ПОДХОДЫ К ПОСТРОЕНИЮ СИСТЕМ АГЕНТНОГО МОДЕЛИРОВАНИЯ

Митраков А.А.
Пермский государственный национальный исследовательский университет
Пермь, Россия

APPROACHES TO AGENT-BASED SYSTEMS DESIGNING

Mitrakov A.A.
Perm State National Research University
Perm, Russia

Аннотация

В статье предлагается формальное описание модели системы распределённого имитационного моделирования на базе агентного подхода. Акцент в работе делается на математическую модель, основанную на теории множеств. Предлагаемый подход является математически строгим, однако достаточно гибким в том плане, что на его основе можно предложить множество различных реализаций. Также в работе представленные проблемы и задачи, которые могут возникнуть в процессе разработки агентной платформы.

Abstract

This is a research that describes the formal model of a distributed agent-based simulation system. The attention is drawn to the mathematical model based on the set theory. This approach is mathematically rigorous and therewith rather flexible to get the different platform implementations possible. Also the paper shows different issues related to the process of the agent-based platform designing.

Введение

В данной статье предлагается один из возможных подходов к формальному описанию системы распределённого агентного моделирования. В первую очередь основную сложность представляет описание непосредственно модели задачи, поскольку она отражает все характеристики, ограничения и допустимые операции самой агентной платформы. Построить математическую модель для произвольной имитационной модели, определить её свойства, структуру и семантику – задача, безусловно, нетривиальная, поскольку мы имеем дело с понятием метамодели.

Основная идея способа представления модели – отражение её свойств и компонентов с различных точек зрения. В нашем случае мы имеем дело с распределённой агентной платформой на базе дискретно-событийного моделирования. Это, в свою очередь, приводит к тому, что потребуется рассмотреть модель как минимум с трёх различных аспектов: с точки зрения агентного подхода (модель как совокупность взаимосвязанных агентов), с точки зрения классического дискретно-событийного моделирования (модель как набор логических процессов) и с точки зрения параллельных вычислений (модель как множество узлов вычислительной сети).

При переходе к формальному описанию модели в иерархическом структурированном виде следует иметь в виду, что все её компоненты взаимосвязаны и взаимозависимы. Поэтому

также стоит изучить свойства субмоделей, варианты их наложения и связанные с ними проблемы и задачи.

Поэтому помимо описания различных субмоделей в статье будут предложены варианты наложения моделей, а также сопутствующие задачи и возможные ограничения. Предполагается, что читатель владеет такими понятиями, как теория графов, онтологии, агентной подход, системная динамика, формальные грамматики, имитационное моделирование и т.д.

Существующие работы и состояние области исследования

В целом, данной области исследования посвящено немало работ, преимущественно зарубежных. Также существует множество готовых программных продуктов и реализаций. К ним, например, относят как инструменты для организации агентного моделирования (Swarm, NetLogo, RePast, Mason, Cougar, JAMES, AnyLogic) [1, 2 7], так и средства создания мультиагентных систем (Jade, MACE3J) [2].

Если рассматривать сферу распределённого моделирования, то к настоящему времени специалисты выделяют два подхода к организации параллельных средств машинной имитации [4]:

- монолитные системы, в которых моделирование сводится к взаимодействию совокупности логических процессов;
- программные компоненты, которые позволяют объединить существующие средства имитации в одну общую сеть для реализации параллельного моделирования.

К последним относят технологию HLA (High Level Architecture), которая де-факто объявлена стандартом распределённого моделирования в США. К агентным системам, поддерживающим HLA, относят “HLA-AGENT” и “SIMAGENT TOOLKIT” [2].

Также существует множество работ, предлагающих особые алгоритмы и методики для реализации систем моделирования. К таковым относят, например, применение модели акторов для уменьшения накладных расходов коммуникации агентов [9], модель Штрассбургера [2], использующая эффективные алгоритмы синхронизации и вычисления GVT, распределённая система “Мера” [8], отечественная разработка для распределённого моделирования многопроцессорных ВС “Triad”[3-5].

Описание системы распределённого агентного моделирования

Прежде чем переходить к формальному представлению, попробуем описать модель на общедоступном интуитивно понятном уровне. Это предоставит общее понимание системы и поможет выявить сложности и проблемы, связанные с разработкой подобного рода платформы.

Итак, предлагаемая система представляет собой программно-аппаратный комплекс для проектирования, разработки, реализации, эксплуатации и сопровождению агент-ориентированных имитационных моделей. Подразумевается, что модель запускается в гетерогенной вычислительной сети, где происходит имитационный прогон, по результатам которого собирается статистика.

Общий цикл работы с точки зрения исследователя выглядит следующим образом:

- описание модели;
- настройка конфигурации сети;
- задание параметров модели;
- запуск;
- получение статистики (и, возможно, дальнейшая постобработка).

Данная статья посвящена, прежде всего, первому этапу. Т.к. от чёткой спецификации модели зависит качество моделирования, исследователю в первую очередь следует сосредоточиться на полном, точном и непротиворечивом описании модели, а разработчику платформы – уделить особое внимание проработке этого аспекта.

Основная задача исследователя – описать структуру и поведение агентов. Суть агентного моделирования состоит в том, что локальное поведение агентов, работающих по своим собственным правилам, формирует глобальное поведение системы в целом (концепция проектирования «снизу-вверх»). Это отличается от традиционных подходов типа «сверху-вниз», когда заданы глобальные законы функционирования системы, на базе которых работают её элементы.

Например, в системной динамике исследователь указывает интенсивность потока заявок, задержки при работе обслуживающих устройств и т.п. в виде математических формул (например, в виде стационарного пуассоновского потока, распределение которого выражается в виде $P(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda x}$). Подобные предположения и допущения, несомненно, ухудшают качество модели.

Агентное моделирование является более прогрессивным в том плане, что глобальное функционирование системы заведомо неизвестно исследователю. При описании агента следует задать лишь локальные правила поведения, методы взаимодействия и коммуникации с другими агентами и т.д. Легко показать, что даже два-три простейших правила уже могут привести к весьма разнообразным формам поведения в группе агентов. Примером может послужить boids-моделирование и теория клеточных автоматов.

Представление модели. Общие сведения

Итак, модель, очевидно, должна содержать сведения о структуре, поведении и семантике агентов. Далее следует учесть, что ввиду логической обособленности агентов единственным способом коммуникации между ними является обмен сообщениями (данный термин нужно понимать в широком смысле, а не в плане конкретной программной реализации). Следовательно, модель должна содержать сведения о структуре и способах передачи сообщений.

Идея агентного подхода получила широкое распространение лишь в середине 90-х, но совсем не в силу своей новизны и «революционности», а благодаря развитию вычислительных мощностей компьютера (а также методов искусственного интеллекта). Дело в том, что масштабирование модели, скажем, супермаркета, выполненной в классической парадигме («сверху-вниз»), до размеров миллионного города не сильно скажется на производительности. Агентный же подход предполагает вычисление каждого агента в

отдельности, поэтому масштабирование с супермаркета до миллионного города пропорционально увеличит число агентов и экспоненциально – суммарное число связей между ними.

Поэтому следующим этапом служит реализация алгоритмов распределённого моделирования [6]. Какой бы производительной не была аппаратура программно-аппаратного комплекса, задачу можно сколь угодно точно детализировать, и сколь угодно сильно расширять, что неблагоприятно скажется на скорости моделирования. Поэтому возможность параллельных вычислений закладывается в платформу априори.

Следовательно, в модели должны быть заложены данные о структуре вычислительной сети, алгоритмах пересылки сообщений и т.д. Причём модель должна обладать определённой гибкостью в том плане, что выход одного вычислительного узла в составе кластера из строя не повлечёт полного переписывания структуры агентов, изменения связей между ними и т.д. Другими словами, концептуальная агентная модель не должна зависеть от вычислительной сети.

Ещё одной проблемой при переходе к распределённым вычислениям является нарушение так называемой «локальной каузальности». Суть проблемы кроется непосредственно в природе дискретно-событийного моделирования. Дело в том, что логические процессы в ходе своей работы изымают события из списков событий и обрабатывают их, изменяя при этом состояния связанных с ними объектов (в нашем случае агентов). При этом могут порождаться новые события как для собственного, так и для других логических процессов. Если событие планируется для соседнего процесса, ему отправляется сообщение с соответствующей временно́й меткой.

Итак, проблема локальной каузальности возникает тогда, когда логическому процессу в момент времени t_1 приходит сообщение с временной меткой t_2, и при этом $t_1 > t_2$. В этом случае нарушается естественная хронология обработки событий, в результате чего нарушается смысл моделируемых во времени процессов. Вариантом решения проблемы могут выступать различные алгоритмы по синхронизации времени [1, 2, 5, 6]. При формальном описании модели, несомненно, также стоит учитывать этот момент.

Ранее было замечено, что развитие агентного моделирования обязано увеличению мощности вычислительной техники и развитию искусственного интеллекта. Следующее, что мы должны учесть в модели, – это наличие интеллектуальных средств, помогающих агенту опознать ту или иную ситуацию, и принять правильное решение. При этом могут использоваться методы для решения задач регрессии, классификации, кластеризации, распознавания образов, семантического поиска, машинного обучения и т.д. В этой статье мы лишь упомянем наличие таких механизмов, однако подробнее мы их рассмотрим позже.

Также для повышения интеллектуальности могут применяться методы управления знаниями на основе онтологического инжиниринга. Введение в модель знаний из предметной области, как мы покажем, имеет массу преимуществ: от удобства работы исследователя до интеллектуального управления балансировкой и синхронизацией времени.

Итак, мы увидели, что представление модели в случае основанного на знаниях распределённого агентно-ориентированного дискретно-событийного моделирования действительно является нетривиальным. Чтобы привести всё вышесказанное в порядок и

ввести систематизацию в определение сущности модели, следует сформулировать точное математическое определение этого понятия. Мы увидим в дальнейшем, что определение модели, во-первых, будет иерархическим, а во-вторых, в нём не будут содержаться все детали и тонкости реализации. Наоборот, оно будет весьма обобщённым, что позволит спроектировать некую абстрактную агентную платформу, на базе которой можно будет легко предложить массу различных реализаций.

Представление модели. Теоретико-множественное обоснование

В данном разделе мы предложим формальное представление модели, используемой системой агентного моделирования. Для этого сразу положим, что модель содержит 3 составляющие: модель вычислительной сети (CNM), модель логических процессов (LPM), модель концептуального уровня (CLM):

$$Model = (CNM, LPM, CLM).$$

Все они классифицируются по степени зависимости от задачи и предметной области, а также по зависимости от программно-аппаратного комплекса. Далее рассмотрим каждую из этих моделей в отдельности, а также предложим варианты их сочетания друг с другом.

Модель вычислительной сети (CNM)

Наиболее простой по своей структуре является модель вычислительной сети (*Computing Network Model*). Она представляется связным неориентированным графом:

$$CNM = (V^C, E^C)$$

Вершинам узлов соответствуют вычислительные узлы, рёбрам – связи между ними. Конфигурация вершин зависит от типа ВС – это могут быть как процессоры в сильносвязанных SMP или MPP-системах, так и отдельные компьютеры в составе кластера. В качестве связей подразумевается установление постоянного логического соединения типа точка-точка (и сопутствующего стека протоколов). Рёбрам графа могут быть приписаны различные характеристики как числового, так и категориального характера (пропускная способность, латентность, таймаут соединения и т.д.) Топология сети может быть произвольной.

Модель вычислительной сети является основой для построения остальных 2 моделей. Исследователю при формализации задачи требуется проработать топологию сети в соответствии с требованиями объёма вычислительных затрат и интенсивности пересылки данных. Очевидно, при интенсивной пересылке больших объёмов данных системы с общей памятью будут иметь преимущество перед теми же кластерами.

Однако, несмотря на необходимость проработки модели вычислительной сети под каждую задачу, в целом подразумевается, что *CNM* не зависит от задачи и/или предметной области, т.е. предполагается, что она является наиболее абстрактной по сравнению с концептуальной моделью и моделью логических процессов.

Следует сразу оговорить, что исследователю не всегда придётся задумываться о топологии сети. Во-первых, это связано с ограниченностью ресурсов (компьютеров в

кластере), а во-вторых, уже упоминалось, что концептуальная модель сама может служить основой для построения рекомендуемой топологии сети (за счёт метазнаний).

Модель логических процессов (LPM)

Модель логических процессов представляет собой следующую ступень абстракции после модели вычислительной сети. Поскольку в основе платформы лежит идея дискретно-событийного моделирования, основными акторами системы являются логические процессы.

Итак, данная модель представляется следующим образом [3]:

$$LPM = (T, \Sigma, L, M, \Theta),$$

где

- T – абстрактное линейно упорядоченное множество с отношением порядка «$\preceq$». Далее под T будем понимать множество моментов моделируемого времени.
- Σ – Множество событий – функциональное множество, для каждого из элементов которого ставится в соответствие временная метка из T. Вообще говоря, элементы множества представляют собой отображения, при которых меняется состояние логического процесса: $\Sigma = \{\delta_t | \delta_t : State \to State\}$.
- L – Множество логических процессов – представляется в виде ориентированного графа $L = (V^L; E^L)$. Вершины графа – логические процессы, дуги – связи между ними.
- M – множество сообщений, передаваемых между логическими процессами. В общем случае под сообщением будем понимать следующую структуру:

$$m_t = (S, D, \delta_t, \mu),$$

где $S \in V_L$ – процесс-отправитель,
$D \in V_L$ – процесс-получатель,
δ_t – передаваемое событие с временно́й меткой t,
μ – тип сообщения (метаданные).

О необходимости введения метаданных будет пояснено позже в разделе «Модель концептуального уровня».

- Θ – Механизм передачи сообщений – совокупность алгоритмов, согласно которым производится передача сообщений между логическими процессами и обеспечивается синхронизация времени.

Мы уже описывали суть дискретно-событийного моделирования. Более формально этот процесс описывается следующим образом: логические процессы изымают события из списков событий в соответствии с временной меткой t. Затем выполняется вычисление функции δ_t, в результате чего логический процесс переходит из одного состояния *State* в другое. Следует отметить, что новое состояние необязательно является согласованным и непротиворечивым. Существуют специальные алгоритмы разрешения таких ситуаций.

Каждый логический процесс V^L обладает, как минимум, тремя составляющими:

1) *State* – состояние;
2) *InputBuffer* – буфер для входящих сообщений;
3) *OutputBuffer* – буфер для исходящих сообщений.

Далее в разделе «Модель концептуального уровня» понятие логического процесса будет расширено с помощью четвёртого элемента

4) MemoryBuffer – локальная память логического процесса.

Модель логических процессов *LPM*, очевидно, уже больше зависит от задачи и конкретной предметной области, поскольку она является связующим звеном между моделями *CNM* и *CLM*.

Модель концептуального уровня (CLM)

Модель концептуального уровня является наименее абстрактной. Она зависит как от самой задачи, так и от предметной области в целом. Здесь исследователь уже оперирует конкретными субъектами моделирования – агентами.

Итак, модель на концептуальном уровне представляется следующим образом:

$$CLM = (G, \Omega, \Psi),$$

где G – агентный граф (в общем случае ориентированный мультиграф $G = (V^A, E_1^A, E_2^A, \dots, E_n^A)$);

Ω – среда моделирования ($\Omega = (R, \Phi)$);

Ψ – онтология предметного уровня.

1) На тип графа G не накладываются ограничений, но предполагается, что он может поддерживать множество типов дуг (мультиграф). Следует понимать, что для моделирования сложных трудноформализуемых задач требуется минимизировать число допущений, накладываемых на структуру графа. Так, например, граф может быть:

- циклическим (допускается наличие циклов),
- несвязным (допускаются наличие вершин без связей),
- псевдографом (допускается наличие дуг типа (V_i, V_i), т.е. дуг со степенью = 1),
- метаграфом (допускаются связи множества вершин с множеством),
- гиперграфом (допускаются наличие дуг типа $(V_1, V_2 \dots V_n)$, т.е. дуг со степенью $\neq 2$).

Поддержка множества типов связей требуется априори, поскольку коммуникация агентов предполагает введение различных сложных отношений между ними.

2) Среда моделирования Ω представлена двумя множествами:

$$\Omega = (R, \Phi).$$

R – множество ресурсов – конечное множество элементов произвольной природы, потенциально полезных для агентов. Примерами могут послужить товары при моделировании работы супермаркета, материалы на складе при моделировании цепочек поставок и т.д. Для системной динамики аналогом служат ***уровни***.

Φ – множество правил (ограничений и функциональных отображений), определённых для среды моделирования. Для системной динамики аналогом служат ***потоки***, однако множество Φ следует понимать гораздо шире. Упрощённо говоря, Φ характеризует множество глобальных законов, описываемых на некотором языке.

Для этого могут применяться:

- правила на языке исчисления предикатов,
- функции распределения,

- системы алгебраических уравнений,
- системы дифференциальных уравнений,
- дискретно заданные функции,
- булевы функции,
- математические неравенства,
- регулярные выражения,
- характеристические функции,
- рекуррентные соотношения,
- утверждения на языке регулярных грамматик и т.д.

Выбор того или иного средства представления правил зависит от конкретной задачи, а также от сложности вычислительного процесса. Очевидно, проверять, удовлетворяет ли поведение агента булевой функции несколько проще, чем системе интегральных уравнений.

3) Ψ – онтология предметного уровня. Вообще говоря, Ψ может включать и предметно-независимые онтологии более верхних уровней (например, для решения целого класса задач). Основное применение онтологий в предлагаемой агентной платформе заключается в следующем:

- для спецификации знаний, используемых интеллектуальными компонентами агентов;
- для интеллектуального управления синхронизацией времени при распределённом моделировании;
- для управления ходом балансировки загрузки.

На определение онтологий в общем случае не накладывается ограничений, поэтому будем понимать Ψ в классической трактовке:

$$\Psi = (P, N, I),$$

где P – множество концептов; N – множество связей между ними; I – способы интерпретации концептов из P посредством связей N.

В зависимости от конкретной задачи и предметной области определение онтологии может быть расширено. Примером Ψ может послужить онтология, описывающая понятия из отрасли сбыта товаров в супермаркете. В этом случае исследователю при описании задачи не потребуется вводить понятия и определения, связанные с товарами, кассами, безналичными платежами и т.д. При использовании онтологий как исследователь, так сами агенты (на программном уровне) разделяют одни и те же знания о предметной области.

Подробнее об идее использования знаний в управлении процессом моделирования см. в работе [1].

Агентный граф G

Агентный граф, будучи основным компонентом агентно-ориентированной платформы, нуждается в более детальном рассмотрении. Одной из основных задач при разработке агентного графа является сокращение так называемого «семантического разрыва» между предметно-зависимыми и независимыми компонентами: исследователь должен иметь возможность удобным образом задать определение графа, используя лишь понятия из предметной области.

Итак, граф *G*, прежде всего, условно делится на абстрактный и конкретный.

Абстрактный агентный граф – мультиграф, в вершинах которого находятся прототипы агентов, реализующие общие родовидовые характеристики моделируемых сущностей. Связи между прототипами задаются статически в соответствии с условиями задачи.

$$G^A = (V^A; \{E^A\}),\ \ \{E^A\} = \{E_1^A, E_2^A \dots E_n^A\}.$$

Конкретный агентный граф – совокупность зависимых от времени графов, в вершинах которых находятся конкретные экземпляры агентов-прототипов, а связи характеризуют текущие отношения между ними.

$$G^S(t) = (V^S(t);\ \{E^S\}),\ \ \{E^S\} \subseteq \{E^A\},$$

причём все вершины конкретного графа связаны с вершинами абстрактного графа отношением «класс-экземпляр»: $(\forall t\ \forall g \in V^S(t)\ \ \exists p \in V^A: isInstance(g, p))$.

Главная идея абстрактного агентного графа – декларативно описать виды агентов и связи между ними. Конкретный же агентный граф служит математической абстракцией состояния модели в ходе имитационного прогона (и строится лишь в динамике).

Примером G^A в случае супермаркета может послужить следующий граф:

$$G^A = (V^A;\ E_1, E_2, E_3),$$

где $V^A = (\text{Покупатель, Кассир, Товаровед, Фасовщик, Администратор, Охранник})$;

$E_1 = \text{Обращение покупателя к фасовщику;}$

$E_2 = \text{Обращение покупателя к кассиру;}$

$E_3 = \text{Производственные отношения между работниками зала;}$

Тогда конкретным агентным графом будут реальные экземпляры покупателей и работников зала, вступающие в одно или несколько отношений $\{E_1, E_2, E_3\}$. Причём этих экземпляров может быть как один (охранник), так и много (покупатель).

Понятие агента

В вершинах агентного графа находятся агенты (далее под агентами будем понимать агентов-прототипов). Так как в области имитационного моделирования и МАС не устоялось чётко определённого понимания сущности агента, существует масса определений и трактовок.

Одно из наиболее кратких определений: *агент* – это аппаратная или программная сущность, способная действовать в интересах достижения целей, поставленных перед ним пользователем. Более подробно об определениях агента см в [7], а мы дадим собственное понимание в контексте того, что интеллектуальность не присуща программным объектам априори. Так, агент представлен следующим образом:

$$v^A = (Q, F, \lambda, Z),$$

где Q – состояние агента (в простейшем случае ассоциативный массив);

F – функции, преобразующие состояние агента ($F: Q \to Q$);

λ – функции высшего порядка, преобразующие функции ($\lambda: Q \times F \to F$);

Z – локальная память агента (также можно полагать ассоциативным массивом).

Для системной динамики аналогом *F* служат ***вентили***.

Данный подход, нетрудно заметить, не даёт никакого упоминания об интеллектуальности

и обучаемости агентов. Цели и убеждения скрываются во множестве Q, а методы принятия решений – в F. Способность к обучению – т.е. трансформации текущих правил вывода – закладываются в λ и Z. Также следует иметь в виду наличие «внеагентных» параметров – множества правил Φ, скрытое в окружающей среде, и онтологические знания Ψ, существующие обособленно как от агентов, так и среды.

Данное определение, несмотря на рассмотренный выше недостаток, является довольно гибким: с его помощью можно построить как реактивных агентов (действующих по простым правилам, и зачастую не имеющих функций преобразования функций), так и интеллектуальных – подходящих под понимание агентов в широкоупотребительном смысле.

В Таблица 1 представлены все 3 модели в обобщённом виде.

Таблица 1. Модели различного уровня

CNM – модель вычислительной сети	***LPM – модель логических процессов***	***CLM – концептуальная модель***
V^C – множество выч. узлов	T – множество моментов времени	$G = (G^A E^A)$ – агентный граф
E^C – множество связей	Σ – множество событий	$\Omega = (R, \Phi)$ – окружающая среда
	$L = (V^L E^L)$ – граф лог. процессов	$\Psi = (P, N, I)$ – онтология
	M – множество сообщений	
	Θ – механизм передачи сообщений	

Взаимодействие моделей

Общая схема взаимодействия субмоделей изображена на рис. 1. Напомним, что модель представляет собой иерархическую совокупность трёх составляющих:

$$M = (CNM, LPM, CLM)$$

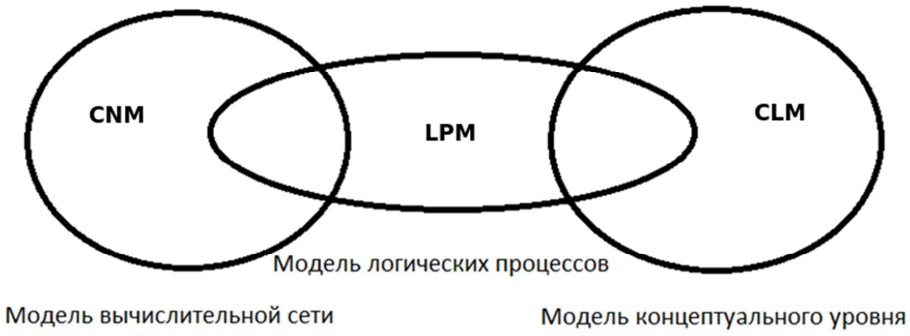

Рис. 1. Взаимодействие моделей

Рассмотрим сначала, как эти модели накладываются друг на друга, а затем попробуем определить задачи, возникающие на каждом из этих уровней.

Прежде всего, введём операцию “наложения” моделей [3]. Под наложением будем понимать бинарную операцию, операндами которой является модель (субмодель), а результатом – также модель, задаваемая взаимно однозначным отображением

$$\mathfrak{A}: Model \times Model \longrightarrow Model.$$

Далее данную операцию будем обозначать знаком «↔».

Связь модели вычислительной сети и модели логических процессов

Наложение моделей *CNM↔LPM* не вызывает особых затруднений. В простейшем варианте вычислительные узлы связаны с логическими процессами посредством агрегации. Основные свойства наложения моделей перечислены ниже:

- на одном узле могут выполняться несколько логических процессов;
- логические процессы могут быть связаны лишь в одном из 2-х случаев:
  - Они находятся на 1-м узле;
  - Существует связь между узлами в CNM;
- связи между логическими процессами направленные;
- граф должен оставаться связным.

Строго говоря, граф модели вычислительной сети отображается в редуцированный ориентированный метаграф модели логических процессов. Во-первых, это значит, что некоторые дуги опускаются, а во-вторых, новый граф соединяет множество с множеством.

Механизм передачи сообщений также проецируется на CNM. Поскольку вычислительные узлы по-прежнему связаны одиночной связью по типу точка-точка, то было бы крайне неэффективно заводить отдельные сетевые соединения для каждого логического процесса. Поэтому рекомендуется следующий вариант отображения:

Для каждого ребра из графа CNM ставится в соответствие 1 дуга (либо 2 взаимно противоположно направленные дуги) из графа LPM.

При этом для каждого узла во множестве логических процессов появляется дополнительная вершина-посредник (назовём её *почтовый агент*), которая будет заниматься мультиплексированием и пересылкой сообщений по общему каналу (рис. 2).

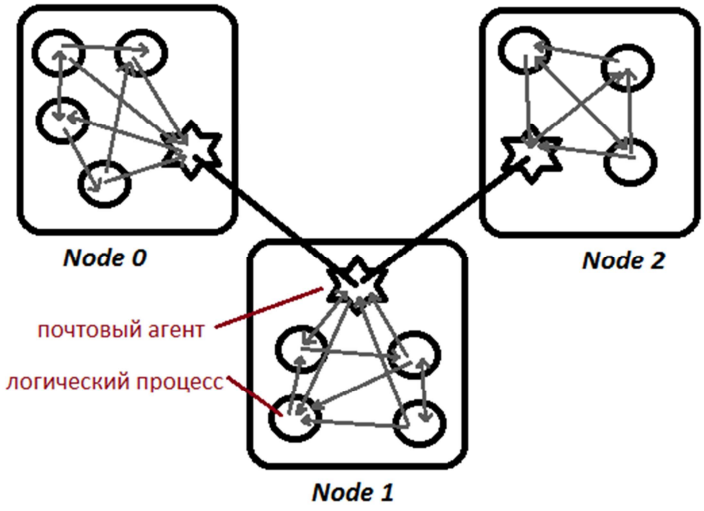

Рис. 2. Наложение модели логических процессов на модель вычислительной сети

Задачи, возникающие при наложении CNM↔LPM

Главной проблемой, возникающей при наложении модели вычислительной сети на модель логических процессов, является размещение всех процессов по вычислительным узлам при $CARD(V^C) < CARD(V^L)$, т.е. когда процессов больше, чем компьютеров. В этом случае мы имеем дело с классической задачей балансировки.

Выделяют два основных типа балансировки [5]:

1. Статическая балансировка (задаётся до начала имитационного прогона).

2. Динамическая балансировка (выполняется в ходе имитационного прогона).

С точки зрения модели балансировки можно также выделить следующие категории:

- Балансировка выполняется вручную исследователем.
- Балансировка выполняется динамически на основании текущей загрузки узлов сети.
- Балансировка выполняется динамически на основании собранной ранее статистики.
- Балансировка выполняется динамически на основе знаний из модели концептуального уровня.

Наилучшими являются два последних варианта. В одном случае собирается статистика по нескольким имитационным прогонам, после чего принимаются более адекватные решения балансировки. Во втором случае система из заложенных в неё знаний о предметной области сама определяет, что, например, охранник в супермаркете не занимает много вычислений, поэтому связанный с ним логический процесс можно переместить на более слабый узел.

Связь модели логических процессов и модели концептуального уровня

Данный тип наложения моделей является более сложным. Здесь требуется связать агентный граф G, окружающую среду Ω и онтологии Ψ с множеством связанных логических процессов. Как и прежде, существует много вариантов (распределить множество агентов по каждому логическому процессу, распределить каждого агента фрагментарно по нескольким логическим процессам и т.д.). Мы предложим следующий способ (рис. 3):

1. Абстрактный агентный граф G^A доступен каждому логическому процессу. Это значит, что каждый процесс имеет сведения о структуре, поведении и семантике концептуальной модели.

2. Множество вершин конкретного агентного графа G^S распределяется по всем процессам равномерно (согласно некоторой метрики). В итоге одному процессу соответствует множество агентов-экземпляров.

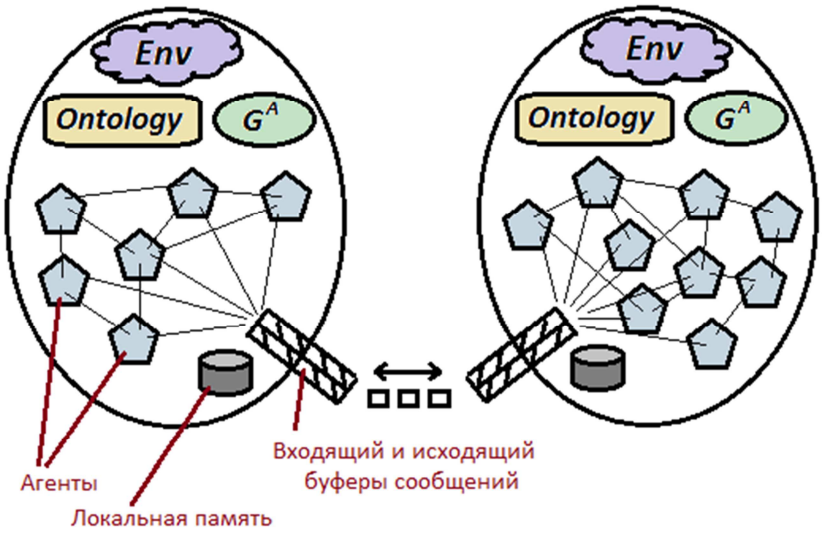

Рис. 3. Наложение модели концептуального уровня на модель логических процессов

LPM↔CLM сохраняет все свойства наложения *CNM↔LPM*, и помимо этого добавляет ещё несколько специфических:

- Состояние логического процесса определяется множеством состояний вершин конкретного агентного графа ($State = State(Q_1 Q_1 \dots Q_n)$).
- При наложении агентного графа G^A на граф логических процессов L для каждой дуги из G^A должен существовать ориентированный путь в L (т.е. если агенты соединены логически, то они должны быть соединены физически).
- Поскольку G^S как мультиграф трансформируется в обычный ориентированный граф L, всем передаваемым сообщениям приписывается доп. метка, означающая тип связи.

3. Окружающая среда Ω и онтологии Ψ подобно G^A распределяется между всеми логическими процессами в виде локальных копий. Следует отметить, что если Ψ полагается статичной и не меняется во время моделирования, то Ω может изменяться (агенты ввиду своей активности способны изменять состояние среды). Поэтому следует также иметь в виду необходимость алгоритмов синхронизации.

Вообще говоря, онтология Ψ не обязана быть статичной: вполне возможно пополнение Ψ новыми фактами прямо в ходе имитационного прогона, ведь совокупное поведение агентов выводит новые знания об исследуемой системе в целом. Однако для простоты оставим это допущение и будем полагать Ψ изменяемой лишь извне.

Задачи, возникающие при наложении LPM↔CLM

В данном случае следует оговорить следующие возможные проблемы:

1. Балансировка нагрузки.
2. Проблема поиска минимального маршрута между двумя связанными агентами.
3. Синхронизация времени между логическими процессами.
4. Синхронизация состояния среды моделирования.

О балансировке уже упоминалось в разделе «*CNM↔LPM*». В данном случае задача ставится схожим образом, только теперь требуется распределять вершины конкретного агентного графа по различным логическим процессам, что в целом сложнее, чем в предыдущем случае.

Если рассматривать проблему поиска минимального маршрута между двумя агентами, то мы получаем задачу минимизации количества транзитивных путей между ними. Другими словами, если агент-экземпляр A связан с агентом-экземпляром B, то они должны лежать в рамках одного или, по крайней мере, смежных логических процессов. Если же для этого потребуется путь из большого числа процессов, то интенсивная пересылка сообщений, очевидно, становится узким местом.

Проблема синхронизации времени логических процессов была подробно освещена в разделе «Представление модели. Общие сведения». Однако следует учитывать одно существенное ограничение, накладываемое требованиями оптимистических алгоритмов: должна быть возможность откатить состояние процесса при нарушении локальной каузальности. В связи с этим следует ещё раз вернуться к нашему определению агента:

$$v^A = (Q, F, \lambda, Z),$$

где Q – состояние агента;

F – функции, преобразующие состояние агента ($F: Q \to Q$);

λ – функции высшего порядка, преобразующие функции ($\lambda: Q \times F \to F$);

Z – локальная память агента.

При использовании оптимистических алгоритмов синхронизации потребуется наложить, по крайней мере, одно из следующих ограничений:

1. Состояние Q должно быть обратимым (т.е. должна быть возможность «мгновенного снимка» состояния агента, которое потом можно было бы восстановить);
2. $\forall f \in F \quad \exists g = f^{-1}: \quad \frac{T(f)}{T(g)} = O(1)$, (т.е. должна быть возможность легко выполнить обратные вычисления для организации отката).

Под выражением $\frac{T(f)}{T(g)} = O(1)$ имеется в виду тот факт, что функция $g=f^{1}$ должна быть того же порядка сложности, что и сама функция *f*.

Соблюдение данных условий предоставляет возможность откатить состояние агента на предыдущее согласованное непротиворечивое состояние. В случае мгновенных снимков от логического процесса требуется наличие локального буфера (тем самым мы проигрываем по памяти). В случае обратных вычислений мы, очевидно, проигрываем по процессорному времени, однако гораздо бóльшую проблему привносит тот факт, что f^{-1} может не существовать (либо она будет трудновычислимой). Примером могут послужить односторонние функции (например, факторизация простых чисел), либо функции, содержащие операции ввода-вывода.

О синхронизации среды моделирования упоминалось выше. Поскольку её состояние может изменяться агентами, следует позаботиться о пересылке состояния множества R между логическими процессами. На механизме передачи сообщений это отразится введением ещё одной метки типа сообщений. Как и в предыдущем случае, внезапное обновление среды Ω также может послужить причиной нарушения локальной каузальности.

Подробнее об оптимистических алгоритмах синхронизации см. в [2, 5, 6].

Заключение

В заключение хотелось бы подвести некоторые итоги. Заметим, что агентное моделирование сделало шаг вперёд по сравнению с другими видами имитационного моделирования именно за счёт возможности исследования неформальных (или слабоформализуемых) предметных областей. Это, в свою очередь, является следствием сложности и в некоторой степени противоречивости требований, предъявляемых к агентной платформе.

Сложность этих требований приводит к соответствующим трудностям при описании модели. Языки классического моделирования (например, GPSS) существуют до сих пор, несмотря на то, что они теоретически не способны описать модель с достаточной степенью детализации. Агентный подход решает эту проблему, однако за качество моделирования приходится платить как вычислительной сложностью, так и сложностью проектирования.

Казалось бы, разработка строго формализованного математического аппарата для

подобного рода модели практически невозможна. Однако данная статья показывает один из возможных способов применения теории множеств (и в частности теории графов) к строгому описанию модели. Более того, предложенное теоретико-множественное обоснование получилось достаточно гибким в том плане, что на его базе можно построить множество фактических реализаций в зависимости от требований к архитектуре ВС, языкам программирования, сетевым стандартам, способам представления знаний и т.д.

Не стоит забывать, что ввиду специфики решаемых задач изложенный подход к формализации не является сугубо единственным. Можно построить математических аппарат на базе UML, сетей Петри, модели акторов, теории конечных автоматов, трёхслойной архитектуры «структур-рутин-сообщений» и др. В любом случае, он должен опираться на такие фундаментальные основы, как теория графов, теория алгоритмов и вычислимых функций, формальные грамматики и т.д.

Литература

1. *Ермаков С.А., Замятина Е.Б.* Оптимизация распределённых алгоритмов имитационного моделирования // Материалы конференции ИММОД-2011 [URL http://simulation.su/files/immod2011/material/22.pdf].
2. *Pawlaszczyk D., Strassburger S.* Scalability in distributed simulations of agent-based models // URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.1282&rep=rep1&type=pdf
3. *Миков А.И.* Автоматизация синтеза микропроцессорных управляющих систем. – Иркутск: Изд-во Иркут. ун-та, 1987.
4. *Миков А.И., Замятина Е.Б,* Распределённые системы и алгоритмы: учеб. пособие // URL: http://www.intuit.ru/department/algorithms/distrsa/.
5. *Замятина Е.Б.* Современные теории имитационного моделирования: учеб. пособие // URL: http://window.edu.ru/resource/717/41717/files/imit_mod_lect.pdf .
6. *Damitio M., Turner S.J.* Comparing the Breathing time buckets algorithm and the time warp operating system on a transputer architecture // URL: http://tom-trix.ru/keep/time-buckets.pdf
7. *Chan W.K.V.* Agent-based simulation tutorial – simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation // URL: http://www.informs-sim.org/wsc10papers/014.pdf .
8. *Окольнишников В.В.* Разработка средств распределенного имитационного моделирования для многопроцессорных вычислительных систем // Москва: РГБ, 2007 (Фонд Российской Государственной Библиотеки).
9. *Myeong J., Gul A.* Agent framework services to reduce agent communication overhead in large-scale agent-based simulations // URL: http://osl.cs.uiuc.edu/publications/journals/simpra/JangA06.html

Научный руководитель: доцент кафедры математического обеспечения вычислительных систем ПГНИУ, к.ф.-м.н., доцент *Е.Б. Замятина*