А.А. Летичевский

Общая теория взаимодействия и когнитивные архитектуры

Введение

Общая теория информационных взаимодействий начинается с нейронных сетей Мак-Каллока Питса [MP43]. Теория нейронных сетей привела к появлению теории абстрактных автоматов, теории, которая позволяет изучать поведение и взаимодействие эволюционирующих систем независимо от их структуры. Первоначально теория автоматов развивалась как теория конечных автоматов и алгебра Клини-Глушкова [Kleene56, Glushkov60] служила основным средством описания их поведения. В дальнейшем теория автоматов концентрировалась на исследовании вопросов анализа и синтеза, изучении обобщений конечных автоматов и на вопросах сложности. Сети из автоматов исследовались в прикладных областях, связанных с проектированием электронных схем компьютеров. Взаимодействие в явной и общей форме появилось только в 70-х годах как теория взаимодействующих информационных процессов. Она включает CCS (исчисление взаимодействующих процессов) [Milner80, Milner89], π-исчисление Милнера [Milner91], CSP (взаимодействующие последовательные процессы) Хоара [Hoare85], ACP (алгебра взаимодействующих процессов) [BK84] и много других ответвлений этих базовых теорий. Достаточно полный обзор классической теории процессов представлен в справочнике по теории процессов [Handbook01], опубликованном в 2001 году. Одновременно начали развиваться модели параллельных вычислений под влиянием практических запросов параллельного программирования. Наиболее абстрактные модели – это сети Петри [Petri62], модели акторов (актеров) [Hewitt73], а также широко распространенные идеи объектно-ориентированного (параллельного) программирования. Они занимают промежуточное место между сетевыми моделями (нейронные и автоматные сети, схемы потоков данных) и моделями теории процессов.

Инсерционное моделирование сформировалось в 90-е годы как одно из направлений в общей теории взаимодействия. Первоначальное название – модель взаимодействия агентов и сред. Основные понятия инсерционного моделирования (среда, агенты, функция погружения) были введены в работах [LG96, LG98, LG99], 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 [Glushkov65, GluLet69] 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 [KapLet88], 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.

Модели, исследуемые в теории процессов, 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 [Rutten2000], подход, предложенный Хоаром в [Hoare99] к унификации теорий программирования, логика условного переписывания Месегера [Meseguer92] и др.

В последние годы инсерционное моделирование становится инструментом разработки прикладных систем верификации требований и спецификаций распределенных взаимодействующих систем [Baranov03, Kapitonova05, Letichevsky05-1, Letichevsky05-2]. Система VRS, разработанная в Украине по заказу фирмы Моторола с участием сотрудников Института Кибернетики им. В.М. Глушкова применяется для верификации требований и спецификаций в области телекоммуникационных систем, встроенных систем и систем реального времени. Новая система инсерционного моделирования IMS [Letichevsky11], которая разрабатывается в настоящее время в Институте кибернетики, должна расширить области применения инсерционного моделирования.

Когнитивные архитектуры – 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) [BICA]. В рамках этого направления было выполнено сравнение большого количества существующих когнитивных архитектур, выделены основные черты, которыми должны обладать такие архитектуры, ежегодно проводятся конференции, собирающие специалистов из различных областей, заинтересованных в продвижении в этом направлении.

Изучая современные системы, используемые для построения когнитивных архитектур [Samsonovich10], мы нашли много общего в этих системах и средствах разработки программных систем, основанных на формальных методах. Это обстоятельство стимулировало начало работы над созданием новой когнитивной архитектуры Икар (инсерционная когнитивная архитектура), основанной на инсерционном моделировании в системе IMS.

В лекции представлены основные принципы инсерционного моделирования и концепция когнитивной модели, которая создается на этих принципах. Основные понятия теории взаимодействия такие, как транзиционная система, трассовая и бисимуляционная эквивалентность, алгебра процессов и поведений, последовательная и параллельная композиция транзиционных систем предполагаются знакомыми.

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.
4. Среда, рассматриваемая как агент, может быть погружена в среду верхнего уровня.

5. Новые агенты могут погружаться в среду, перемещаясь 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.

Говоря об иерархии агентов и сред применительно к архитектуре интеллектуального агента, мы имеем в виду не только иерархию внешних сред, начинающуюся с той, в которую агент погружен, но и иерархию его внутренних сред и составляющих его частей, которые также рассматриваются как агенты, погруженные в свои среды. В этом плане наша точка зрения подобна неформальным построениям, которые рассматриваются Минским в его «Society of mind» [Minsky88]. Более специальные иерархии на формальном уровне рассматриваются в исчислении мобильных амбиентов Л. Карделли [Cardelli2000] и биоамбиентов [Cardelli2003].

Агенты. Переходя к математическим уточнениям, вберем в качестве уточнения понятия агента транзиционную систему – 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. Размеченная транзиционная система определяется своим множеством состояний и множеством размеченных переходов $s \xrightarrow{a} s'$. Переходы из состояния в состояние размечаются действиями, эквивалентность состояний – равенство поведений системы в этих состояниях. В случае бисимуляционной эквивалентности поведения – это размеченные деревья, в случае трассовой эквивалентности – это множества последовательностей действий (трасс), путей возможного эволюционирования системы. 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. Обычно транзиционную систему рассматривают как функционирующую в дискретном времени, непрерывное время может быть введено в действия. В качестве действий можно

рассматривать любые структуры данных. Например, в действие можно включить дискретный или непрерывный интервал времени, в течение которого оно выполняется. Можно также рассматривать в качестве действий числовые функции вещественного переменного, определенные на полуоткрытых интервалах. Таким образом, нейроны и нейронные сети, как с дискретным, так и с непрерывным временем, можно рассматривать как транзиционные системы.

Среда – это агент, который обладает функцией погружения. Более точно, среда – это набор $< E, C, A, \mathrm{Ins} >$, где E – это множество состояний среды, 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 (алгебра поведений рассматривается ниже). 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 $u+v$ (ассоциативная, коммутативная и идемпотентная операция), где a – действие, u и v – поведения. Кроме того, в алгебре поведений есть две терминальные константы – успешное завершение Δ и тупиковое поведение 0 (нейтральный элемент операции недетерминированного выбора). Всякое поведение можно представить в нормальной форме

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

где ε_u терминальная константа (если $I = \varnothing$, то сумма равна 0 и $u = \varepsilon_u$). 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. часть дерева, которая определяется конечным множеством конечных путей, которые начинаются в корне) называется *префиксом* поведения. Непрерывные функции от поведений характеризуются тем, что их значения зависят только от конечных префиксов их аргументов.

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 $(E_{\eta\vartheta})_{\eta \in \mathrm{H}, \vartheta \in \Theta}$ и алгебр поведений агентов $F(A_\vartheta)_{\vartheta \in \Theta}$ в качестве основных множеств. Тогда функция погружения среды $E_{\eta\vartheta}$ будет определена как $Ins_{\eta\vartheta} : E_{\eta\vartheta} \times F(A_\vartheta) \to E_{\eta\vartheta}$ есть функция погружения агентов типа $\vartheta \in \Theta$ в среду типа $\eta \in \mathrm{H}$. Теперь 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 (все состояния разложимы) 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. Пусть агент «хочет» выполнить действие b (одно из возможных). Тогда 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* (look-ahead).

Рассмотренных правил не достаточно для полного описания функции погружения. 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

$$e[u + v] = e[u] + e[v], (e + f)[u] = e[u] + f[u] \quad \text{corrected}$$

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: $e[u, v] = e[u \parallel v]$. Здесь ( )||( ) обозначает параллельную композицию поведений.

Рассмотренные классы функций погружения не исчерпывают функции погружения интересные с точки зрения моделирования когнитивных процессов. Наиболее важными являются функции погружения, вычисление которых представляет собой комбинацию развертывания рекурсивных определений и техники переписывания в расширенной алгебре поведений. Более подробно такие функции погружения рассмотрены в [Letichevsky05]

Важным свойством агентов в многоуровневых средах является свойство *мобильности* – возможности переходить из одной среды в другую. Это свойство может быть описано, например, правилами следующих видов:

$$\frac{u \xrightarrow{moveup\ E} u'}{E[F[u,v],w] \xrightarrow{moveup(F \to E)} E[F[v],u',w]} P(E,F,u,moveup(E))$$

переход из внутренней среды во внешнюю,

$$\frac{u \xrightarrow{movedn\ F} u'}{E[F[v],u,w] \xrightarrow{movedn(E \to F)} E[F[u',v],w]} P(E,F,u,movedn(E))$$

переход из внешней среды во внутреннюю,

$$\frac{u \xrightarrow{moveto\ F} u'}{E[F[u,v],G[w]] \xrightarrow{moveto(F \to G)} E[F[v],G[u',w]]} P(F,G,u,moveto(G))$$

переход из одной среды в другую на том же уровне. Во всех этих правилах *v* и *w* последовательности агентов. Условия применимости каждого из этих правил включают в

себя условие совместимости агента и среды, в которую он погружается. Это условие состоит в том, что множество действий агента должны быть включены в множество действий агентов, погружаемых в эту среду. Более точно, агент *u* с множеством действий A' совместим со средой $< E, C, A, \mathrm{Ins} >$, если $A' \subseteq A$.

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 [Letichevsky05-1] 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 из 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 ($\mathrm{check}\,\alpha$, где α – формула базового языка). Соответствующие правила имеют вид:

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

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

где e' есть результат применения присваивания к состоянию e. Разрешающее условие для присваивания может накладывать ограничение на однозначность значений для аргументов левой части (для функциональных атрибутов) 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 взаимодействующих систем (программных, технических, биологических) находятся в области темпоральной, динамической и некоторых других видов модальных логик. 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 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 локальных свойств 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 $\forall x(\alpha \rightarrow < P > \beta)$, где x – список (типизированных) параметров, α и β – формулы базового языка, P – 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 P может быть активировано 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. 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, которые рассматриваются в следующем разделе.

Сетевые среды. Агенты в сетевых средах имеют наборы входных и выходных каналов, по которым передаются и принимаются сигналы, представляемые атрибутами определенных типов. Будем называть такие агенты *сетевыми*. Действия сетевых агентов представляются парами *x*/*y*, где *x* – набор значений входных атрибутов, *y* – набор значений выходных атрибутов. Отношение переходов $s \xrightarrow{x/y} s'$ интерпретируется таким образом: если агент находится в состоянии *s* и набор его входных атрибутов принимает значение *x*, то набор его выходных атрибутов может принять значение *y* и агент может перейти в состояние s'. Очевидным образом сетевой агент может рассматриваться как недетерминированный автомат, а его атрибуты можно рассматривать как входные и выходные каналы.

Состояние сетевой среды (*сети*) определяет связи между значениями входных и выходных атрибутов сетевых агентов, погруженных в эту среду. Локальные свойства функции погружения для атрибутной сетевой среды могут выглядеть следующим образом:

$$\forall(x,y,u,v,r')($$
$$connect(m,r,x,y) \wedge (state(m)=u) \wedge u \xrightarrow{x/y} v \wedge newval(m,v,r,r')$$
$$\rightarrow < P(r,r') >$$
$$(r := r') \wedge (state(m)=v)$$
$$)$$

Здесь r – некоторый список атрибутов среды и агентов. Он включает те атрибуты, от которых зависит набор значений x входных атрибутов агента m и атрибуты, значения которых могут измениться в следующий момент времени. Предикат *connect* определяет возможные значения входных и выходных атрибутов агента m, а предикат *newval* определяет новые значения r' атрибутов r в следующий момент времени. Процесс $P(r,r')$ визуализирует некоторые атрибуты из списка r и их новые значения. Локальное описание определяет функционирование агента m в среде, в которую он погружен.

Предикаты *connect* и *newval* определяют структуру сетевой среды. Они могут рассматриваться как атрибуты функционального типа, и тогда сеть может менять свою структуру, скажем, при погружении в нее очередного агента. Правила соответствующих изменений также могут быть выражены в терминах локальных свойств.

Если во множестве атрибутов сетевой среды выделить входные и выходные атрибуты сети, то сеть можно будет рассматривать как сетевой агент и осуществлять его погружение в сетевую среду следующего уровня. При этом локальные свойства должны быть такими, которые допускают произвольные изменения входных атрибутов сети, а действия сети должны быть парами вида x/y, где x – набор значений входных атрибутов сети, а y – набор значений ее выходных атрибутов.

Наиболее типичными представителями класса сетевых сред являются *сети из автоматов*. В этом случае сетевые агенты – это автоматы, а их входные и выходные атрибуты – это их входные и выходные каналы, соответственно. Аналогично, входные и выходные атрибуты сети – это ее входные и выходные каналы. Для определения структуры сети используются промежуточные атрибуты, которые называются внутренними каналами, портами, узлами и т.д., а предикаты *connect* и *newval* определяются с помощью равенств. В частности, каждый внутренний канал отождествляется с выходным каналом одного из автоматов или входным каналом сети, а также с несколькими входными каналами автоматов и выходными каналами сети.

Приведенное выше локальное описание функционирования сетевого агента в сетевой среде соответствует случаю, когда в один момент дискретного времени меняет

свое состояние только один сетевой агент. Для описания более сложных переходов можно пользоваться правилами, которые получаются в виде композиции простых правил.

Другим важным классом систем, которые могут быть представлены как сетевые среды, являются биологические или искусственные *нейронные сети*. В этом случае сетевой агент нижнего уровня – это нейрон. Входные атрибуты модели биологического нейрона – это дендриты, а единственный выходной атрибут – аксон. Сетевые атрибуты, используемые для организации связей между нейронами – это синапсы. Искусственный нейрон – это просто устроенный автомат, который суммирует значения своих входных каналов и вырабатывает на выходе значение в соответствии со своей передаточной функцией. Область значений входных и выходных атрибутов нейрона это либо множество вещественных чисел, либо конечное множество числовых сигналов. Нейроны и нейронные сети можно рассматривать как агенты, функционирующие в дискретном или непрерывном времени. Для перехода от непрерывного к дискретному времени удобно значения входных и выходных атрибутов представлять как функции, определенные на вещественных интервалах. Веса синаптических связей составляют атрибуты среды, которые используются для реализации алгоритмов обучения, самообучения, распознавания и т.п. Все эти алгоритмы реализуются средой.

Абстракции.

При моделировании больших систем, таких как телекоммуникационные системы, сети типа Интернет, многопроцессорные системы с большим числом компонент, мозг человека, 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 [Letichevsky05-1] были введены понятия абстракции и конкретизации атрибутных транзиционных систем. Эти понятия также важны для построения когнитивных архитектур. Например, в эпизодической памяти, которая носит динамический характер, последовательности событий запоминаются в абстрактном виде. Абстракции дают возможность находить общее в различных процессах, воспринимаемых когнитивным агентом.

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.

Значение этих утверждений в технических приложениях 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 [Letichevsky05-1] была построена общая семантика систем базовых протоколов, сопоставляющая каждой системе P базовых протоколов некоторую фиксированную транзиционную систему $\mathbf{S}(P)$, 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 $\mathbf{S}(P)$ 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 [Letichevsky05-1], 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 $\mathbf{S}(P)$

является абстракцией любой системы из класса $C(P)$. Эта абстракция будет прямой, если выбран универсальный метод и обратной, если выбран экзистенциальный метод.

В [Letichevsky08] 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 [ITU99] с инсерционной семантикой [Letichevsky02]. Система также допускает использование языка SDL и соответствующих диаграммных представлений языка UML.

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.

Для когнитивных архитектур применение методов исследования моделей имеет двоякое значение. С одной стороны, они могут быть полезными для разработчика интеллектуальных агентов в среде когнитивной архитектуры. С другой стороны, некоторые из этих методов должны быть реализованы внутри интеллектуального агента, который будет их использовать для исследования создаваемых им моделей внешней среды.

Инсерционные машины

Для реализации инсерционных моделей некоторого класса используется интерпретатор моделей, который мы называем *инсерционной машиной* (рис. 1). *Простая 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 модели.
2. *Анфолдер поведений агентов*. Текущее состояние модели представляется в виде алгебраического выражения в расширенной алгебре поведений. Входной язык инсерционной машины допускает использование рекурсивных определений для описания поведений агентов и структур данных для описания поведения или состояний среды. Анфолдер поведений использует эти описания для того, что бы получить разложение состояния модели 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 $\varepsilon = \Delta$ или $I = \varnothing$.

Простая инсерционная машина воспроизводит поведение модели, погруженной в нее, ничего не меняя в этой модели, а лишь детерминируя поведение в случае недетерминированного выбора. Более сложные, когнитивные машины будут рассмотрены позже.

Различаются два типа инсерционных машин: *машины реального времени* или *интерактивные машины* и *аналитические инсерционные машины*.

Машины реального времени работают как агенты, погруженные во внешнюю реальную или виртуальную среду, взаимодействуя с ней 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 драйверы.

Машины реального времени. Интерактивный драйвер после нормализации состояния должен выбрать в точности один переход и выполнить его путем передачи

действия своей модели 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. С другой стороны, если отождествить модель агента, погруженного в инсерционную машину с самим агентом, то интерактивный драйвер можно рассматривать как среду для этого агента. Если интерактивный драйвер используется как среда для последовательности агентов, их можно объединить с помощью некоторой композиции, например параллельной, и получить один агент, поведение которого будет приводиться к нормальной форме с использованием определения этой композиции.

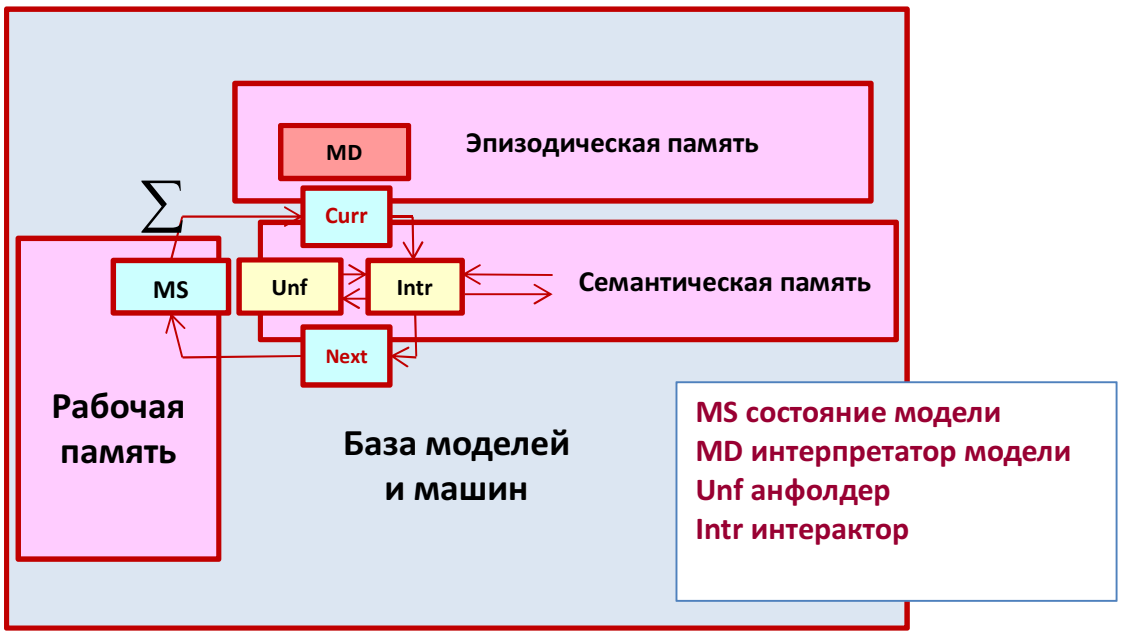

Рис.1 Архитектура когнитивной машины

Естественно предположить, что внешняя среда не знает структуры машины и погруженной в нее модели. Поэтому, взаимодействие машины и среды происходит на уровне одношагового префиксного погружения:

$$\frac{E \xrightarrow{x} E', M[u] \xrightarrow{y} M'[u']}{E[M[u]] \xrightarrow{c} E'[M'[u']]} P(E,x,y,c)$$

Здесь E – среда, M – инсерционная машина P – разрешающий предикат.

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. Простому интерактивному драйверу доступна вся модель, и он может развертывать дерево поведения этой модели неограниченным образом, прогнозируя будущее. Иными словами, функция погружения для интерактивного драйвера – это прогнозирующая функция:

$$\frac{M \xrightarrow{z} M', u \xrightarrow{a} u'}{M[u] \xrightarrow{b} M'[u']} Q(M, u, z, a, b)$$

Разрешающий предикат Q определяет способ выбора направления движения в случае недетерминированного поведения агента $u = \sum_{i \in I} a_i . u_i + \varepsilon$. Простейший механизм выбора направления движения для интерактивного драйвера – это использование приоритетов. Предположим, что на множестве агентов, погруженных в среду, задана непрерывная функция приоритета $pr : F(A) \times A \times F(A) \to U$, где U – некоторое частично упорядоченное множество. Тогда, после приведения к нормальной форме, интерактивный драйвер недетерминированным образом выбирает индекс $i \in I$ такой, что $pr(u, a_i, u_i)$ принимает одно из максимальных значений. В случае нескольких максимальных значений, должны работать дополнительные механизмы детерминизации, включая использование вероятностных критериев.

В более сложных случаях драйвер инсерционной машины может накапливать опыт и менять функцию приоритета в соответствии с некоторыми критериями успеха своего функционирования.

Аналитические машины. 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. Результатом поиска будет одна или несколько трасс, подтверждающих достижимость целевых состояний.

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.

Система инсерционного моделирования IMS (Insertion Modeling System), разработанная в Институте кибернетики им. В.М. Глушкова представляет собой среду для разработки инсерционных машин различного типа, включая когнитивные архитектуры. Она содержит в своем составе ряд прототипов инсерционных машин для моделей 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 [Letichevsky12], описана инсерционная машина для доказательного программирования. Она ориентирована на верификацию императивных аннотированных программ, использующих различные языки программирования.

Когнитивная архитектура Икар

Как и всякая сложная система, архитектура системы Икар имеет многослойную структуру, представленную на рис. 2. В основе пирамиды, представляющей архитектуру Икар, лежит язык программирования. В настоящее время это C++ в операционной системе Windows или Unix, но, поскольку верхние слои остаются неизменными, переход на другие платформы, в том числе на платформы с реальным параллелизмом, технически не представляет большой проблемы.

Верхний уровень системы Икар представляет собой систему инсерционного моделирования IMS, используемую как инструмент для разработки инсерционных машин, и одной из этих машин является *интеллектуальный агент* Икар. Мы используем одно и то же имя для системы (когнитивной архитектуры) и интеллектуального агента, который создается на базе этой архитектуры.

Система IMS реализована на базе системы алгебраического программирования APS [aps], которую можно охарактеризовать следующими ключевыми словами: системы

переписывающих правил и стратегии переписывания. Язык APLAN – это входной язык системы APS, APLANC – это библиотека базовых функций системы APS в языке C++.

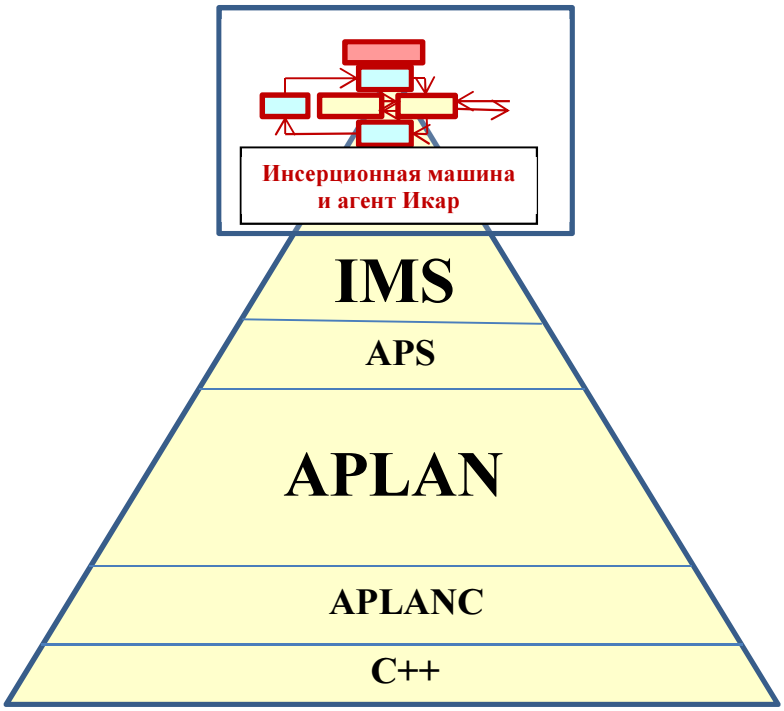

Рис.2 Система Икар

Технической основой реализации агента Икар является система инсерционных машин, реализующих поведение различных подсистем агента Икар и организованных как многоуровневая среда. Для представления моделей, имплементируемых инсерционными машинами системы Икар, используются системы рекурсивных определений в соответствующих алгебрах поведений для подсистем нижнего уровня, локальные описания, атрибутные и сетевые среды.

Интеллектуальный агент Икар – это *когнитивная* инсерционная машина реального времени, 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.

Когнитивная инсерционная машина отличается от простой тем, что она может трансформировать погруженную в нее модель, уточняя ее на основе полученного опыта и усовершенствуя ее в соответствии со своими критериями, которые также могут меняться со временем. Начальная модель, погруженная в среду когнитивной машины, – это модель

внешней среды, в которую погружена сама машина. Эта модель включает в себя и модель самой когнитивной машины. В случае биологических систем эта модель передается генетически, в случае компьютерной модели, она содержит начальную базу знаний, которую создает разработчик.

Когнитивный драйвер машины Икар – его внутренняя среда устроен как многоуровневая среда, наполненная когнитивными машинами и моделями, погруженными в эти машины. Структура интеллектуального агента Икар представлена на рис. 3.

Уровень Эго (Self) – это верхний уровень внутренней среды агента Икар. Он обеспечивает управление всеми агентами, погруженными в нее, и через них контроль функционирования всех подсистем нижних уровней. На этом уровне также находится модельный драйвер *основной модели* внешнего мира с погруженной в нее моделью Икара. Икар непрерывно наблюдает за всеми изменениями, происходящими во внешней среде, анализируя поступающую образную и языковую информацию, и формирует реакцию в виде своих действий (передвижение в пространстве, генерация языковой или образной информации) с помощью основной модели. Различение своей внутренней контролируемой и управляемой среды и населяющих ее агентов с одной стороны, и внешней среды с другой стороны, и есть осознание агентом Икар самого себя. На верхнем уровне реализуются также модели эмоций и чувств, в том объеме, в котором они необходимы для обеспечения эффективного и плодотворного взаимодействия Икара с людьми.

На верхнем уровне находятся также механизмы оценки успеха своей деятельности. Основные критерии успеха закладываются разработчиками Икара, но они зависят от критериев успешности функционирования компонент нижних уровней, которые относятся к различным видам деятельности и не могут быть выбраны заранее. Эти критерии должны формироваться в процессе накопления опыта, обучения и самообучения с помощью когнитивного драйвера.

Сфера активности. Определяет взаимодействие Икара с внешней средой. Составляет часть основной модели. Сюда входят такие традиционные для автономных агентов компоненты, как выбор цели и подцелей, планирование и реализация плана, взаимодействие с другими агентами, погруженными во внешнюю среду. На входы Икара (его входные атрибуты) непрерывно поступает образная информация о нахождении агента в трехмерном (виртуальном или реальном) пространстве и расположении в нем предметов

и других агентов, а также символьная (языковая) информация, поступающая от пользователей и других агентов из внешней среды. Выходы Икара (выходные атрибуты) воспринимаются внешней средой и приводят к изменению ее состояния в соответствии с функцией погружения (перемещение, взаимодействие с предметами и агентами, передача и восприятие внешней средой символьной информации).

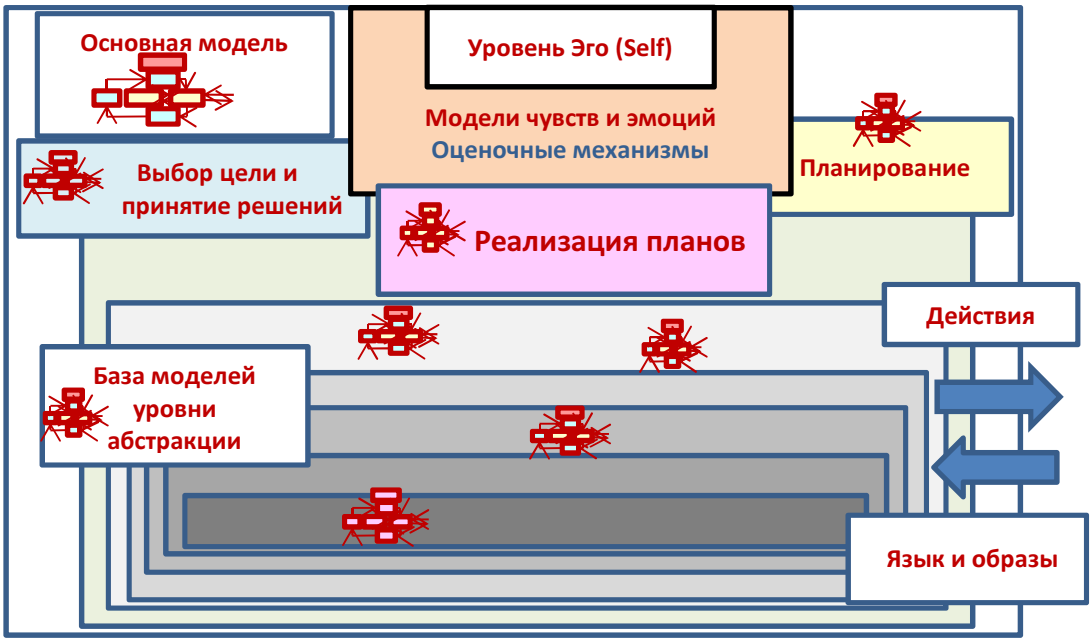

Рис.3 Агент Икар

Языковое (символьное) общение. Икар должен общаться с пользователями на естественном языке. В символьном виде Икар получает информацию о внешней среде, ее состоянии и событиях происходящих в ней в дополнение к образной информации. Восприятие и понимание этой информации формирует изменения в основной модели. Основная модель помимо поведенческих аспектов включает в себя память о прошлом, прогнозы возможного будущего, равно как и логические модели внешней и внутренней среды Икара. С другой стороны, высказывания Икара, которые он генерирует, общаясь со своими собеседниками, относятся к описанию его внутреннего состояния, которое прямо или косвенно связано с состоянием его основной модели. Но, состояния всех моделей имеют описания в языке инсерционного моделирования. Таким образом, язык инсерционного моделирования выступает в качестве семантической основы языка

общения Икара, а сам язык общения можно также использовать как метаязык для языка инсерционного моделирования.

Память. Основные виды памяти, которыми пользуется Икар – это рабочая память, эпизодическая память, семантическая память, а также база моделей различного уровня абстракции и инсерционных машин. Все эти виды памяти связаны через основную модель, как показано на рис.1. Состояние модели располагается в рабочей памяти. Обычно – это формула базового языка. Здесь же размещается информация из семантической памяти необходимая для развертывания рекурсивных определений и вычисления функции погружения. Она определяется понятиями (функциональные и предикатные символы) используемыми в формуле, а также следствиями, которые определяют релевантные локальные описания необходимые для вычисления функции погружения. Результат работы машины – символьная трасса передается в эпизодическую память. Здесь представлен прошлый опыт функционирования основной модели. Множество трасс, порожденных основной моделью в процессе взаимодействия с внешней средой, представляются на различных уровнях абстракции и объединяются в структуры данных, которые снабжаются дополнительной информацией, характеризующей вероятностные оценки типа Байесовских сетей, а также информацию об успехах и неудачах. Вся эта информация используется для прогнозирования поведения внешней среды и создания предпочтений при выборе продолжений основной моделью. Здесь могут быть использованы идеи Джеффа Хокинса по структуре иерархической временно́й памяти [Hawkins05], предложенной им в качестве модели неокортекса.

Основную часть памяти Икара составляет база инсерционных машин и моделей. Здесь, в частности находятся машины для решения различных классов задач и соответствующие им модели. При необходимости, они могут активизироваться основной моделью и погружаться в ее среду для участия во взаимодействии с внешней средой. Другой пример – машины для работы с языковыми конструкциями, которые воспринимают и распознают языковую информацию, порождают ответы на вопросы, дают объяснения в духе экспертных систем на естественном языке. Здесь могут быть машины для участия в играх, выполнения работ в пространственных условиях и т.п.

Обучение. Основная цель Икара состоит в совершенствовании своей основной модели в соответствии с критериями, определенными его создателем. Основную роль в этом процессе играют методы обучения и самообучения. Помимо известных методов, адаптированных к инсерционному моделированию, предполагается обучение путем

объяснения различных приемов и методов решения задач, правил и рекомендаций поведения в различных ситуациях. Такого рода уроки должны обеспечиваться соответствующими упражнениями и примерами. Выполняя эти упражнения, Икар должен вырабатывать соответствующие абстракции для того, что бы расширить сферу применимости соответствующих методов для ситуаций, похожих, но отличных от контекста, сопровождающего уроки.

Икар должен также владеть методами изменения основной модели и специализированных машин и моделей на основе накопленного опыта, конструировать новые полезные модели и машины.

Заключение

В лекции были рассмотрены основы инсерционного моделирования, новой парадигмы в общей теории взаимодействия. В настоящее время основной областью применения инсерционного моделирования является верификация технических систем, однако потенциальные возможности этого направления далеко не исчерпываются этой областью.

Парадигма инсерционного моделирования в настоящее время используется для построения новой когнитивной архитектуры Икар. Разработка Икара пока еще находится на уровне выработки требований, часть из которых изложена в лекции. Основными строительными модулями Икара являются инсерционные машины и модели. Конструирование и совершенствование этих моделей и машин составляет основу разумной деятельности интеллектуального агента.

Литература

[aps] www.apsystem.org.ua

[Baranov03] 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.

[BK 84] J. A. Bergstra and J. W. Klop, Process algebra for synchronous communications, *Information and Control* **60** (1/3) (1984), 109–137.

[Glushkov60] В.М.Глушков, Об одном алгоритме синтеза абстрактных автоматов, Украинский математический журнал, 12, 2, 147-156, 1960.

[Glushkov65] В.М.Глушков, теория автоматов и вопросы проектирования структур цифровых машин, Кибернетика 1,1965, 3-11, 104.

[GluLet69] V. M. Glushkov and A. A. Letichevsky, Theory of algorithms and discrete processors, in: *Advances in Information Systems Science* (J. T. Tou, ed.), vol. 1, Plenum Press, 1969, 1-58.

[Handbook 01] J. A. Bergstra, A. Ponce, and S. A. Smolka, eds., *Handbook of Process Algebra*, North-Holland, 2001, 1342.

[Hewitt 73] Carl Hewitt, Peter Bishop, and Richard Steiger, A Universal Modular Actor Formalism for Artificial Intelligence, IJCA, 1973.

[Hawkins05] Jeff Hawkins with Sandra Blakeslee, On intelligence, Times Books, Henry Holt and Co, 2005.

[Hoare 85] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[Hoare99]C. A. R. Hoare and He Jifeng, Unifying Theories of Programming, Prentice Hall, 1999

[ITU99] ITU-T. Z.120 Recommendation Z.120 (11/99): Languages for telecommunications applications – Message Sequence Charts (MSC), 1999.

[Kleene56] Kleene, Stephen C. Representation of Events in Nerve Nets and Finite Automata, In Shannon, Claude E.; McCarthy, John. Automata Studies. Princeton University Press. pp. 3–42, (1956).

[KapLet88] Ю.В.Капитонова, А.А.Летичевский, Математическая теория проектирования вычислительных систем, Москва, Наука, 1988, 295с.

[Kapitonova05] 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.

[LG96]. 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.

[LG98]. A. Letichevsky and D. Gilbert. A general theory of action languages. Кибернетика и Системный Анализ, 1, 1998, 16-36, 192.

[LG99] 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.

[Letichevsky05] 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.

[Letichevsky02] 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.

[Letichevsky05-1] 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.

[Letichevsky05-2] 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.

[Letichevsky08] 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.

[Letichevsky11] A. Letichevsky, O. Letychevskyi, V. Peschanenko, Insertion Modeling System, PSI 2011, Lecture Notes in Computer Science, Vol.7162, pp. 262-274, Springer (2011)

[Letichevsky12]А.А.Летичевский (мл), М.К.Мороховец, В.В.Песчаненко, Система доказательного программирования, УСиМ 6, 2012.

[Meseguer92] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science* **96** (1992), 73–155.

[MP43] McCulloch W.S., Pitts W., A logical calculus of the ideas immanent in nervous activity, Bull. of math. biophy., 5, 115-133,1943.

[Milner 80] R. Milner, A Calculus of Communicating Systems, vol. 92 of Lecture Notes in Computer Science, Springer-Verlag (1980).

[Milner 80] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[Milner 89] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[Milner 91] 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).

[Petri 62] Kommunikation mit Automaten. Petri, C.A., Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[Rutten2000] J. Rutten, Universal coalgebra: a theory of systems, *TCS* 249 (2000), 3-80 .

[Samsonovich10] A.V. Samsonovich, Toward a unified Catalog of Implemented Cognitive Architectures, in A.V. Samsonovich et al (Eds), Biologically Inspired Cognitive Architectures 2010, IOS Press, 2010, 195-244.

[BICA] http://bicasociety.org