КИБЕРНЕТИКА

А.А. ЛЕТИЧЕВСКИЙ, А.Ал. ЛЕТИЧЕВСКИЙ, В.С. ПЕСЧАНЕНКО, А.А. ГУБА

УДК 519.686.2

ГЕНЕРАЦИЯ СИМВОЛЬНЫХ ТРАСС В СИСТЕМЕ ИНСЕРЦИОННОГО МОДЕЛИРОВАНИЯ

Аннотация. Описан новый генератор символьных трасс, разработанный для последней версии системы инсерционного моделирования. Основными характеристиками генератора являются использование графического представления описания многоуровневых моделей, разделение локальных описаний и отношения следования, возможность настройки на различные стратегии поиска, применение нового предикатного трансформера, допускающего кванторы общности с ослабленными ограничениями относительно предыдущих версий.

Ключевые слова: верификация, инсерционное моделирование, язык UCM.

ВВЕДЕНИЕ

Инсерционное моделирование представляет собой направление, которое развивается в течение последних двух десятилетий в качестве подхода к общей теории взаимодействия агентов и сред в сложных распределенных многоагентных системах.

История инсерционного моделирования начинается с работ А. Летичевского и Д. Гильберта, выполненных во второй половине 90-х годов прошлого столетия, в которых исследованы абстрактные семантические модели недетерминированных параллельных языков программирования [1, 2]. В дальнейшем этот подход был обобщен до общей модели взаимодействия агентов и сред [3], основанной на понятии функции погружения и алгебры поведений, являющейся разновидностью алгебры процессов. Модели инсерционного моделирования обобщают большинство традиционных моделей взаимодействующих процессов, включая CCS (исчисление взаимодействующих процессов) и π-исчисление Р. Милнера [4–6], CSP (взаимодействующие последовательные процессы) Т. Хоара [7], ACP (алгебра взаимодействующих процессов) Бергстры [8], исчисление мобильных амбиентов Л. Карделли [9] и многие другие ответвления этих основных теорий.

Каждую теорию взаимодействующих процессов можно получить в результате выбора подходящей функции погружения — параметра инсерционной модели. Отметим, что в одной модели можно использовать несколько функций погружения (многоуровневые среды). Это позволяет объединять различные теории взаимодействия. Инсерционное моделирование применяется также для унифицированного представления таких абстрактных моделей параллельных вычислений, как сети Петри [10], модели актеров Хьюита [11], автоматные сетевые модели и различные абстракции парадигмы объектно-ориентированного параллельного программирования.

Современное представление теоретических основ инсерционного моделирования описано в [12], где агенты рассмотрены как непрерывные преобразования среды, а их поведения — как элементы полной непрерывной алгебры поведений, расширенной непрерывными операциями, среди которых главное место занима-

ет функция погружения. Термин «инсерционное моделирование» введен в [13] вместо «моделирование взаимодействия агентов и сред».

В 2000-х годах начались работы по применению инсерционного моделирования при разработке систем верификации требований и спецификаций распределенных взаимодействующих систем [14–18]. Система VRS (Verification of Requirement Specifications), разработанная по заказу фирмы Моторола с участием сотрудников Института кибернетики им В.М. Глушкова, успешно использовалась для проверки требований и спецификаций в области систем реального времени, а также телекоммуникационных и встроенных систем.

Прототипы для основных компонентов системы VRS созданы на базе первых версий системы инсерционного моделирования, которая вначале разрабатывалась как интерпретатор для языка действий [2]. Новая версия системы инсерционного моделирования IMS [19] строится как среда для разработки инсерционных машин.

ОПИСАНИЕ МОДЕЛЕЙ В СИСТЕМЕ IMS

Основой построения инсерционных моделей являются локальные описания $\forall x\,(\alpha(x) \to < P(x) > \beta(x))$ [13, 20]. Они представляют локальные свойства системы: если для некоторого состояния $s = \sigma[u_1, u_2, \dots]$ системы с погруженными в нее агентами $u_1, u_2, \dots$ и некоторого набора значений параметров x предусловие $\alpha(x)$ на состоянии s истинно, то начинается процесс $P(x)$ и после его успешного завершения на новом состоянии будет истинным постусловие $\beta(x)$. Все состояния рассматриваются с точностью до бисимуляционной эквивалентности и могут отождествляться с поведениями системы в этих состояниях. Локальные описания имеют не только логическую, но и императивную семантику. Каждое из них определяет некоторый недетерминированный оператор на множестве состояний системы, который можно выполнять, применяя его к текущему состоянию. С помощью этого оператора определяется отношение переходов $s \xrightarrow{a} s'$. Действие a, определяющее переход, может быть локальным описанием (крупношаговая семантика) или процессом $P(x)$ с подставленными значениями параметров. В этом случае можно перейти от одного перехода к одной или нескольким последовательностям переходов, рассматривая мелкошаговую семантику [21]. В дальнейшем для локальных описаний системы будет рассмотрена крупношаговая семантика. Если s — состояние системы, а Q — множество локальных описаний, то система уравнений

$$s = \sum_{s \xrightarrow{B} s'} B.s' + \varepsilon_s, \;\; B \in Q,$$

представляет крупношаговую семантику системы локальных описаний.

Для полного задания модели одних локальных описаний недостаточно. Отметим, что в локальных описаниях не существует ограничений на последовательности их применения, что приводит к рассмотрению нежелательных историй и трасс. Следующий уровень описания модели состоит в определении отношения следования на множестве локальных описаний. Это отношение можно ввести путем определения дополнительных управляющих атрибутов и условий на них, ограничивающих условия применения локальных описаний. Недостатком такого описания является необходимость разделения основных и вспомогательных атрибутов управления. Кроме того, усложняются собственно сами локальные описания. В системе VRS используются специальные структуры управления поиском (guided search), однако и они специфичны и не позволяют полностью решить проблемы. В последнее время в системе VRS для задания отношения следования использует-

ся язык UCM, разработанный организацией ITU, как один из языков для формализации требований к программным системам [22]. Формальная семантика языка UCM построена в работе [23]. Семантика плоских карт и стабов представлена с помощью инсерционных моделей в работах [24, 25].

В системе IMS верхним уровнем описания модели является управляющая система, которая определяет порядок применения локальных описаний базовой системы. Управляющую систему можно задать системой уравнений в расширенной алгебре поведений, в которой помимо основных операций и функций используются функции погружения для различных сред [20]. Особенность управляющей системы заключается в том, что во множество ее действий входят локальные описания для базовой системы. Состояние управляющей системы представляется выражением $U[S]$, где U — состояние управляющей системы, а S — состояние базовой системы, определенной множеством локальных описаний Q. Отношение переходов для системы с управлением зададим правилами

$$\frac{U \xrightarrow{a} U'}{U[S] \xrightarrow{a} U'[S]}\ a \notin Q,$$

$$\frac{U \xrightarrow{a} U',\ S \xrightarrow{a} S'}{U[S] \xrightarrow{a} U'[S']}\ a \in Q.$$

Допустим, что переходы с действиями, отличными от локальных описаний базовой системы, упрятаны, т.е. первое правило заменено правилом

$$\frac{U \xrightarrow{a} U'}{U[S] \longrightarrow U'[S]}\ a \notin Q.$$

Тогда внешний наблюдатель не заметит разницы между функционированием базовой системы с управлением и без него. Он увидит только трассы той же самой базовой системы. При этом с использованием управляющей системы их может быть меньше, а также возможны тупиковые состояния, которых не существует для базовой системы. Управляющая система как среда имеет доступ к состоянию базовой системы и анализирует ее состояния так же, как инсерционная машина реального времени. Такие машины можно использовать как управляющие системы. Однако для генерации трасс нужен еще один уровень управления — аналитическая инсерционная машина, описанная далее.

УПРАВЛЕНИЕ МОДЕЛЯМИ В СИСТЕМЕ IMS

Для разработки управляющих компонентов инсерционных моделей в системе IMS используется язык LiveUCM-IMS (в дальнейшем LiveUCM). Он имеет два представления: алгебраическое и графическое. В основе семантики LiveUCM лежат основные семантические конструкции языка UCM, выраженные в терминах инсерционного моделирования. Язык UCM на абстрактном уровне можно рассматривать как удачную интеграцию двух концепций: параллелизм в сетях Петри и идея мобильных амбиентов [9]. Язык LiveUCM дополняет эти две концепции идеей многоуровневой среды.

Описание управляющей системы в LiveUCM, как и в UCM, состоит из множества вложенных карт. На каждой карте изображена сеть направленных путей. Пример такой системы, состоящей из трех карт: одной карты верхнего уровня и двух вложенных в нее карт, представлен на рис. 1. Каждый путь имеет начальную вершину (рис. 2, *а*), причем их может быть несколько. Конечные максимальные пути заканчиваются конечными вершинами, представленными жирными коротки-

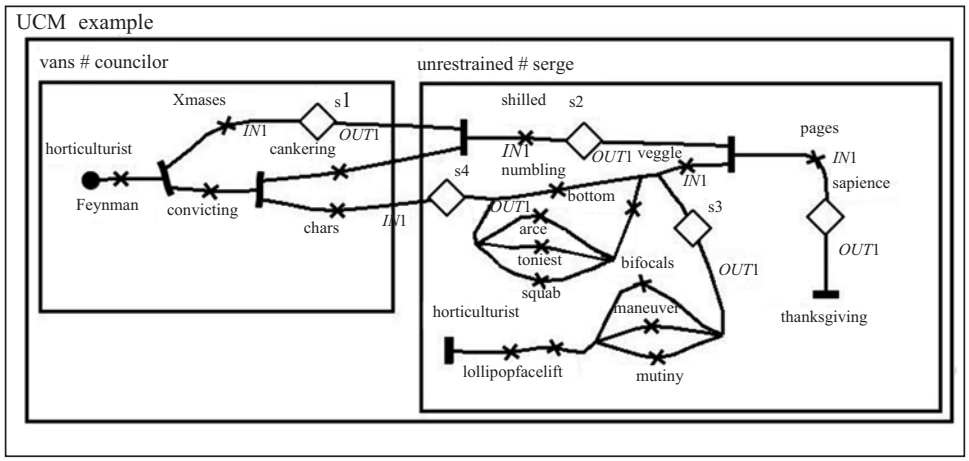

Рис. 1

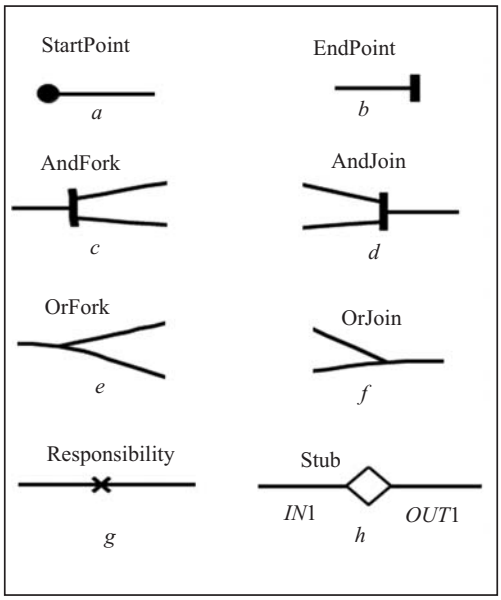

Рис. 2

ми линиями, ортогональными путям, которые они завершают (рис. 2, *b*). Из начальной вершины выходит, а в конечную входит только один путь.

Пути могут разветвляться и сливаться (fork and join). Существует два вида ветвления и слияния: параллельные и альтернативные. Параллельные ветвления (AndFork, рис. 2, *c*) и слияния (AndJoin, рис. 2, *d*) обозначены жирными линиями, ортогональными путям. У параллельного ветвления существует один входной путь и несколько выходных. Параллельное слияние может иметь несколько входных путей, но только один выходной. У альтернативных ветвления (OrFork, рис. 2, *e*) и слияния (OrJoin, рис. 2, *f*) такая же структура, как и у параллельных (ветвление имеет один входной путь и несколько выходных, слияние — несколько входных и один выходной путь). Альтернативные ветвления и слияния представлены точками на путях.

Кроме ветвлений и слияний на путях имеются два вида символов: обязательство, обозначенное крестиком (Responsibility, рис. 2, *g*), и стаб, обозначенный ромбом (Stub, рис. 2, *h*). Символы обязательств имеют по одному входному и выходному пути, символ стаба может иметь по нескольку входных и выходных путей. С каждым обязательством ассоциировано некоторое локальное описание системы локальных описаний, которые, в основном, относятся к базовой системе, хотя в некоторых случаях — и к управлению.

Пути могут переходить из одной карты в другую, расположенную на том же или смежном уровне (верхнем или нижнем) (рис. 3). Таким образом, пути соединяют вершины, находящиеся в различных картах.

С каждым стабом ассоциирована некоторая карта, на которую ссылается данный стаб, входные пути последнего связаны с начальными, а выходные пути — с конечными вершинами ассоциированной карты.

Интуитивную семантику языка LiveUCM можно представить следующим образом. В начальный момент времени активными являются все начальные вершины каждой карты, кроме ассоциированной со стабами. Далее активные точки движутся по своим путям. Состояние системы изменяется только, когда активные точки на путях проходят через точки переходов. К ним относятся: обязательства, ветвления, слияния, точки a — выходы из карты нижнего уровня на верхний уровень, точки b — входы в карты нижнего уровня (см. рис. 3), стабы, начальные и конечные вершины.

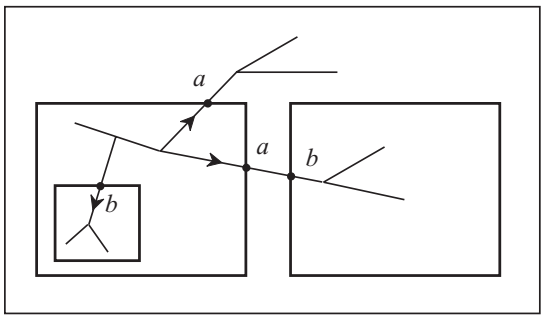

Рис. 3

Все точки переходов и карты имеют имена, уникальные внутри одной карты. Для того чтобы различать имена в различных картах, к ним достаточно добавить имена карт, лежащих на пути к именуемым элементам от карты верхнего уровня.

Изменение состояний системы происходит по правилу интерливинга: в каждый момент времени переход может произойти в результате срабатывания только одной точки перехода.

При прохождении активной точки через обязательство система делает попытку выполнить локальное описание, ассоциированное с данным обязательством. Если это удается, то локальное описание выполняется и изменяется состояние базовой системы. Единственный путь, выходящий из точки обязательства, активизируется, т.е. на нем появляется активная точка, прошедшая через данное обязательство. В противном случае активная точка останавливается перед обязательством и переходит в состояние ожидания, переходы системы выполняются в результате прохождения других активных точек через точки переходов. Когда все активные точки каждой карты находятся в состоянии ожидания, система попадает в тупиковое состояние (dead lock).

При прохождении параллельного ветвления активизируются все выходные пути. Параллельное слияние синхронизирует движение активных точек. На каждом пути со временем накапливается некоторое количество контрольных точек, которые находятся в состоянии ожидания. Когда на каждом входном пути параллельного ветвления имеется по крайней мере одна активная точка, которая находится в состоянии ожидания, активизируется выходной путь и на всех входных путях слияния количество активных точек уменьшается на единицу.

Прохождение альтернативного ветвления выполняется по следующему алгоритму. Определяются все точки переходов, отличные от альтернативных ветвлений, которые можно достичь, проходя только через альтернативные ветвления и слияния. Выбирается недетерминированным образом одна из полученных точек переходов. При этом запрещается выбирать обязательство, локальное описание которого неприменимо к текущему состоянию базовой системы. Если такую точку выбрать нельзя, то активная точка переходит в состояние ожидания. Путь, по которому активная точка проходит к выбранной точке перехода, также выбирается недетерминированным образом. Неоднозначность выбора этого пути определяется возможными точками альтернативного слияния. При движении активной точки от альтернативного ветвления до выбранной точки перехода все другие активные точки приостанавливаются. Когда активная точка достигает выбранной точки перехода, последний выполняется, т.е. происходит соответствующее изменение состояния системы. Альтернативное слияние может быть пройдено всегда.

Выход из карты нижнего уровня на верхний и вход в карту нижнего уровня возможны всегда.

Когда активная точка достигает стаба, последний заменяется копией ассоциированной с ним карты. При этом начальные вершины ассоциированной карты становятся точками входа, а все конечные вершины — точками выхода. Все входные пути связываются с соответствующими точками входа ассоциированной карты, а все выходные — с соответствующими точками выхода. После процесс продолжается в соответствии с описанными выше правилами.

В начальный момент времени активными являются отрезки путей, выходящих из начальных вершин всех карт, которые не ассоциированы со стабами.

Активные точки не различаются и их местоположение на отрезке пути между двумя точками переходов несущественно. Поэтому состояние системы управления однозначно определяется количеством активных точек на каждом отрезке, соединяющем две ближайшие точки переходов. А состояние всей системы определяется состояниями систем управления и базовой.

ФОРМАЛЬНАЯ СЕМАНТИКА LiveUCM

Структура среды. Каждая карта представляет собой атрибутную среду для погруженных в нее агентов. Сигнатура карты включает конечное множество T символьных констант для обозначения имен точек перехода и отношение следования $Succ \subseteq T^2$, которое описывает структуру путей карты: $(a, b) \in Succ$ тогда и только тогда, когда существует путь из точки перехода a в точку перехода b, на котором не имеется других точек перехода. Стабы отличаются от других точек перехода тем, что их входные и выходные пути упорядочены. Поэтому каждому стабу b сопоставим еще несколько точек перехода: точки входа в стаб $(1:b)$, $(2:b), \ldots$ и точки выхода из стаба $(b:1), (b:2), \ldots$ Отношение следования есть статическая структура данных, которая меняется только при входе в стаб. Множество точек перехода карты включает и все точки перехода каждой вложенной в нее карты.

Для обозначения типов точек переходов используется другая серия символьных констант: *Resp* (обязательство), *OrFork* (альтернативное ветвление), *AndFork* (параллельное ветвление), *OrJoin* (альтернативное слияние), *AndJoin* (параллельное слияние), *MapOut* (выход из карты нижнего уровня на верхний или выход из стаба), *MapIn* (вход в карту нижнего уровня или вход в стаб), *Stub* (стаб), *StubIn* (точка входа в стаб), *StubOut* (точка выхода из стаба), *Init* (начальная вершина), *Termnl* (конечная вершина). Они являются значениями перечислимого типа *PoinType.* Атрибут $PT : T \to PointType$ позволяет определить тип точки перехода. Вместо формулы $PT(b) = t$ используется более короткое выражение $t(b)$.

Если $(a, b) \in Succ$, то точку a будем называть точкой входа для b, а точку b — точкой выхода для a. Отношение следования должно удовлетворять следующим требованиям:

— каждое обязательство (точки типа *MapOut* и *MapIn*) имеет точно по одной точке входа и выхода;

— каждое ветвление имеет точно одну точку входа и по крайней мере две точки выхода;

— каждое слияние имеет по крайней мере две точки входа и точно одну точку выхода;

— начальная вершина не имеет точек входа и имеет точно одну точку выхода;

— конечная вершина не имеет точек выхода и имеет точно одну точку входа;

— точкой выхода для точки входа в стаб есть стаб, а точка выхода из стаба имеет стаб в качестве входа;

— пусть $StubIn(b)$ — множество точек входа в стаб b, $StubOut(b)$ — множество точек выхода из стаба b. Тогда

$$StubIn(b) = \{(i:b) \mid i = 1, 2, \ldots, m\},\ \ StubOut(b) = \{(b:i) \mid i = 1, 2, \ldots, n\},$$

где m и n — соответственно число начальных и конечных вершин карты, ассоциированной со стабом b.

Состояние карты как среды описывается функциональным атрибутом $active : Succ \to N$, где N — множество целых неотрицательных чисел, и состояниями всех вложенных в нее карт. Равенство $active(a, b) = n$ означает, что на отрезке между a и b находится в точности n активных точек.

Агенты. В качестве агентов, погруженных в среду карты, рассматриваются активные точки, движущиеся по ее путям, а также карты следующего уровня, вложенные (погруженные) в данную карту. В текущий момент времени активная точка находится на некотором отрезке пути между двумя точками переходов: точкой входа и выхода. Поведение агента, находящегося на отрезке (a, b), отождествляется с его состоянием и обозначается $behavior(a, b)$, а действие, которое выполняется при прохождении точки b из отрезка (a, b), обозначается $action(a, b)$. Если тип вершины b есть обязательство, то $action(a, b)$ есть локальное описание, соответствующее данному обязательству, во всех остальных случаях это может быть любым выражением, которое содержит полную информацию о границах интервала a и b. Поведения агентов связаны соотношениями алгебры поведений, которые однозначно их определяют как наименьшее решение соответствующей системы уравнений. Вот эти соотношения:

$$Resp(b) \wedge (b, c) \in Succ \to behavior(a, b) = action(a, b).behavior(b, c),$$

$$OrFork \to behavior(a, b) = action(a, b). \sum_{(b, c) \in Succ} behavior(b, c),$$

$$AndFork \to behavior(a, b) = action(a, b). \coprod_{(b, c) \in Succ} behavior(b, c)$$

(символ $\coprod_{x \in X} u(x)$ обозначает параллельную композицию поведений $u(x)$ по всем элементам $x \in X$);

$$OrJoin(b) \wedge (b, c) \in Succ \to behavior(a, b) = action(a, b).behavior(b, c),$$

$$AndJoin(b) \wedge (b, c) \in Succ \to behavior(a, b) = action(a, b).behavior(b, c)$$

(все активные точки на входе параллельного слияния стремятся перейти в точку c, однако, как будет видно дальше, среда разрешает только одной точке (не важно какой) перейти через параллельное слияние, остальные (по одной для каждого входного пути) прекращают свое движение);

$$MapOut(b) \wedge (b, c) \in Succ \to behavior(a, b) = action(a, b).behavior(b, c),$$

$$MapIn(b) \wedge (b, c) \in Succ \to behavior(a, b) = action(a, b).behavior(b, c)$$

(поведение агента при переходе из одной карты в другую ничем не отличается от переходов агента через обязательства или слияния);

$$Termnl(b) \to behavior(a, b) = action(a, b).\Delta.$$

Заметим, что приведенные соотношения определяют правила переходов агентов. Например, первое соотношение можно записать в виде

$$Resp(b) \wedge (b, c) \in Succ \to behavior(a, b) \xrightarrow{action(a, b)} behavior(b, c).$$

Уравнений для прохождения стаба не существует. Последнее объясняется тем, что подстановку делает среда. В результате этой подстановки меняется структура карты и тип второй вершины интервала, на котором находится активная точка, также может измениться. Поэтому, дойдя до стаба, активная точка ожидает изменения структуры карты на одном из его входов.

Функция погружения. Перейдем теперь к рассмотрению общих правил вычисления функции погружения для карт, рассматриваемых как среды. Текущее состояние карты представляется выражением $M[u_1, u_2, \dots]$, где M — неразложимое состояние карты, т.е. набор значений ее атрибутов, $u_1, u_2, \dots$ — состояния активных точек и вложенных карт, погруженных в карту M. Функция погружения карты коммутативна. Поэтому, если необходимо применить правило перехода для одного агента, его нужно переместить на последнее место и применить одно из правил перехода к среде, представленной в виде $(M[u_1, u_2, \dots])[u] = M'[u]$.

Функция погружения для карты является параллельным погружением. Это значит, что $M[u, v] = M[u \,||\, v]$. Другое свойство функции погружения — аддитивность: $M[u + v] = M[u] + M[v]$.

Используя эти свойства, а также известные свойства параллельной композиции, получаем, что состояние карты можно привести к виду $\sum_{i \in I} M[a_i.u_i] + \varepsilon$, где ε есть $M[\Delta]$ или 0. Состояние $M[\Delta]$ соответствует случаю, когда для всех пар $(a, b) \in Succ$, $active(a, b) = 0$, т.е. карта M не имеет вложенных карт и активных точек.

Предположим, что множество T имен активных точек и карт меняется соответствующим образом при каждом погружении новой карты. Иными словами, если $M \xrightarrow{a} M'[N]$, где N — новая карта, то перед выполнением перехода $M \xrightarrow{a} M'$ к множеству T имен точек перехода карты M добавляются имена точек перехода карты N. Аналогично меняется и отношение следования карты M: к нему добавляется отношение следования карты N.

Функция погружения для действия x, соответствующего прохождению активной точки через обязательство, ветвление, слияние или конечную вершину M, определяется правилом

$$\frac{M \xrightarrow{x} M',\ u \xrightarrow{x} u'}{M[u] \xrightarrow{x} M'[u']}.$$

Переход $M \xrightarrow{x} M'$ карты M задается с помощью локальных описаний, зависящих от типа вершины, которую проходит активная точка. Вот эти описания:

$\forall (a, b)($
$\quad active(a, b) > 0 \wedge (Resp(b) \vee AndFork(b) \vee OrJoin(b) \vee Termnl(b))$
$\quad \to < action(a, b) >$
$\quad active(a, b) := active(a, b) - 1 \wedge$
$\quad \forall c((b, c) \in Succ \to active(b, c) := active(b, c) + 1)$
$);$
$\forall (a, b, c)($
$\quad active(a, b) > 0 \wedge OrFork(b) \wedge$
$\quad (behavior(a, b) = action(a, b).(active(b, c).u + v))$

$$\rightarrow < action(a, b) >$$
$$active(a, b) := active(a, b) - 1 \wedge active(b, c) := active(b, c) + 1$$
);
$$\forall (a, b, c)($$
$$active(a, b) > 0 \wedge AndFork(b) \wedge ((b, c) \in Succ) \wedge$$
$$\forall a'((a', b) \in Succ \rightarrow active(a', b) > 0)$$
$$\rightarrow < action(a, b) >$$
$$active(a, b) := active(a, b) - 1 \wedge$$
$$\forall a'((a', b) \in Succ \rightarrow active(a', b) := active(a', b) - 1)$$
);

Для переходов из одной карты в другую применяются иные правила:

$$\frac{M \xrightarrow{x} M',\ N \xrightarrow{x} N',\ u \xrightarrow{x} u'}{M[N[u]] \xrightarrow{x} M'[N', u']}\ (x = action(a, b) \wedge MapOut(b)),$$

$$\frac{M \xrightarrow{x} M',\ N \xrightarrow{x} N',\ u \xrightarrow{x} u'}{M[N[u]] \xrightarrow{x} M'[N'[u']]}\ (x = action(a, b) \wedge MapIn(b) \wedge b \in N).$$

Локальное описание для переходов карты имеет вид

$$\forall (a, b, c)($$
$$active(a, b) > 0 \wedge MapIn(b) \wedge (b, c) \in Succ \wedge PointType(c) \neq Stub$$
$$\rightarrow < action(a, b) >$$
$$active(a, b) := active(a, b) - 1 \wedge active(b, c) := active(b, `c) + 1$$
);
$$\forall (a, b, c)($$
$$active(a, b) > 0 \wedge MapOut(b) \wedge (b, c) \in Succ$$
$$\rightarrow < action(a, b) >$$
$$active(a, b) := active(a, b) - 1 \wedge active(b, c) := active(b, c) + 1$$
);

Появление активной точки перед входом в стаб может стать причиной выполнения перехода среды, при котором стаб заменяется ассоциированной с ним картой. Правило перехода имеет вид

$$(u = action(a, b).u') \wedge (b, c) \in Succ \wedge Stub(c) \rightarrow (M[u] \rightarrow M'[u, N(c)]).$$

Карта $N(c)$ получается из карты, ассоциированной со стабом c, подстановкой вершин из множества $StubIn(c)$ вместо ее начальных вершин и вершин из множества $StubOut(c)$ вместо ее конечных вершин с учетом порядка на множествах начальных и конечных вершин.

Если в системе управления не допускаются рекурсивных вызовов стабов, то все подстановки можно сделать статически и работать в системе без стабов. В этом случае на уровне реализации можно использовать дополнительные возможности оптимизации, связанные со спецификой предметной области. В настоящей статье не рассмотрено окончательной формализации подстановок карт, чтобы не описывать деталей реализации, например, технологий обеспечения уникальности имен и способов доступа к ним. Эта формализация требует более низкого уровня алгоритмизации, а спецификации не должны требовать ограниченных решений.

Базовая система. Генератор трасс рассчитан в первую очередь на символьное моделирование. Это значит, что состояние базовой системы представляется формулой логики первого порядка или темпоральной логики и продвижение по путям системы управления требует применения дедуктивных средств для рас-

познавания применимости локальных описаний. Базовая система может иметь свою многоуровневую иерархическую структуру, не зависящую от иерархии карт. В некоторых случаях эти две иерархии полезно строить совместно. Это значит, что среди атрибутов карт позволено располагать некоторые атрибуты базовой системы с формулами, описывающими свойства их значений, а переход из одной карты в другую может сопровождаться изменением сигнатуры и состояния базовой системы. Атрибуты базовой системы могут быть доступны системе управления и, наоборот, атрибуты системы управления могут использоваться базовой системой. Такое взаимодействие обеспечивает более полное единство управления и обработки данных и его можно применять как на уровне моделирования, так и на уровне реализации. Использование этих возможностей удобно реализовать на следующем уровне управления в виде среды, которая содержит управление и базовую систему в качестве своих агентов. Для моделирования этот уровень обеспечивается генератором трасс.

ГЕНЕРАТОР ТРАСС

Символьный генератор трасс системы IMS — аналитическая инсерционная машина систем локальных описаний с управлением, представленных в виде LiveUCM-IMS. Эта инсерционная машина используется для решения различных задач, требующих генерации трасс. Поэтому она представлена как настраиваемая машина с возможностью адаптации к различным классам задач. Далее приведено неформальное описание генератора трасс. Формальное описание можно сделать на основании формальной семантики языка LiveUCM-IMS, изложенной ранее.

Обычно генератор трасс рассматривается как среда для анализируемой модели и полное состояние генератора трасс описывается с помощью выражения $G[U[S]]$, где G, U и S — соответственно состояния генератора, управляющей и базовой системы. Генератор трасс обходит дерево поведения системы $U[S]$, порождая ее символьные состояния в соответствии с некоторой стратегией генерации, которая задается с помощью некоторых параметров, определяющих настройку генератора на решаемую задачу.

Основные классы задач, решаемых с помощью генератора трасс, — это символьная верификация модели и построение тестовых наборов для проверки модели на конкретных состояниях. В первом случае в процессе генерации проверяется выполнимость заданных условий, во втором — строится система трасс, удовлетворяющая некоторому критерию покрытия.

В процессе генерации трасс накапливается множество пройденных состояний. При вторичном попадании в уже пройденное состояние генерация для этого состояния прекращается. В случае символьного моделирования состояние есть формула, которая прокрывает множество конкретных состояний, и для прекращения генерации можно использовать более слабые условия, например, из нового состояния логически следует уже пройденное.

Первым параметром генератора трасс является множество контрольных точек на картах системы управления, в качестве которых выбираются некоторые точки переходов. Начальные и конечные вершины, входы в стабы и параллельные ветвления выбираются обязательно, а стабы никогда не являются контрольными точками. Выбор контрольных точек может определяться некоторым критерием или конкретным указанием точек перехода, например все обязательства (для задач верификации), все точки переходов или все обязательства и ветвления (для задачи тестирования).

Две контрольные точки называются смежными, если на соединяющих их путях не существует других контрольных точек. Предположим, что контрольные

точки разрезают все циклы отношения следования. Так, если $(a_1, a_2, \dots, a_m)$ — последовательность точек переходов такая, что $(a_i, a_{i+1}) \in Succ,\ i = 1, 2, \dots, m-1$ и $a_1 = a_m$, то существует i такое, что $1 < i < m$ и a_i есть контрольная точка. Из этого предположения следует, что количество путей, соединяющих две контрольные точки, конечно.

Рассмотрим трассу системы управления, началом которой есть параллельная композиция всех начальных точек, и пару смежных контрольных точек, находящихся на этой трассе. Тогда множество трасс, соединяющих эти две контрольные точки и не проходящих через другие контрольные точки, конечно. Действительно, причиной бесконечного количества трасс, соединяющих два состояния, могут быть только параллельные ветвления и переходы внутрь стаба. Но они включены в множество контрольных точек и не находятся внутри трасс, соединяющих две смежные контрольные точки.

Исходя из этих соображений, для любой пары достижимых из начального состояния смежных контрольных точек можно построить процесс, представляющий собой недетерминированную сумму последовательной композиции обязательств по всем путям, соединяющим заданные контрольные точки и не проходящим через другие контрольные точки. Назовем этот процесс укрупненным локальным описанием.

Трасса, которая получается из обычной трассы проекцией на контрольные точки, называется укрупненной. После выбора контрольных точек генератор трасс строит укрупненные трассы и применяет укрупненные локальные описания к символьным состояниям базовой системы. Переход от одной контрольной точки к другой можно рассматривать как результат выполнения укрупненного действия, которое будем обозначать как $Act(a, a')$, где a и a' — смежные контрольные точки. Если рассматривать только укрупненные действия, получим укрупненную управляющую систему, а перенос укрупненных действий на базовую систему приведет к укрупненной базовой системе.

Основной атрибут генератора трасс — дерево, составленное из укрупненных трасс. Листьями этого дерева являются состояния управляющей системы с соответствующими состояниями базовой системы, полученные в результате прохождения укрупненных трасс. Каждое состояние отмечено трассой, которая привела к нему. Таким образом, состояние генератора в текущий момент времени можно записать в виде

$$G[p_1 : U_1[S_1],\ p_2 : U_2[S_2], \dots,\ p_n : U_n[S_n]].$$

Выражения $p_i : U_i[S_i]$ рассматриваются как состояния агентов, погруженных в среду генератора трасс. Эти агенты не взаимодействуют, их состояния относятся к различным моментам времени и они имеют альтернативные возможности движения базовой системы с управлением. Отметки состояний $p_1, p_2, \dots, p_n$ могут являться ссылками на последние вершины трассы в дереве трасс, по которым трассы восстанавливаются однозначно. Состояние U_i обычно представляет собой параллельную композицию активных точек, после развертывания которой оно превращается в охраняемую недетерминированную сумму параллельных композиций. Их взаимодействие осуществляется путем синхронизации точек параллельного слияния и ожидания применимости локальных описаний при прохождении обязательств.

Основное правило, описывающие функционирование генератора трасс, имеет вид

$$\frac{G \xrightarrow{Act(a,a')} G',\ U[S] \xrightarrow{Act(a,a')} U'[S'],\ U = Act(a,a').U' + U''}{G[p * a : U[S]] \xrightarrow{Act(a,a')} G'[p * a * a' : U'[S'] + p * a : U''[S]]}\ Allowed.$$

Здесь * обозначает конкатенацию трасс в дереве трасс, переход на состояниях генератора меняет соответствующим образом дерево трасс. Главные свойства генератора трасс спрятаны в функции *Allowed*. Она выбирает трассу, которую полезно продолжать, и слагаемое недетерминированной суммы, находящееся в конце этой трассы, которое также будет продолжено. Эти два выбора можно делать одновременно. Алгоритм выбора представляет собой второй главный параметр настройки генератора на класс решаемых задач.

Одним из классов задач, для которых разработана настройка генератора трасс, есть построение системы тестов, удовлетворяющей следующему критерию покрытия: любой переход между контрольными точками должен иметь трассу, покрывающую этот переход, и все трассы должны заканчиваться в заданных терминальных точках. Тогда критерий выбора состоит в том, чтобы покрыть как можно большее число переходов и продвинуться как можно ближе к терминальным точкам. В случае, когда эти два критерия вступают в противоречие, первому отдается предпочтение, если покрыто мало переходов, второму — если число покрытых переходов достаточно велико. Точное значение этих критериев выражается максимизацией взвешенной суммы соответствующих числовых параметров. Веса подбираются экспериментально или с подключением алгоритмов самообучения при достаточно большом наборе примеров.

Дерево трасс имеет значительные преимущества по сравнению с простым множеством. Прежде всего для некоторых задач, в частности для задачи тестирования, представление результата в виде дерева может использоваться в дальнейшем для проведения непосредственно тестирования. Однако отметим, что это удобная структура данных для хранения различного рода вспомогательных данных для процесса генерации. Так, в точках трасс могут храниться последовательности состояний базовой системы для определения пройденных состояний, а также аппроксимации инвариантов и другие полезные данные.

ЗАКЛЮЧЕНИЕ

В статье описан новый генератор символьных трасс в системе инсерционного моделирования. Главным отличием от предыдущих версий является добавление управляющей системы и возможность настройки на предметную область. Язык управляющей системы модели — это базовое подмножество языка UCM с базовыми протоколами, используемыми для реализации обязательств.

Семантика языка управления описана в формализме инсерционного моделирования, который допускает всевозможные расширения, имплементирующиеся на уровне действий среды. В частности, таким образом можно реализовать неиспользованные функциональности полного UCM.

Другие полезные расширения, которые будут реализованы в дальнейшем, — введение метрических характеристик отрезков путей между такими смежными точками переходов, как длина или время прохождения отрезка. Предполагается также ввести вероятностные распределения на альтернативных ветвлениях. Для повышения эффективности генерации трасс будет реализовано вычисление инвариантов, использование зависимостей по данным и другие средства.

СПИСОК ЛИТЕРАТУРЫ

1. Gilbert D.R., Letichevsky A.A. A universal interpreter for nondeterministic concurrent programming languages // Fifth Compulog Network Area Meeting on Language Design and Semantic Analysis Methods (26 Oct., 1996). — Aachen, 1996. — P. 14.
2. Letichevsky A., Gilbert D. A general theory of action languages // Cybernetics and System Analysis. — 1998. — N 1. — P. 16–37.

3. Letichevsky A., Gilbert D. A model for interaction of agents and environments // Lecture Notes in Computer Science. — 1999. — **1827**. — P. 311–328.
4. Milner R. A calculus of communicating systems. — H.: Springer-Verlag, 1980. — 192 p.
5. Milner R. Communication and concurrency. — NJ.: Prentice Hall, 1989. — 260 p.
6. Milner R. The polyadic π-calculus: a tutorial // Tech. Rep. ECS-LFCS-91-180. — 1991. — N 180. — P. 91–180.
7. Hoare C.A.R. Communicating sequential processes. — NJ.: Prentice Hall, 1985. — 256 p.
8. Bergstra, J.A., Klop J.W. Process algebra for synchronous communications // Information and Control. — 1984. — **60**, N 1/3. — P. 109–137.
9. Cardelli L., Gordon A.D. Mobile ambients // Foundations of Software Science and Computational Structures. Lecture Notes in Computer Science. — 1998. — **1378**. — P. 140–155.
10. Petri C.A. Kommunikation mit Automaten. — Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. — 123 p.
11. Hewitt C., Bishop P., Steiger R. A universal modular actor formalism for artificial intelligence // 3rd Intern. Joint Conf. on Artificial Intelligence (Aug., 1973). — Standford, 1973. — P. 235–245.
12. Letichevsky A. Algebra of behavior transformations and its applications // Structural Theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry. — 2005. — **207**. — P. 241–272.
13. Капитонова Ю.В., Летичевский А.А. Инсерционное моделирование // Праці міжнар. конф. «50 років Інституту кібернетики ім. В.М. Глушкова НАН України». — Київ: Вид-во ІК НАНУ, 2008. — С. 293–301.
14. Leveraging UML to deliver correct telecom applications in UML for real / S. Baranov, C. Jervis, V. Kotlyarov, A. Letichevsky, T. Weigert // Design of Embedded Real-Time Systems. — 2003. — P. 323–342.
15. Kapitonova J., Letichevsky A., Volkov V., Weigert T. Validation of embedded systems. — M.: CRC Press, 2005. — P. 58.
16. Basic protocols, message sequence charts, and the verification of requirements specifications / A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov, T. Weigert // Workshop on Integrated reliability with Telecommunications and UML Languages (Rennes, 4 Nov., 2005). — Rennes: Elsevier, 2005.— P. 42.
17. Basic protocols, message sequence charts, and the verification of requirements specifications / A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov, T. Weigert // Computer Networks. — 2005. — **47**. — P. 662–675.
18. System specification with basic protocols / A.A. Letichevsky, J.V. Kapitonova, V.A. Volkov, A.A. Letichevsky Jr., S.N. Baranov, V.P. Kotlyarov, T. Weigert // Cybernetics and System Analysis. — 2005. — N 4. — P. 3–21.
19. Letichevsky A., Letychevskyi O., Peschanenko V. Insertion modeling system // Lecture Notes in Computer Science. — 2011. — **7162.** — P. 262–274.
20. Летичевский А.А. Инсерционное моделирование // Управляющие системы и машины. — 2012. — № 6. — С. 3–14.
21. Insertion modeling in distributed system design / A. Letichevsky, J. Kapitonova, V. Kotlyarov, A. Letichevsky Jr, N. Nikitchenko, V. Volkov, T. Weigert // Problems in Programming. — 2008. — N 4. — P. 13–39.
22. Recommendation Z.151 User Requirements Notation (URN) // J. International Telecommunication Union, 2008. — 208 p.
23. Hassine J., Rilling J., Dssouli R. An ASM operational semantics for use case maps // Proceedings of the 13th IEEE International Conference on Requirements Engineering (Paris, 29 Aug. – 2 Sept., 2005). — IEEE Computer Society, 2005. — P. 467.
24. Губа А.Б., Шушпанов К.И. Инсерционная семантика плоских многопотоковых моделей языка UCM // Управляющие системы и машины. — 2012. — № 6. — С. 15–21.
25. Годлевский А.Б. Инсерционная семантика параллельных процедурных конструктов языка UCM // Там же. — 2012. — № 6. — С. 22–34.

Поступила 26.09.2014