004, 384 с.
116. В.В.Окольнишников. Распределённая система имитационного моделирования. Труды Всероссийской науно-практической конференции «Методы и средства обработки информации», МСО-2003, М.: Издательский отдел факультета ВМиК МГУ, М.:МГУ,2005,-стр. 468-474 .
117. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата. Труды Всероссийской науно-практической конференции «Методы и средства обработки информации», МСО-2005, М.: Издательский отдел факультета ВМиК МГУ, М.:МГУ,2005,-стр. 59-76.

Толковый словарь

Accreditation (аккредитация) - официальное признание того, что модель применима для определенной цели.

Accuracy (точность) - степень точности модели с точки зрения результатов моделирования

ALSP (Aggregate Level Simulation Protocol) - стандарт для обеспечения взаимодействия между имитационными моделями (тренажёрами) и переиспользования этих моделей.

As-fast-as-possible simulation - отношение между изменением модельного времени и процессорным временем не имеет значения

Checkpoint - процесс сохранения снимка состояния системы для обеспечения отката.

Continuous simulation (непрерывное моделирование) - процесс моделирования, при котором переменные, описывающие состояние системы, постоянно изменяются во времени.

Deterministic Model (Детерминированная модель) - модель, в которой результаты полностью определяются через известные отношения состояний и событий и в которой данный ввод всегда приводит к одному и тому же выходу.

DIS (Distributed Interactive Simulation) - стандарт для реализации взаимосвязи между различными имитационными моделями, например, тренажёрами**.**

Event driven simulation (событийно-ориентированное моделирование) - имитационное моделирование, управляемое событиями. При управлении моделированием с помощью событий создаётся календарь событий. Каждый элемент календаря событий представляет собой пару (e_i, t_i), где e_i – событие, а t_i – это модельное время, на которое запланировано событие. Алгоритм имитации выбирает из календаря событий событие e_i с минимальным t_i и запускает это событие на выполнение.

Fidelity(точность) - точность представления моделью реальной системы.

GVT (Global Virtual Time) - термин, используемый при оптимистической синхронизации – минимальная метка времени всех необработанных сообщений в системе. Ограничивает снизу время любого последующего отката.

HLA - 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.

LAN (Local Area Network) - локальная компьютерная сеть, обычно соединяет компьютеры, принадлежащие одной организации. Физическое расстояние между узлами обычно 10 километров и менее. Узел такой сети – это обычно рабочая станция, файловый сервер или сервер печати, т.е. относительно маленькая станция, специализирующаяся на особых функциях внутри организации. Главная задача локальной сети – это обычный обмен информацией и разделение ресурсов.

LBTS - Lower Bound of the Time Stamp - нижняя граница временных меток сообщений, которые данный процесс может получить в будущем.

Local causality constraint (LCC) - локальное ограничение причинной связи - ограничение обеспечивает естественный порядок наступления событий от «причины к следствию» в имитационной модели. Это требование гарантирует, что выполнение имитационной модели на распределённой ВС приведёт к тем же результатам, что и на последовательной.

Lookahead - предварительный просмотр (или забегание вперёд) – задержка реакции процесса на сообщение. Используется многими (особенно консервативными алгоритмами синхронизации).

Model Testing (тестирование) - определение того, есть ли в модели неточности или ошибки.

Model Validation (валидация модели) - определение того, что поведение имитационной модели приближает поведение изучаемой системы достаточно точно (accurate) для данной области применения.

Model Verification (верификация модели) - определение того, что модель достаточно точно (accurate) описывает реальную систему, либо достаточно точно переведена из одной формы описания в другую.

Pending event list - список необработанных событий (или список запланированных событий или календарь событий). В дискретном событийно-ориентированном моделировании событие может запланировать другое событие, которое должно быть выполнено через k единиц модельного времени, т.е. в момент времени T + k, где T –текущее модельное время.

Precision (точность) - точность выполнения отдельной операции, которая связана с неточностью машинного представления данных.

Real-time simulation - изменение модельного времени пропорционально, либо ограничено процессорным временем.

Rollback (откат) - восстановление сохраненного состояния системы с отменой всех произошедших с тех пор изменений.

Simulator - устройство, компьютерная программа или система, выполняющая моделирование.

Simultaneous events - одновременные события, т.е. события, которые запланированы на один и тот же момент модельного времени. Зачастую порядок их обработки является важным, т.к. может иметь небольшое, но значительное влияние на общий ход моделирования.

SMP (Symmetric Multi Processing) - симметричные мультипроцессорные системы. Для объединения процессоров используют способ их объединения на единой симметрично адресуемой общей памяти.

Static simulation - представление системы в конкретный момент времени, или моделирование процесса, в котором время роли не играет.

Stochastic Model (Стохастическая (вероятностная) модель) - модель, результаты которой, определяются при помощи одной или более случайной переменной, что отражает неполное знание о процессе, либо в которой данный вход формирует выход согласно некоторому вероятностному распределению.

Time stepped simulation (пошаговое моделирование) - процесс моделирования, при выполнении которого происходит продвижение модельного времени на фиксированную величину Δt.

WAN (Wide Area Network) - глобальная компьютерная сеть, обычно соединяет компьютеры, принадлежащие различным организациям (предприятия, университеты и т.д.). Физическое расстояние между узлами обычно составляет 10 километров и более. Каждый узел такой сети – это законченная компьютерная система, включающая всю периферию и значительное количество прикладного программного обеспечения. Главная задача глобальной сети – это обмен информацией между пользователями различными узлов.

Временная метка (Time Stamp) - момент времени, на которое запланировано выполнение некоторого события.

Имитационная модель(simulation) - система, которая имитирует поведение физической системы.

Каузальность (causality) - зависимость одного события от другого. Событие e_2 каузально зависит от e_1, если e_1 имеет временную метку, меньшую, чем у события e_2 и e_1 изменяет состояние имитационной модели, например значение некоторой переменной, которая прямо или косвенно используется в e_2.

Компьютерная сеть - это набор компьютеров, соединенных коммуникационными средствами, с помощью которых компьютеры могут обмениваться информацией. Обмен информацией выполняется при посылке или при получении сообщения. В зависимости от расстояния между компьютерами и их принадлежностью, компьютерные сети называются либо глобальными, либо локальными.

Консервативная синхронизация (conservative synchronization) - механизм, не позволяющий процессам, участвующим в распределённом моделировании, обрабатывать те сообщения, временные метки которых не соответствуют хронологическому порядку. В системах моделирования с консервативной синхронизацией обрабатываются только те события, о которых можно с уверенностью сказать, что в будущем они не получат сообщения с меньшим временем.

Механизм (алгоритм) синхронизации - механизм (протокол), обеспечивающий процессам, участвующим в распределенном моделировании сохранение LCC.

Модельное время (Simulation Time) - представление физического времени в модели. Так работу предприятия в модельном времени можно представить отрезком времени [8.00,17.00], за единицу модельного времени (Δt) можно принять временной интервал в 1 минуту, в 10 минут, в 30 минут, в один час и т.д.

Оптимистическая синхронизация (optimistic synchronization) - в отличие от консервативного алгоритма, позволяет обрабатывать любые события, но при обнаружении нарушения LCC, восстанавливает одно из предыдущих состояний системы, при помощи механизма откатов.

Процессорное время (Wallclock time) - время работы симулятора на компьютере. Так, например, моделирование предприятия может занять 1 час работы на компьютере. Иногда (при использовании тренажёров) продвижение модельного времени должно быть синхронизировано с процессорным.

Состояние и Событие - состояние, событие и время – три наиболее фундаментальных концепций в моделировании систем. Состояния и события – дополняют друг друга, т.к. изменение состояния системы происходит только как результат наступления события. **Состояние** описывает систему в определенный

промежуток времени. **Событие** – момент времени, обозначающий смену состояний.

Технология HLA - 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.

Федерат - одна или набор взаимодействующих систем имитации.

Федерация - соглашения одного или нескольких федератов, действуя совместно в распределённом моделировании, добиться одной и той же цели**.**

Физическая система - реальная или воображаемая система, интересующая исследователя.

Физическое время (***Physical Time)*** - время, которое используется в реальной (физической) системе, которую моделируют. Например, мы моделируем работу некоторого предприятия в течение рабочего дня с 8.00 до 17.00.