ЛОГИЧЕСКИЙ ПОДХОД В МЕТОДОЛОГИИ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ АКТИВНЫХ СИСТЕМ

В. И. Гурьянов (Чебоксары)

Имитационные модели позволяют адекватно моделировать поведение сложных систем в том случае, когда они довольно точно воспроизводят объект исследования. Поэтому характерна ситуация, когда программная симуляция оказывается столь же сложным объектом, как и сам объект моделирования, что, в частности, вызывает затруднения в объяснении результатов имитационных экспериментов. Одно из решений – обратиться к формальным моделям имитирующих программ. В данной работе предлагается один из подобных подходов, основанный на темпоральной логике CTL*.

Имитационные модели активных систем. В работе [1] предложен проект профиля UML (*Scientific Profile*), предназначенного для объектного имитационного моделирования сложных систем. *Scientific Profile* представляет собой формальный язык, наделенный двойственной семантикой – предметной и вычислительной. Код имитирующей программы рассматривается как текст, описывающий предметную область. Прагматика задается стереотипом *Research Use Case Model*, семантика – *Research Analysis Model*, а синтактика – *Research Design Model*. Важным качеством профиля является то, что имитационные модели, сформулированные на *Scientific Profile*, не зависят от платформы, на которой проводятся вычислительные эксперименты. Далее рассматриваются имитационные модели активных (организационных) систем, заданные на этом языке.

На рис. 1. представлена диаграмма классов, определяющая объектные модели активных систем, и пример классов для одной из конкретных моделей активных систем. Абстрактный класс Subject фиксирует определяющие качества активных систем (активность, управление, целенаправленность) и имеет метод exist <<Exist>> (он определяет единицу дискретно-событийного времени), свойства product {Concept = Продукт}, materials {Concept = Материалы} и внутренние процедуры managment {Concept = Управление} и business {Concept = Бизнес-процесс}. С точки зрения вычислительной семантики данный класс задает пользовательский тип TSubject, который определяет общий интерфейс для всех элементов модели.

Теория активных систем рассматривает два базовых способа организации совместных действий активных элементов – механизм стимулирования и механизм планирования, которые в конкретных моделях обычно тесно переплетаются [2]. Экземпляр потомка класса Context моделирует окружающую среду изучаемой системы и определяет ее системную динамику посредством понятия *эффективность функционирования* активной системы. Экземпляр потомка класса ActiveSystem моделирует саму активную систему (которая обычно рассматривается в роли центра). Экземпляры потомка класса Agent моделируют агентов, составляющих активную систему и обладающих свойством активности, в том числе способностью свободы выбора своего состояния. Агенты могут обмениваться сообщениями через экземпляры агрегации между классами Agent и ActiveSystem. Агрегация также используется как канал связи между центром и агентами. Рефлексивная ассоциация для класса Agent отражает возможность существования прямых связей между агентами. Ментальность агентов будем моделировать исходя из гипотез рационального поведения и благожелательности. Для многих задач также необходимо выделение отдельного класса рефлектирующего агента.

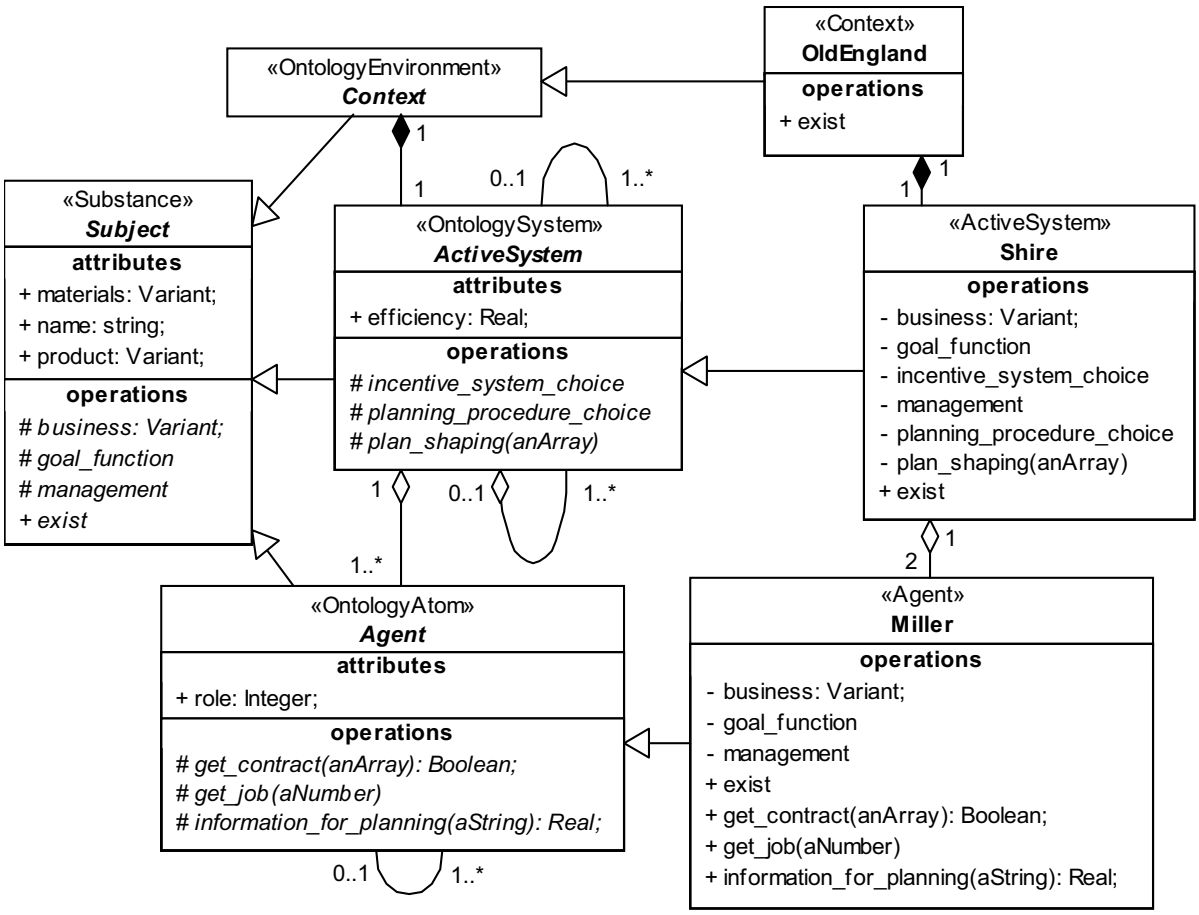

Рис. 1. Диаграмма классов для имитационной модели активной системы

На рис. 1. имена классов, методов и процедур совпадают с концептами, присвоенными им. Подчеркнем, что это исключительно мнемонический прием, поскольку в *Scientific Profile* существует регулярный механизм назначения предметной семантики программным сущностям. Для этой цели используется механизм значений с меткой Concept. Кроме того, мы предполагаем, что для модели использован субпрофиль (стереотипы субпрофиля находятся в отношении обобщения со стереотипами профиля). Для определения концептов субпрофиля использован глоссарий из книги [2].

В качестве примера объектной модели рассмотрим задачу «Лорд и два мельника». Центр организует бизнес-процесс при условии, что существует дефицитный ресурс (вода) и два агента (водяные мельницы). Затем бизнес-процесс выполняется и вычисляется эффективность управления. Имитационная модель учитывает стимулирующие, институциональные, мотивационные механизмы и несколько механизмов распределения ресурса (в примере их два). Все процессы являются параллельными и взаимодействуют между собой как путем обмена сообщениями, так и посредством общих переменных. Центр принимает решения на основе сообщений агентов, которые могут содержать недостоверную информацию (тем самым возникает ситуация игры). Подробнее об этих механизмах управления см. [2]. Целью *Исследователя* (стереотип *Researcher* "metaClass" Actor, {ProfileConcept = Researcher}) является поиск решения игры с учетом одновременного воздействия всех механизмов управления и некоторых других факторов.

Обратимся теперь к формальным моделям имитирующих программ, которые будем рассматривать как модели объяснения.

Формальные модели программных симуляций. В *computer science* известны многочисленные формальные модели программ, появление которых в значительной степени продиктовано проблемами верификации. Наиболее эффективным методом является *model checking* [3]. Представляется полезным применить данный метод в имитационном моделировании. На этапе разработки имитационной модели метод *model checking* сохраняет свою традиционную роль – как инструмент верификации. Однако на этапе эксплуатации его роль кардинально меняется, поскольку *model checking* можно рассматривать как метод исследования. Покажем, как это можно сделать для имитационных моделей активных систем.

Метод *model checking* применяется итерационно. Последовательность действий на каждой итерации может быть следующей.

1. Определяется (или модифицируется) множество атомарных высказываний *AP* и множество начальных состояний S_0. Исследуется множество состояний *S*, множество переходов *R* и определяется функция разметки *L* для программной симуляции посредством серии имитационных экспериментов. Генерируется структура Крипке $M = (S, S_0, R, L)$, после чего модель Крипке следует упростить. Цель этого упрощения – понижение размерности задачи.

2. В зависимости от задачи составляется (или уточняется) целостная система высказываний, адекватно отображающая теоретико-игровую ситуацию (позиционная, кооперативная, иерархическая, рефлексивная игра). Каждое высказывание записывается в виде формул CTL* и преобразуется затем к CTL или LTL.

3. Формулируется некоторая гипотеза о свойствах программной симуляции. Посредством стандартных процедур выполняются вычисления на структуре Крипке. Результат вычислений представляется в форме таблицы истинности.

Итерации выполняются до тех пор, пока не будет получено объяснение результатов имитационных экспериментов. Проведение подобных манипуляций с имитационной моделью невозможно или ограничено, так как это может поставить под сомнение валидность модели.

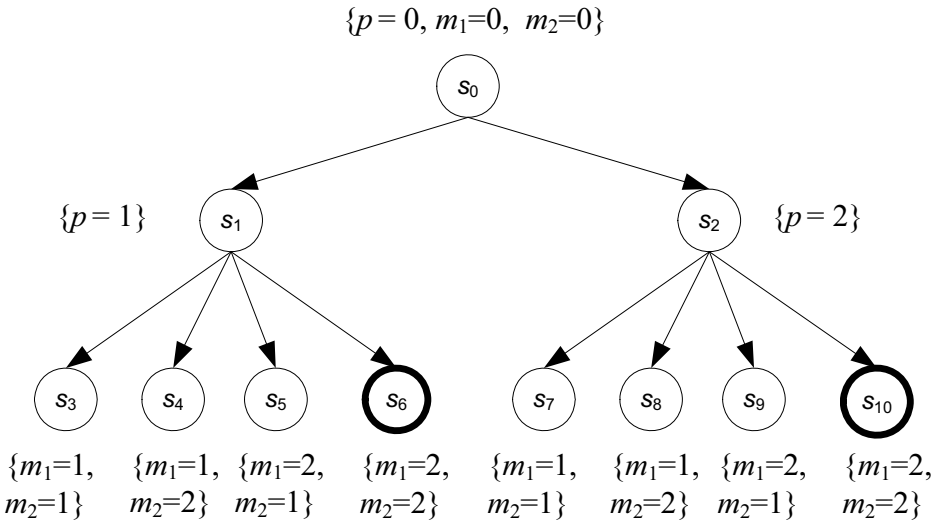

Рис. 2. Модель Крипке для программной симуляции

Вернемся к нашему примеру. На первой итерации можно выделить ведущий механизм управления – механизм распределения. Пусть множество переменных есть $V = \{p, m_1, m_2\}$, которые приобретают значения на домене интерпретации $D = \{0, 1, 2\} \times \{0, 1, 2\} \times \{0, 1, 2\}$, где 0 – игрок еще не сделал хода, 1 – выбрал первую альтернативу, 2 – выбрал вторую альтернативу. Определим множество атомарных высказываний $AP = \{p = 0, \ p = 1, \ p = 2, m_1 = 0, \ m_1 = 1, \ m_1 = 2, \ m_2 = 0, \ m_2 = 1, \ m_2 = 2\}$. Высказывания

будут следующие: $p = 1$ – центр предложил процедуру планирования, при которой агенты сообщают свои идеальные точки r_j , $j = 1,2$ функций предпочтения; $p = 2$ – центр предложил процедуру планирования, при которой агенты сообщают заявки; $m_j = 1$ – j-й агент сообщает недостоверную информацию; $m_j = 2$ – j-й агент сообщает достоверную информацию. Функция разметки L: $S \to 2^{AP}$ и отношение переходов R показаны на рис. 2. Множество начальных состояний $S_0 = \{(0, 0, 0)\}$.

Приведем теперь некоторые формулы CTL* для теоретико-игровых моделей. Формулы CTL* строятся из логических операций $\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$, темпоральных кванторов **A** («на всех путях»), **E** («на некоторых путях») и темпоральных операторов **X**(neXt), **F**(Future), **G**(Global), **U**(Until), **R**(Release). В нашем случае игра будет иерархической, и для этой ситуации можно выписать следующие формулы.

1. Начальное и терминальное состояние (окончание игры)

$f_0 = (p=0) \wedge (m_1=0) \wedge (m_2=0)$, $f_1 = \neg (p=0) \wedge \neg (m_1=0) \wedge \neg (m_2=0)$.

2. Предтерминальное состояние

$\neg f_1 \wedge \mathbf{EX} f_1$.

3. Равновесие Нэша. Составим формулу, которая позволяет определять состояния равновесия в теоретико-игровом смысле. Равновесие в смысле Нэша – это такое состояние, одностороннее отклонение от которого не выгодно ни одному из агентов. Пусть w^* – экстремум целевой функции центра, а v_j^* – соответствующее значение функции предпочтения j-го агента. Расширим множество атомарных высказываний AP, добавив в него высказывания $w<w^*$ и $w=w^*$, а также $v_j<v_j^*$, $v_j=v_j^*$, $j = 1,2$. На основе данных имитационных экспериментов выполним разметку для терминальных состояний (на рис.2 выделены состояния, в которых высказывание $w=w^*$ истинно).

Составим следующую формулу для начального состояния s_0:

$\varphi = \mathbf{EF}(\varphi_1 \wedge \mathbf{EX}\varphi_2) \wedge \mathbf{EG}$ isOptimumPath.

Потребуем, чтобы подформула $\varphi_1 \wedge \mathbf{EX}\varphi_2$ была истинной в некотором предтерминальном состоянии, а φ_2 – в некотором терминальном состоянии. Формулы для φ_1 и φ_2 запишем в следующем виде:

$\varphi_1 =$ isOptimumPath $\wedge \neg f_1 \wedge \mathbf{AX}\,(\delta \wedge f_1)$;

$\varphi_2 =$ isOptimumPath $\wedge f_1 \wedge (w=w^*) \wedge [(v_1=v_1^*) \wedge (v_2=v_2^*)]$,

где $\delta = [(w<w^*) \vee (w=w^*)] \wedge [(v_1<v_1^*) \vee (v_1=v_1^*) \wedge (v_2<v_2^*) \vee (v_2=v_2^*)]$; т.е. если рассматриваемый путь оптимальный, то на следующем шаге все состояния кроме равновесного менее (точнее – не более) выигрышны для каждого игрока.

Рассмотренные формулы допускают непосредственное обобщение на некоторые модели, представляющие практическую ценность. В модель анализа прецедентов (см. [1]), в пакет со стереотипом *Epistemology Entity* необходимо добавить библиотеку процедур для вычисления **EX**, **EG**, **EU** и процедуры заполнения таблицы выполнимости подформул. Сама таблица и средства визуализации определяются в пакете со стереотипом *Research Instruments*.

Приведенные выше формулы позволяют доказывать разного рода общие утверждения о свойствах программной симуляции. В рассматриваемом примере можно доказать (вычислением), например, следующее утверждение. Существует два решения игры, а именно – метаигрок выбирает либо первую, либо вторую процедуру планирования, и оба игрока в обоих случаях сообщают истинное значение плановой информации.

В нашем примере все вычисления можно выполнить непосредственно в таблице истинности. Все формулы CTL* – формулы состояния, поэтому таблицу истинности

можно представить в следующем виде. Из таблицы после завершения вычислений можно удалить атомарные высказывания, все «ординарные» состояния (точки орбит) и подформулы. Таблица истинности тогда даст компактное и стилизованное представление множества состояний. Для нашего примера два пути будут обращать φ в истину ($s_0s_1s_6$ и $s_0s_2s_{10}$). Заметим, что если из модели ментальности агентов исключить гипотезу благожелательности (заменив ее, допустим, на противоположную), и $r_1 < 1/2$, $r_1 + r_2 > 1$, то решением игры будет только единственный путь $s_0s_1s_6$.

Свернутая таблица истинности для пути $s_0s_1s_6$

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
f_0	*true*										
f_1				*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
$\neg f_1$ **EX** f_1		*true*	*true*								
isOptimumPath	*true*	*true*					*true*				
φ_1		*true*									
φ_2							*true*				
φ	*true*										

На последующих итерациях рассматривается влияние других механизмов управления на полученный результат. Такой анализ позволяет объяснить, почему одни механизмы распределения внезапно становятся неманипулируемыми, а другие, напротив, теряют это качество.

Итак, в статье предложен субпрофиль для объектного имитационного моделирования активных систем. Предложена методика исследования объектных моделей, за основу которой взят метод *model checking*. Приведены формулы CTL*, описывающие ситуацию иерархических игр. Реализация выполнена в среде Cincom Smalltalk (VisualWorks 7.6).

Выводы

В тех случаях, когда имитационные модели, как, например, модели организационных систем, демонстрируют сложное и неожиданное поведение, возникает настоятельная необходимость внесения в методологию имитационного моделирования средств объяснения. Необходимы методики, сочетающие традиционный имитационный эксперимент с анализом поведения симуляций на формальных моделях имитирующих программ. Подобные методики способствуют пониманию сущности явлений, дают гарантию достоверности результатов экспериментов и позволяют обобщать полученные выводы. В этом плане заслуживают внимания такие исчисления, как CCS (R.Milner), CSP (C.A.R. Hoare), π-исчисление (R.Milner) и др. В данной работе в этом качестве рассмотрен метод *model checking*.

Литература

1. **Гурьянов В. И.** Специальный UML-профиль для моделирования сложных систем // Информационные технологии моделирования и управления. Воронеж: Научная книга, 2010. № 3(62). С. 356–362. (см. также http://econf.rae.ru/article/5385)
2. **Новиков Д. А.** Теория управления организационными системами. М.: МПСИ, 2005. 584 с.
3. **Кларк Э. М., Грамберг О., Пелед Д.** Верификация моделей программ: Model checking М: МЦНМО, 2002. 416 с.