УДК 519.686.2

А.А. Летичевский

Инсерционное моделирование

Введение

Инсерционное моделирование представляет собой направление, которое развивается на протяжении последнего десятилетия как подход к построению общей теории взаимодействия агентов и сред в сложных распределенных многоагентных системах. Основные понятия инсерционного моделирования (среда, агенты, функция погружения) были введены в работах [14, 20, 21], опубликованных в 90-х годах. Идейным прототипом инсерционного моделирования следует считать модель взаимодействующих управляющего и операционного автоматов, предложенную В.М. Глушковым еще в 60-х годах [1, 16] для описания структур вычислительных машин, а также ее развитие в теории дискретных преобразователей 70-х годов. В этих моделях система представляется в виде композиции двух автоматов – управляющего и информационного. Управляющий автомат играет роль агента, а информационный – роль среды, в которую погружен этот агент. Модели макроконвейерных параллельных вычислений, исследованные в 80-е годы [3], еще больше приблизились к современной модели взаимодействия агентов и сред. В этих моделях процессы, соответствующие параллельно работающим процессорам можно рассматривать как агенты, взаимодействующие в среде распределенных структур данных.

Другим источником инсерционного моделирования является общая теория взаимодействующих информационных процессов, которая сформировалась в 70-х годах и служит основой для современных исследований в этой области. Она включает в себя CCS (Calculus of Communicated Processes) [28, 29] и π-исчисление Р.Милнера [30], CSP (Communicated Sequential Processes) Т.Хоара [17], ACP (Algebra of Communicated Processes) [10] и много других различных ответвлений этих базовых теорий. Достаточно полный обзор классической теории процессов представлен в справочнике по алгебре процессов [11], изданном в 2001г. Модели, исследуемые в этих теориях, можно эквивалентным образом представить в виде композиции среды и погруженных в нее

агентов. Различные предложения по унификации общей теории взаимодействия в распределенных системах активно обсуждаются, начиная с 90х годов. К ним относятся чисто математические исследования на основе коалгебр [35], подход, предложенный Хоаром [18], логика условного переписывания Месегера [27] и др.

В последние годы инсерционное моделирование становится инструментом разработки прикладных систем верификации требований и спецификаций распределенных взаимодействующих систем [9, 19, 23, 24, 5]. Система VRS, разработанная в Украине по заказу фирмы Моторола с участием сотрудников Института Кибернетики, успешно применяется для верификации требований и спецификаций в области телекоммуникационных систем, встроенных систем и систем реального времени.

Данная работа носит обзорный характер. Основные понятия теории взаимодействия такие, как транзиционная система, трассовая и бисимуляционная эквивалентность, алгебра процессов и поведений, последовательная и параллельная композиция транзиционных систем предполагаются известными. Соответствующие определения можно найти в [22] и в [7].

Агенты и среды

Инсерционное моделирование занимается построением моделей и изучением взаимодействия агентов и сред в сложных распределенных многоагентных системах. Неформально основные положения парадигмы инсерционного моделирования можно сформулировать следующим образом.

1. Мир есть иерархия сред и агентов, погруженных в эти среды.
2. Агенты и среды есть сущности, эволюционирующие во времени и обладающие наблюдаемым поведением.
3. Погружение агента в среду изменяет поведение этой среды и порождает новую среду, готовую к погружению в нее новых агентов (если для них есть место в этой среде).
4. Среда, рассматриваемая как агент, может быть погружена в среду верхнего уровня.
5. Агенты могут погружаться в среду из среды верхнего уровня, а также производится внутренними агентами, уже погруженными в среду ранее.
6. Агенты и среды могут моделировать другие агенты и среды на различных уровнях абстракции.

Говоря об агентах и средах, мы имеем в виду как технические, так и реальные системы – физические, биологические и социальные, а взаимодействия, которые нас интересуют – это в первую очередь информационные взаимодействия, абстрагированные от физических процессов, которыми они сопровождаются.

Переходя к математическим уточнениям, вберем в качестве понятия агента наиболее абстрактное математическое понятие, моделирующее системы, эволюционирующие во времени. Таким образом, **агент** – это размеченная транзиционная система, состояния которой определяются с точностью до бисимуляционной или трассовой эквивалентности. Главным преимуществом понятия транзиционной системы по сравнению с другими моделями систем эволюционирующих во времени является разделение наблюдаемой части системы, которая выражается в действиях, и скрытой части системы, которая определяется ее внутренними состояниями.

Среда – это агент, который обладает функцией погружения. Более точно, среда – это набор $< E, C, A, \mathrm{Ins} >$, где E – это множество состояний среды (отождествляемых с поведениями), C – множество действий среды, A – множество действий агентов, погружаемых в среду, $\mathrm{Ins} : E \times F(A) \to E$ – функция погружения. Здесь $F(A)$ – полная алгебра поведений агентов с множеством действий A. Таким образом, всякая среда E допускает погружение любого агента с множеством действий A. Поскольку состояния транзиционных систем рассматриваются с точностью до бисимуляционной эквивалентности, то их можно отождествлять с поведениями и говорить о непрерывности функций. Основное требование к среде – это *непрерывность функции погружения*. Из этого предположения вытекает ряд полезных следствий. Например, тот факт, что функцию погружения можно задавать как наименьшую неподвижную точку системы функциональных уравнений. Результат Ins(e,u) погружения агента, находящегося в состоянии u, в среду, находящуюся в состоянии e, обозначается также как $e[u]$. Полагая $e[u,v] = (e[u])[v]$, получаем возможность говорить о совокупности агентов, погруженных в среду и рассматривать состояния среды вида $e[u_1, u_2, ..., u_m] = (...((e[u_1])[u_2])...)[u_m]$. Принимая во внимание, что среда есть агент, ее можно погружать в среду верхнего уровня, рассматривая многоуровневые среды вида $e[e_1[u_{11}, u_{12}, ...]_{E_1}, e_{21}[u_{21}, u_{22}, ...]_{E_2}, ...]_E$, где обозначение $e[u_1, u_2, ...]_E$ явно указывает среду E, которой принадлежит состояние e. Поведение u инициализированного агента определяет преобразование $[u] : E \to E$, определенное соотношением $[u](e) = e[u]$ и инсерционную эквивалентность агентов $\sim_E$

относительно среды E, определенную соотношением $u \sim_E v \Leftrightarrow [u] = [v]$. Эта эквивалентность, обычно более слабая, чем бисимуляционная, играет основную роль в приложениях, поскольку преобразования алгоритмических и программных реализаций агентов, функционирующих в некоторой среде, следует выполнять как преобразования, сохраняющие инсерционную эквивалентность.

Расширенная алгебра поведений. Напомним, что алгебра поведений имеет две основные операции: префиксинг $a.u$ и недетерминированный выбор $u+v$, где a – действие, u и v – поведения. Кроме того, в алгебре поведений есть две терминальные константы – успешное завершение Δ и тупиковое поведение 0 (неопределенное поведение и отношение аппроксимации, которые используются при построении полной алгебры поведений, нам сейчас не понадобятся). Всякое поведение можно представить в нормальной форме

$$u = \sum_{i \in I} a_i.u_i + \varepsilon_u,$$

где ε_u терминальная константа (если $I \neq \varnothing$, то $\varepsilon_u = \Delta$, поскольку $u + 0 = u$). Если все поведения в правой части рекурсивно также представлены в нормальной форме, то для u получится представление в виде ориентированного дерева, возможно бесконечного, с дугами размеченными действиями и некоторыми вершинами, отмеченными символом Δ. Любая конечная часть такого дерева (т.е. конечное множество конечных путей, которые начинаются в корне) называется *префиксом* поведения.

Система поведений

$$u_i = \sum_{j \in J(i)} a_{ij}.u_j + \varepsilon_i, i \in I, J(i) \subseteq I$$

может рассматриваться как система рекурсивных определений. Если она конечна, то поведения называются *конечно определенными*. Такие поведения могут быть отождествлены с состояниями конечной транзиционной системы с отношением переходов $u_i \xrightarrow{a_{ij}} u_j, i \in I, j \in J(i) \subseteq I$ и с множеством заключительных состояний $U_\Delta = \{u_i \mid \varepsilon_i = \Delta\}$.

Расширим алгебру поведений, добавив к ней функции погружения, рассматриваемые как бинарные операции на соответствующих множествах. Формально, мы должны рассмотреть многоосновную алгебру, которая получается, если зафиксировать некоторое семейство сред $(E_\eta)_{\eta \in \mathrm{H}}$ и алгебр поведений агентов $F(A_\vartheta)_{\vartheta \in \Theta}$ в качестве основных множеств. Тогда функция погружения будет определена как $Ins_{\eta\vartheta} : E_\eta \times F(A_\vartheta) \to E_\eta$ и выражения, рассмотренные в начале этого раздела, можно будет

рассматривать как алгебраические выражения соответствующей многоосновной алгебры. Среда, определенная с помощью такой алгебры называется *многоуровневой средой*.

Основные свойства функций погружения. Будем рассматривать только такие среды, для которых имеет место тождество $e[u,\Delta]=e[u]$. Состояние среды e называется *неразложимым*, если из того, что $e=e'[u']$ следует, что $e=e', u'=\Delta$. Множество неразложимых состояний среды назовем ее *ядром*. Неразложимое состояние среды соответствует состоянию до погружения в нее каких бы то ни было агентов, а среда с пустым ядром предполагает, что в нее изначально погружено бесконечное множество агентов. Среда называется *конечно-разложимой*, если всякое ее состояние можно представить в виде $e[u_1,...,u_m]$ с неразложимым e. В дальнейшем, если не оговорено противное, будут рассматриваться только конечно-разложимые среды.

Одношаговое погружение определяется правилом

$$\frac{e\xrightarrow{a}e', u\xrightarrow{b}u'}{e[u]\xrightarrow{c}e'[u']}P(a,b,c)$$

Здесь $P(a,b,c)$ – разрешающий предикат. Правило применимо, только в том случае, когда разрешающий предикат истинен. Для конечно-разложимой среды, в которую погружено несколько агентов, следует воспользоваться производным правилом:

$$\frac{e[u_1,...,u_{m-1}]\xrightarrow{c}e'[u'_1,...,u'_{m-1}], u_m\xrightarrow{b}u'_m}{e[u_1,...,u_m]\xrightarrow{d}e'[u'_1,...,u'_m]}P(c,b,d),$$

где e – неразложимое состояние среды.

Другая форма правила одношагового погружения:

$$P(a,b,c)\Rightarrow (a.e'+e'')[b.u'+u'']\xrightarrow{c}e'[u']$$

Одношаговое правило можно понимать следующим образом. Если агент хочет выполнить действие b (одно из возможных), то среда разрешает это действие, если у нее найдутся два действия a и c, такие, что имеет место $P(a,b,c)$. В этом случае переход зависит только от того, что среда может сделать за один шаг.

Более общее правило может быть сформулировано, если в разрешающий предикат добавить поведение среды: $P(e,a,b,c)$. Но для того, что бы обеспечить непрерывность функции погружения, нужно потребовать непрерывность разрешающего предиката: *он должен зависеть только от конечного префикса поведения e*. Такое правило называется правилом *одношагового префиксного погружения* (head insertion). Наконец, еще более

общий случай получается, если разрешающий предикат зависит и от поведения агента: $P(e,u,a,b,c)$. Такое погружение называется *прогнозирующим*.

Рассмотренное правило не является полным. Требуются дополнительные правила для терминальных состояний среды и агента, а также для недетерминированного выбора. Примерами таких правил могут быть тождества $0[u]=0, \Delta[u]=\Delta, e[\Delta]=e$, которые можно дополнить тождеством $e[0]=0$ для неразложимого состояния среды *e*.

Функция погружения называется *аддитивной*, если

$$e[u+v]=e[u]+e[v], (e+f)[u]=e[u]+e[v]$$

Аддитивность функции погружения может означать, что как среда, так и агент делают свой выбор независимо, но так, что бы была возможность продолжать движение. При этом следует иметь в виду, что если нет правила, по которому возможен переход из $e[u]\neq\Delta$, то $e[u]=0$. Кроме аддитивных, на практике встречаются коммутативные погружения с тождествами $e[u,v]=e[v,u]$ и, как частный случай коммутативных, параллельные погружения: $e[u,v]=e[u\parallel v]$.

Атрибутные среды. Для многих практических приложений понятие среды и агента является слишком абстрактным, поскольку игнорирует структуру состояний среды и агентов. Кроме того, при переходе в терминальное состояние теряется информация о состоянии среды, если она имеет структуру. Всю необходимую информацию можно, разумеется, передавать через действия, но это часто выглядит неестественно и громоздко. Для того, что бы решить эту проблему, в [5] было введено понятие атрибутной транзиционной системы. Атрибутные транзиционные системы отличаются от размеченных тем, что в них размечены не только переходы, но и состояния. Алгебра поведений для атрибутных систем отличается тем, что поведения могут быть размечены атрибутными разметками. Для этой цели кроме префиксинга вводится еще одна операция, операция разметки $\varphi:u$, где u – поведение, φ – разметка. Теперь вся необходимая информация о состоянии среды может передаваться через ее разметку. В частности теперь может быть много терминальных констант, соответствующих успешному завершению или тупику, поскольку они могут иметь различные разметки.

Атрибутные среды строятся на основе некоторой *логической базы*. Эта база включает в себя набор типов (целые, вещественные, перечислимые, символьные, поведения и т.д.), интерпретированных на некоторых областях данных, символы для обозначения констант из этих областей, и набор типизированных функциональных и

предикатных символов. Часть их этих символов интерпретирована (например, арифметические операции и неравенства, равенство для всех типов и т.д.). Неинтерпретированные функциональные и предикатные символы называются *атрибутами*. Неинтерпретированные функциональные символы арности 0 называются *простыми атрибутами*, остальные – *функциональными атрибутами* (неинтерпретированный предикатный символ рассматривается как функциональный с бинарной областью значений). Функциональные символы используются для определения структур данных таких, как массивы, списки, деревья.

Над логической базой атрибутной среды строится *базовый логический язык*. Обычно это язык первого порядка, возможно с кванторами. При необходимости могут включаться некоторые модальности темпоральной логики. *Атрибутным выражением* называется простой атрибут или выражение вида $f(t_1,...,t_n)$, где f – функциональный атрибут арности n, $t_1,...,t_n$ – уже построенные атрибутные выражения или константы подходящих типов. Если все выражения $t_1,...,t_n$ являются константами, то атрибутное выражение называется *константным*.

В общем случае ядро атрибутной среды состоит из формул базового языка. Атрибутные среды делятся на два класса: *конкретные* и *символьные*.

Неразложимое состояние конкретной атрибутной среды представляет собой формулу вида $t_1 = a_1 \wedge ... \wedge t_n = a_n$, где $t_1,...,t_n$ – различные константные атрибутные выражения, $a_1,...,a_n$ – константы. Обычно такую формулу представляют в виде частичного отображения с областью определения $\{t_1,...,t_n\}$ и областью значений равной множеству всех констант. Конкретные атрибутные среды удобно использовать для формализации операционной семантики языков программирования и языков спецификации программных систем. Состояние среды – это состояние памяти (структур данных, объектов, сетевых конструкций, каналов и т.п.). Программы – это агенты, погруженные в среду. Для параллельных языков программирования взаимодействие осуществляется через общую память или обмен сообщениями. При взаимодействии через общую память основную роль играет параллельная композиция поведений (асинхронная или синхронизируемая). Обмен сообщениями предполагает использование специальных структур данных в среде для организации взаимодействия. Действия агентов в конечном итоге сводятся к присваиваниям ($x := t$, где x – атрибутное выражение, t – алгебраическое

выражение базового языка) и проверкам условий ($\text{check}\,\alpha$, где α – формула базового языка). Соответствующие правила имеют вид:

$$\frac{e \xrightarrow{x:=t} e', u \xrightarrow{x:=t} u'}{e[u] \xrightarrow{x:=t} e'[u']} P(e,x,t)$$

$$\frac{u \xrightarrow{\text{check}\,\alpha} u'}{e[u] \xrightarrow{\text{check}\,\alpha} e[u']} e \models \alpha$$

Разрешающее условие для присваивания может накладывать ограничение на однозначность значений для аргументов левой части и однозначность значения правой части присваивания.

Программные конструкции (различные виды циклов, ветвления и т.п.) в большинстве случаев достаточно просто моделируются рекурсивными уравнениями в алгебре поведений. Например,

$$\text{if}\ \alpha\ \text{then}\ P\ \text{else}\ Q = \text{check}\ \alpha.P + \text{check}\ \neg\alpha.Q,$$

$$\text{while}\ \alpha\ \text{do}\ P = \text{if}\ \alpha\ \text{then}(P; \text{while}\ \alpha\ \text{do}\ P)\text{else}\ \Delta$$

Неразложимые состояния символьной среды – это формулы базового языка произвольного вида. Конкретные и символьные состояния атрибутной среды вида $F[u_1, u_2, ...]$ также рассматриваются как формулы. Именно, предполагается, что все погруженные агенты имеют уникальные имена (это могут быть, например, их атрибутные отметки). Далее, среди атрибутов базового языка есть функциональный атрибут state, определенный на именах агентов и принимающий значения в области их поведений. Если $m_1, m_2, ...$ имена агентов $u_1, u_2, ...$, то $F[u_1, u_2, ...] \Leftrightarrow F \wedge (\text{state}(m_1) = u_1) \wedge (\text{state}(m_2) = u_2) \wedge ...$.

Локальные свойства

Наиболее известные способы спецификации программ и систем (программных и технических) находятся в области темпоральной, динамической и некоторых других видов модальных логик [13]. Однако, в большинстве случаев они применяются тогда, когда уже известна достаточно подробная модель системы и решается проблема проверки свойств этой системы, заданных в логической форме (model checking).

Другим способом описания требований и спецификации систем является описание локальных поведенческих свойств системы. В математической форме речь идет о свойствах отношения переходов транзиционной системы, а в случае, когда система представлена в виде композиции среды и агентов, речь идет об описании функции погружения.

Мы будем рассматривать локальные свойства системы, представленной как конкретная или символьная атрибутная среда. В общем случае локальное свойство атрибутной среды имеет вид формулы $\forall x(\alpha \rightarrow < u > \beta)$, где x – список (типизированных) параметров, α и β – формулы базового языка, u – процесс (конечное поведение специфицируемой системы). Формула α называется предусловием, а формула β – постусловием локального свойства. От параметров могут зависеть как условия, так и поведение системы. Локальное свойство может рассматриваться как формула темпоральной логики, выражающая тот факт, что, если (для подходящих значений параметров) состояние системы удовлетворяет условию α, то, поведение u может быть инициировано и, после его успешного завершения, разметка нового состояния будет удовлетворять условию β. Не трудно проследить аналогию между тройками Хоара (формулы динамической логики) и локальными свойствами систем. Другая аналогия – это продукции, широко применяемые при описании систем искусственного интеллекта.

Локальные свойства, используемые во входном языке системы VRS для представления инсерционных моделей распределенных взаимодействующих систем, носят название *базовых протоколов*. Они привязаны к базовому языку системы VRS, а для представления поведения системы используются MSC диаграммы [33]. Обычно базовые протоколы определяются независимо друг от друга и на них априори не задано никаких отношений следования, которые определяли бы порядок их применения. Поэтому семантика спецификаций на основе базовых протоколов не очевидна. В процессе исследования семантики языка базовых протоколов было разработано несколько подходов к построению такой семантики. В этих подходах существенную роль играют понятия абстракции и конкретизации, на которых остановимся более подробно.

Абстракции. При работе с большими системами, такими как телекоммуникационные системы, сети типа Интернет, многопроцессорные системы с большим числом компонент, невозможно манипулировать полными описаниями их состояний. Поэтому, приято заменять состояния таких систем различного рода абстрактными объектами. В инсерционном моделировании, в качестве абстракций больших систем используются атрибутные системы, состояния которых размечаются формулами базового языка. Если состояния сами представлены формулами, получаем символьные атрибутные модели. Для изучения соотношений между абстрактными и конкретными моделями в [5] были введены понятия абстракции и конкретизации атрибутных транзиционных систем.

Предполагается, что состояния исходной системы и формулы базового языка связаны отношением (семантического) следования $s \models \alpha$, которое означает, что формула α истинна на разметке состояния s (разметка может не быть формулой). Если же состояние s не размечено, то $s \models \alpha$ означает, что α есть тождественно истинная формула базового языка, т.е. истинна на любых разметках. Отношение следования определено также на состояниях абстрактной системы, которые размечены формулами базового языка, и в этом случае $s \models \alpha$ означает, что α есть следствие формулы, которая размечает данное состояние.

Пусть S и S' – две атрибутные (не обязательно различные) системы с размеченными состояниями, одной и той же логической базой и множеством действий. Пусть **BL** есть базовый язык. Определим отношение абстракции $\mathbf{Abs} \subseteq S \times S'$ таким образом, что

$$(s, s') \in \mathbf{Abs} \Leftrightarrow \forall(\alpha \in \mathbf{BL})((s \models \alpha) \Rightarrow (s' \models \alpha)).$$

Система S называется прямой абстракцией системы S', а система S' *прямой конкретизацией* системы S, если существует непустое отношение $\varphi \subseteq \mathbf{Abs}^{-1}$ которое является отношением моделирования, т.е. имеет место следующее утверждение:

$$\forall(s \in S, s' \in S')((s', s) \in \varphi \wedge s \xrightarrow{a} t \Rightarrow \exists(t' \in S')(s' \xrightarrow{a} t' \wedge (t', t) \in \varphi)).$$

Система S называется *обратной абстракцией* системы S', а система S' – обратной конкретизацией системы S, если существует непустое отношение $\varphi \subseteq \mathbf{Abs}^{-1}$, которое является отношением обратного моделирования, т.е. имеет место следующее утверждение:

$$\forall(s \in S, s' \in S')((s', s) \in \varphi \wedge s' \xrightarrow{a} t' \Rightarrow \exists(t \in S)(s \xrightarrow{a} t \wedge (t', t) \in \varphi))$$

Система и ее абстракция связаны следующим образом. Если в прямой абстракции системы из некоторого состояния достижимо заданное свойство, то оно также достижимо из некоторого состояния системы. Для обратной абстракции имеет место двойственное свойство: если из некоторого состояния системы достижимо состояние, в котором заданное свойство нарушается, то из некоторого начального состояния ее обратной абстракции также достижимо состояние, в котором это свойство нарушается.

Прикладное значение этих утверждений состоит в том, что, комбинируя прямую и обратную абстракции можно получать различные результаты по верификации систем и генерации тестов. Например, если есть подозрение в том, что система имеет ошибку, то найти ее можно с помощью прямой абстракции, убедится же в том, что в системе нет

ошибок можно с помощью обратной абстракции. Обратную абстракцию удобно также использовать для генерации тестов, обеспечивающих определенную степень покрытия требований, выраженных в виде систем базовых протоколов.

Семантика. В [5] была построена общая семантика систем базовых протоколов, сопоставляющая каждой системе *P* базовых протоколов некоторую фиксированную транзиционную систему **S**(*P*), определенную с помощью системы уравнений в алгебре поведений. Эта система уравнений строится следующим образом. Сначала по системе *P* строится множество P_{inst} *конкретизированных* базовых протоколов. Конкретизированный базовый протокол $\alpha(z) \to < u(z) > \beta(z)$ получается из формулы $\forall x(\alpha(x) \to < u(x) > \beta(x))$ путем подстановки конкретного набора значений *z* параметров вместо *x*. Далее на множестве действий среды определяется отношение *перестановочности* и на базе этого отношения определяется *частично-последовательная композиция* (()*()) поведений. Для атрибутных систем перестановочность может означать их информационную независимость. На множестве конкретизированных свойств определяются функции $app(F,\alpha) = 0,1$ и $pt(F,\beta) = F'$ (*предикатный трансформер*). Первая используется для определения применимости свойства с предусловием α к состоянию среды *F*, вторая – для вычисления нового состояния среды с помощью постусловия β. Основное требование к функции pt состоит в том, что бы условие β было истинным на состоянии F'. Система уравнений для поведений $\mathbf{S}_F$ системы **S**(*P*) в состояниях *F*, имеет вид:

$$\mathbf{S}_F = \sum_{\substack{app(F,\alpha)=1,\\ \alpha \to < u > \beta \in P_{inst}}} (u * \mathbf{S}_{pt(F \wedge \alpha, \beta)})$$

Для определения применимости протоколов используются два метода: *экзистенциальный* и *универсальный*. Определение применимости по экзистенциальному методу означает выполнимость конъюнкции $F \wedge \alpha$, применимость по универсальному методу означает общезначимость импликации $F \to \alpha$.

Другой подход к определению семантики базовых протоколов, также рассмотренный в статье [5], состоит в сопоставлении системе базовых протоколов *P* некоторого множества *C*(*P*) конкретных атрибутных систем, которые по определению объявляются реализациями системы *P*. Было показано, что система **S**(*P*) является абстракцией любой системы из класса *C*(*P*). Эта абстракция будет прямой, если выбран универсальный метод и обратной, если выбран экзистенциальный метод.

В [26] был предложен новый подход, основанный на общем (абстрактном) понятии реализации системы базовых протоколов. В этой работе базовые протоколы рассматривались с привязкой к интерпретированным MSC диаграммам системы VRS, используемым в качестве способа задания процессов системы. Но понятие реализации легко обобщается и на произвольные локальные свойства.

В соответствии с этой семантикой, каждая спецификация *P* на языке базовых протоколов определяет некоторый класс атрибутных транзиционных систем, которые объявляются реализациями системы *P*. Этот класс определяется аксиоматически и включает в себя как конкретные, так и символьные реализации. На состояниях реализации по-прежнему должно быть определено отношение истинности $s \models \alpha$ и отношение перестановочности действий, которое определяет частично-последовательную композицию. Основная аксиома, определяющая реализацию, имеет вид

$$appl(s,\alpha), s \xrightarrow{[B]*[C]} s' \Rightarrow s' \models \beta$$

Здесь *B* и *C* – базовые протоколы, [*B*] и [*C*]– их процессы. Предполагается, что [*B*] и [*C*] перестановочны. Условие α является предусловием базового протокола *B*, а условие β его постусловием.

Предикатный трансформер. Каждая формула базового языка соответствует множеству конкретных состояний, на котором она истинна (покрывает это множество). Поэтому функция погружения для символьной среды должна быть согласованной с функцией погружения соответствующей конкретной среды. Эта согласованность определяется в терминах предикатного трансформера, который сам по себе определяет переходы в как в конкретной, так и в символьной атрибутной среде. Именно, если в конкретной системе $s \xrightarrow{\beta} s'$ и $s \models F$, то $s' \models pr(F,\beta)$. Таких функций может быть много и простейшая из них определяется как $pr(F,\beta) = \beta$. Но при этом полностью теряется информация о предыдущем состоянии *F*. Между тем, часть этой информации может переходить в новое состояние. Например, если постусловие меняет значения только нескольких атрибутов, то остальные не меняют своих значений. Среди всех функций, определяющих предикатный трансформер, может существовать наисильнейшая. Она должна удовлетворять обратному условию: если $s' \models pr(F,\beta)$ и $s \xrightarrow{\beta} s'$, то должно быть $s \models F$. Вопрос о существовании сильнейшего предикатного трансформера и возможности выразить его средствами базового языка, зависит от этого языка. В работе

[6] был построен сильнейший предикатный трансформер для входного языка системы VRS.

Рассмотрим более детально структуру предикатного трансформера для случая атрибутов простых типов. Пусть базовый протокол имеет вид

$$\forall x(\alpha(z,s,x) \rightarrow < u(z,s,x) > \beta(s,x)), \qquad (1)$$

где x – список параметров, z – список атрибутов, которые входят в предусловия, но не входят в постусловия, s – список атрибутов, которые входят в постусловия. Пусть $F(z,s)$ – формула текущего состояния. Основное предположение, которое используется для построения предикатного трансформера, состоит в том, что после завершения базового протокола изменить свое значение могут только те атрибуты, которые явно входят в постусловие. Условием применимости базового протокола служит одно из двух утверждений $\forall(z,s)(F(z,s) \rightarrow \exists x(\alpha(z,s,x))$ или $\exists(z,s)(F(z,s) \wedge \exists x(\alpha(z,s,x))$. При использовании первого условия получаем прямую абстракцию всех конкретных моделей, а при использовании второго – обратную. Предикатный трансформер должен сформировать сильнейшее постусловие, которое следует из условия применимости. Вот это условие (оно будет одним и тем же как для прямой, так и для обратной абстракции):

$$\exists(x,y)(F(z,y) \wedge \alpha(z,y,x) \wedge \beta(s,x))$$

После формирования этого условия, предикатный трансформер должен удалить кванторы, если это возможно. Таким образом, для реализации абстрактной модели базовых протоколов нужны дедуктивные средства (прувер и солвер для базового языка).

Если в постусловиях используются присваивания, то формула модифицируется с использованием обычной (Хоаровской) семантики присваиваний. Возможны также обобщения конструкции предикатного трансформера в случае, когда допускаются структуры данных такие как массивы, списки и т.п.

Используя семантику базовых протоколов можно генерировать трассы и сценарии функционирования системы как на уровне конкретных, так и на уровне абстрактных моделей. Эти трассы и сценарии можно использовать для динамической проверки различных свойств системы, например, доказывать инвариантность или достижимость условий, выразимых в базовом языке, а также условий, выразимых с помощью различных видов темпоральной логики. Используя дедуктивные средства, можно также проводить статическую верификацию. Например, для доказательства инвариантности свойства γ, достаточно доказать, что это свойство сохраняется всеми базовыми протоколами, а сохранение свойства $\gamma = \gamma(z,s)$ базовым протоколом (1) означает истинность формулы

$$\forall(z,s,x)(\exists y(\gamma(z,y)\wedge\alpha(z,y,x))\wedge\beta(s,x)\rightarrow\gamma(z,s)),$$

смысл которой состоит в том, что, если в какой-то момент γ было истинным и был применим протокол (1), то после его применения свойство γ остается истинным.

Верификация. Язык базовых протоколов, реализованный в системе VRS, допускает использование атрибутов числовых и символьных типов (свободные термы), массивов, списков и функциональных типов данных. Дедуктивная система обеспечивает доказательство утверждений в теории первого порядка, представляющей собой интеграцию теорий целочисленных и вещественных линейных неравенств, перечислимых типов данных, свободных (неинтерпретированных) функциональных символов и теорию очередей. В дедуктивной системе успешно доказываются или опровергаются только некоторые классы формул, поэтому, при верификации иногда могут быть получены отказы для некоторых промежуточных запросов. На практике такие отказы происходят достаточно редко и в большинстве случаев не влияют на получение окончательного результата.

В качестве языка процессов используется язык MSC [33] с инсерционной семантикой [22, 25]. Система также допускает использование языка SDL [34] и соответствующих диаграммных представлений языка UML [37].

Основные средства системы VRS – это конкретный и символьный генераторы трасс и средства статической верификации, которые включают в себя проверку полноты и непротиворечивости базовых протоколов и проверку инвариантности свойств.

Система успешно применялась в ряде практических проектов из области телекоммуникации, встроенных систем, телематики и др. Количество требований, формализованных в виде базовых протоколов достигало до 10 000, а количество атрибутов – до 1000. Методика использования системы включает в себя ряд технологий для верификации систем и генерации тестов.

Одна из типичных технологий применяется для проверки полноты и непротиворечивости базовых протоколов. Два базовых протокола называются *непротиворечивыми*, если при любой конкретизации этих протоколов их предусловия не могут быть одновременно истинными. Противоречивость двух протоколов означает, что для некоторых состояний, выбор протокола, который применяется в этих состояниях, происходит недетерминированным образом. Этот недетерминизм может быть нежелательным, и система сообщает о противоречивости разработчику как предупреждение. Система базовых протоколов называется полной, если для любого

конкретного состояния существует хотя бы один базовый протокол, применимый в этом состоянии. Неполнота системы базовых протоколов означает возможность тупикового состояния (dead lock).

В вышеприведенной формулировке условия непротиворечивости и полноты проверяются статически путем анализа предусловий базовых протоколов с использованием дедуктивных средств. На самом деле непротиворечивость и полноту следует проверять не на всех состояниях, а лишь на достижимых из множества начальных состояний системы. Если эти состояния можно описать формулой базового языка, то снова можно применить статическую проверку этих свойств, включив их в соответствующие формулы.

Если система непротиворечива и полна, проверка закончена. Если же найден случай противоречия или неполноты и он не принят разработчиком в силу того, что не ясно, достижимо ли соответствующее состояние, то возникает новая задача – проверка недостижимости условия, которое выражает нарушение требования непротиворечивости или полноты. Поскольку недостижимость условия означает инвариантность его отрицания, то можно снова применить статический анализ, пытаясь доказать соответствующие свойства. Если это удается, задача решена, в противном случае можно пытаться усилить соответствующие свойства либо применить динамическую верификацию, т.е. конкретную или символьную генерацию трасс, пытаясь доказать или опровергнуть достижимость искомых свойств.

Инсерционные машины

Для реализации инсерционных моделей некоторого класса используется интерпретатор моделей, который мы называем *инсерционной машиной*. Инсерционная машина состоит из трех основных компонент:

1. *Модельный драйвер*. Эта компонента управляет движением модели по дереву ее поведения, вычисляя переходы из текущего в новое состояние.
2. *Анфолдер поведений агентов*. Текущее состояние модели представляется в виде алгебраического выражения в расширенной алгебре поведений. Входной язык допускает использование рекурсивных определений для описания поведений агентов и структур данных для описания поведений или состояний среды. Анфолдер поведений использует эти описания для того, что бы получить разложение состояния в виде выражения $E[u_1, u_2, ...]$, где E – неразложимое состояние среды.

3. *Интерактор среды*. Приводит состояние среды к нормальной форме

$$E = \sum_{i \in I} a_i . E_i + \varepsilon ,$$

используя описание функции погружения. После приведения к нормальной форме драйверу модели остается только выбрать направление движения $i \in I$ и осуществить переход $E \xrightarrow{a_i} E_i$ или остановиться, если $\varepsilon = \Delta$.

Различаются два типа инсерционных машин: *машины реального времени* или *интерактивные машины* и *аналитические инсерционные машины*. Машины реального времени работают в реальной или виртуальной среде, взаимодействуя с внешней средой в реальном времени. Аналитические машины предназначены для анализа моделей, исследования их свойств, решения задач на инсерционных моделях и т.п. Соответственно модельные драйверы также делятся на интерактивные и аналитические. Интерактивный драйвер после нормализации состояния должен выбрать в точности один переход и выполнить его путем передачи своего действия во внешнюю среду. Интерактивная машина с точки зрения внешнего наблюдателя функционирует как агент, погруженный во внешнюю среду с функцией погружения, определяющей законы функционирования этой среды. Внешняя среда, например, может изменить поведенческий префикс состояния внутренней среды в соответствии со своей функцией погружения.

Интерактивный драйвер может быть организован достаточно сложно. Он может иметь критерии успешного функционирования и работать как интеллектуальная система, собирая информацию о прошлом, создавая модель внешней среды и улучшая алгоритмы принятия решений по выбору действий с целью повышения уровня успешности своего функционирования. Кроме того, интерактивный драйвер может иметь интерфейс для обмена физическими сигналами с внешней средой (например, прием визуальной или акустической информации, информации о положении в пространстве и т.п.).

Аналитическая инсерционная машина, в противоположность интерактивной, может рассматривать различные варианты принятия решений о действиях, возвращаясь в точки выбора и рассматривая различные пути в дереве поведения системы. Модель системы может включать в себя также и модель внешней среды для этой системы. В общем случае аналитическая машина осуществляет поиск состояний обладающих заданными свойствами (достижимость целевых состояний) или состояний, в которых заданные свойства нарушаются. Внешняя среда аналитической машины может быть представлена пользователем, который взаимодействует с машиной, формулируя задачи и управляя

активностью машины. Аналитические машины, обогащенные логикой и дедуктивными средствами, используются для символьного моделирования систем.

Система VRS помимо инсерционной машины, использующей специализированный входной язык описания требований и спецификаций, содержит средства для статического анализа инсерционных моделей, генерации тестовых наборов, генерации кода и ряд других средств, поддерживающих разработку распределенных взаимодействующих систем.

В настоящее время в Институте кибернетики им. В.М.Глушкова ведется работа над созданием новой системы инсерционного моделирования IMS (Insertion Modeling System). Эта система представляет собой среду для разработки инсерционных машин различного типа. Она содержит в своем составе ряд прототипов инсерционных машин с различными типами функций погружения и средствами их развития. Базовой системой программирования для IMS является система алгебраического программирования APS, которая обеспечивает гибкость и эффективность при разработке новых инсерционных машин.

В статье [8], представленной в этом выпуске, описана инсерционная машина для доказательного программирования. Она ориентирована на верификацию императивных аннотированных программ, использующих различные языки программирования.

Когнитивные архитектуры

Когнитивные архитектуры или интеллектуальные агенты – это активно развивающееся в последние годы направление в искусственном интеллекте. Целью этого направления является построение модели человеческого разума, обладающей такими же универсальными возможностями как человеческий разум. В отличие от специализированных систем искусственного интеллекта, способных эффективно решать специфические классы задач, когнитивные архитектуры должны адаптироваться к различным новым, часто неожиданным ситуациям, возникающим в результате взаимодействия с внешней средой, обучаться, ставить цели, улучшать свое поведение и т.п. Недавно возникло новое направление BICA (Biologically Inspired Cognitive Architectures, Биологически Мотивированные Когнитивные Архитектуры) [12]. В рамках этого направления было выполнено сравнение большого количества существующих архитектур, выделены основные черты, которыми должны обладать такие архитектуры, ежегодно проводятся конференции, собирающие специалистов из различных областей, заинтересованных в продвижении в этом направлении.

В настоящее время мы начали исследовать возможность создания когнитивной архитектуры на основе инсерционного моделирования.

Наши представления в краткой форме могут быть выражены следующим образом. Инсерционная когнитивная архитектура – это инсерционная машина, которая осознает себя, имеет центр оценки успешности своего поведения и стремится к многократному достижению максимального успеха. Как агент эта машина погружена в свою внешнюю среду и имеет средства взаимодействия с ней. Внутренняя среда создает и совершенствует свою модель и модель внешней среды и населяющих ее агентов. Все эти модели строятся как многоуровневые инсерционные среды. Для достижения своих целей использует основные механизмы, выработанные в области биологически мотивированных когнитивных архитектур, а также в области инженерных проектов, поддерживающих решение сложных интеллектуальных задач.

Уверенность в правильности нашего направления основана на следующих гипотезах:

1. Многоуровневая инсерционная среда может быть с успехом использована для моделирования шести уровней неокортекса (коры головного мозга). Двигаясь от нижних уровней к высшим, мы повышаем уровни абстракции, переходя к все более абстрактным символьным моделям. Агенты, погруженные в среды верхнего уровня, покрывают классы моделей нижнего уровня.
2. Человеческий разум должен содержать механизмы, подобные драйверам инсерционных машин, позволяющие активизировать модели внешних и внутренних сред, произведенные в результате накопления жизненного опыта, обучения и унаследованные генетически.
3. Когнитивная архитектура функционирует как инсерционная машина реального времени, осознавая себя с помощью внутренней среды самого верхнего уровня. Эта среда управляет всеми средами (агентами) нижнего уровня активизируя или тормозя их деятельность. Агенты всех уровней работают параллельно и независимо до тех пор, пока не попадают под контроль сред верхних уровней, осуществляя мобильные погружения в них.

Литература

[1] В.М.Глушков, теория автоматов и вопросы проектирования структур цифровых машин, Кибернетика 1,1965, 3-11, 104.

[2] Годлевский А.Б., Предикатные преобразователи в контексте символьного моделирования транзиционных систем, Кибернетика и системный анализ 2010, №4, 91-99, 192.

[3] Ю.В.Капитонова, А.А.Летичевский, Математическая теория проектирования вычислительных систем, Москва, Наука, 1988, 295с.

[5] Летичевский А.Ад., Капитонова Ю.В., Волков В.А., Летичевский А.А., Баранов С.Н., Котляров В.П., Вейгерт Т., Спецификация систем с помощью базовых протоколов, Кибернетика и системный анализ, 4, 2005, 3-21, 192.

[6] Летичевский А.А., Годлевский А.Б., Летичевский А.А. (мл.), Потиенко С.В., Песчаненко В.С., Свойства предикатного трансформера системы VRS, Кибернетика и системный анализ 2010, №4, 3-16,192.

[7] А.А.Летичевский, Ю.В.Капитонова, Инсерционное моделирование, Праці міжнародної конференції «50 років інституту кібернетики ім. В.М.Глушкова НАН України», Київ 2008, стор. 293-301, 364.

[8] А.А.Летичевский (мл), М.К.Мороховец, В.В.Песчаненко, Система доказательного программирования, настоящий выпуск журнала.

[9] S. Baranov, C. Jervis, V. Kotlyarov, A. Letichevsky, and T. Weigert, Leveraging UML to deliver correct telecom applications in UML for Real: Design of Embedded Real-Time Systems by L.Lavagno, G. Martin, and B. Selic (editors), 323–342, Kluwer Academic Publishers, 2003.

[10] J. A. Bergstra and J. W. Klop, Process algebra for synchronous communications, *Information and Control* **60** (1/3) (1984), 109–137.

[11] J. A. Bergstra, A. Ponce, and S. A. Smolka, eds., *Handbook of Process Algebra*, North-Holland, 2001, 1342.

[12] http://bicasociety.org/

[13] J.Bradfield, C.Stirling, Modal Logic and mu-Calculi: An introduction, in J. A. Bergstra, A. Ponce, and S. A. Smolka, eds., *Handbook of Process Algebra*, North-Holland, 2001, 293-330, 1342.

[14]. D.R. Gilbert, A.A. Letichevsky, A universal interpreter for nondeterministic concurrent programming languages, in: M. Gabbrielli (Ed.), Fifth Compulog network area meeting on language design and semantic analysis methods, September 1996.

[15] R. J. Glabbeek, The linear time—branching time spectrum the semantics of concrete, sequential processes, in: *Handbook of Process Algebra* (J. A. Bergstra, A. Ponce, and S. A. Smolka, eds.), North-Holland, 2001, 1342.

[16] V. M. Glushkov and A. A. Letichevsky, Theory of algorithms and descrete processors, in: *Advances in Information Systems Science* (J. T. Tou, ed.), vol. 1, Plenum Press, 1969, 1-58.

[17] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[18]C. A. R. Hoare and He Jifeng, Unifying Theories of Programming, Prentice Hall, 1999

[19] J. Kapitonova, A. Letichevsky, V. Volkov, and T. Weigert. Validation of Embedded Systems. In R. Zurawski, editor. The Embedded Systems Handbook. CRC Press, Miami, 2005.

[20]. A. Letichevsky and D. Gilbert. A general theory of action languages. Кибернетика и Системный Анализ, 1, 1998, 16-36, 192.

[21] A. Letichevsky and D. Gilbert, Interaction of agents and environments, in: *Resent trends in Algebraic Development technique, LNCS 1827* (D. Bert and C. Choppy, eds.), Springer-Verlag, 1999.

[22] A.A Letichevsky, J.V. Kapitonova, V.P.Kotlyarov, A.A.Letichevsky Jr, V.A.Volkov. Semantics of timed MSC language, Kibernetika and System Analysis, 2002.

[22] A.Letichevsky. Algebra of behavior transformations and its applications, in V.B.Kudryavtsev and I.G.Rosenberg eds. Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry – Vol. 207, pp. 241-272, Springer 2005.

[23] A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V.Kotlyarov, T. Weigert, Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications, ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages) , Rennes, 4 November 2005.

[24] Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V.Kotlyarov, T. Weigert. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. Computer Networks, 47, 2005, 662-675.

[25] A.A. Letichevsky, J.V. Kapitonova, V.P. Kotlyarov, V.A. Volkov, A.A.Letichevsky Jr., and T.Weigert. Semantics of Message Sequence Charts, SDL Forum, 2005.

[26]A.A. Letichevsky, J.V. Kapitonova, V.P. Kotlyarov, A.A. Letichevsky Jr., N.S. Nikitchenko, V.A. Volkov, and T. Weigert, Insertion modeling in distributed system design, Проблеми програмування 2008, №4, 13-38, 118.

[27] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science* **96** (1992), 73–155.

[28] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[29] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[30] R. Milner, The polyadic π-calculus: a tutorial, Tech. Rep. ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK (1991).

[31] D. Park, Concurrency and automata on infinite sequences, in: *LNCS 104*, Springer-Verlag, 1981.

[32] M. Roggenbach and M. Majster-Cederbaum, Towards a unified view of bisimulation: a comparative study, *TCS* **238** (2000), 81–130.

[33] ITU-T. Z.120 Recommendation Z.120 (11/99): Languages for telecommunications applications – Message Sequence Charts (MSC), 1999.

[34] ITU-T. Z.100 Recommendation Z.100 – Specification and Description Language (SDL), 1999

[35] J. Rutten, Coalgebras and systems, *TCS* **249**.

[36] ITU. Recommendation Z.151 – User Requirements Notation (URN), – 2008.

[37] Object Management Group. Unified Modeling Language Specification, 2.0 – 2003.