Модел. и анализ информ. систем. Т. **19**, № **6** (2012) 45–56

УДК 519.7

Как разработать простое средство верификации систем реального времени[1]

Волканов Д.Ю.*, Захаров В.А.*, Зорин Д.А.*, Коннов И.В.**, Подымов В.В.*

* *Московский государственный университет им. М.В. Ломоносова*
** *Technische Universität Wien*

e-mail: valdus@yandex.ru

получена 22 июля 2012

Ключевые слова: верификация, система реального времени, диаграмма состояний, иерархический автомат, временной автомат

Исследуется задача верификации систем реального времени (СРВ). Для описания СРВ удобно использовать диаграммы состояний UML с семантикой, определяемой иерархическими автоматами. Для верификации СРВ часто применяется средство UPPAAL, разработанное для проверки формул логики TCTL на сети временных автоматов. Основным результатом данной статьи является алгоритм трансляции иерархических автоматов в сеть временных автоматов и обоснование его корректности.

1. Введение

Верификация моделей программ — это подход к проверке правильности поведения программ, возникший более тридцати лет назад. Сейчас существует большое разнообразие как формальных языков спецификации поведения программ, так и методов и средств верификации моделей программ. В данной статье изложен наш опыт создания средства верификации систем реального времени (СРВ) на основе существующих средств верификации и описания программ.

Ключевой стадией разработки СРВ является начальная — на ней строится высокоуровневое описание системы, ее модель. В качестве языка моделирования нами были выбраны диаграммы состояний UML, поддерживающие вложенность компонентов СРВ, параллелизм и временные ограничения. Для формализации семантики диаграмм используется модель иерархических временных автоматов [1, 2].

Ранее в статьях [3, 4, 5, 6, 7] были описаны системы верификации UML-диаграмм, основанные на средствах SMV и SPIN. Для верификации СРВ наиболее популярным является средство UPPAAL [8, 9], разработанное для проверки формул логики TCTL (Timed Computational Tree Logic) на сетях (плоских) временных автоматов. Таким образом, для построения системы верификации СРВ достаточно создать

[1]Работа выполнялась по Госконтракту 14.740.11.0399 с Минобрнауки РФ, в рамках федеральной целевой программы "Научные и научно-педагогические кадры инновационной России".

средство трансляции иерархических временных автоматов в сети плоских временных автоматов. Подобные алгоритмы описаны в работах [10, 11], однако они имеют существенные недостатки в описании, такие как недостаточная подробность, ошибки и отсутствие приемлемого обоснования корректности. В настоящей работе нами предложен оригинальный алгоритм трансляции и обоснована его корректность. В отличие от алгоритма в работе [10], входная модель для нашего алгоритма более проста и поддерживает возможность широковещательного взаимодействия.

Структура статьи такова. В разделе 2 описана модификация модели иерархических временных автоматов, учитывающая возможность широковещательного взаимодействия компонентов СРВ, в разделе 3 — модель сети временных автоматов. В разделе 4 приводится описание алгоритма трансляции, в разделе 5 обосновывается его корректность.

2. Иерархические временные автоматы

Модель иерархических временных автоматов с синтаксисом и семантикой, тесно связанными с диаграммами состояний UML, впервые была предложена в работах [1, 10] и может быть описана следующим образом.

Иерархический временной автомат включает в себя, в числе прочего, множества переменных (Var) ограниченного целочисленного типа и таймеров ($Clock$), принимающих действительные значения. Над этими множествами строятся ограничения данных и таймеров. Пусть E_1, E_2 — арифметические выражения над множеством Var, $x \in Var$, $t_1, t_2 \in Clock$ и $\bowtie \in \{<, \leq, =, \leq, >\}$. *Ограничения данных* имеют вид $E_1 \bowtie E_2$, *ограничения таймеров* — $t_1 \bowtie E_1$ и $t_1 - t_2 \bowtie E_1$, *элементарные присваивания* — $x := E_1$ и $t_1 := 0$. Если $\bowtie \in \{<, \leq\}$, то ограничение таймеров описанного вида будем называть *ограничением инвариантов*. Записью $Guard$ будем обозначать множество всех *предусловий* — булевых выражений над ограничениями данных и таймеров. Записью Inv будем обозначать множество всех *инвариантов*. Каждый инвариант представляет собой конъюнкцию ограничений инвариантов или константу $true$. Записью $Reset$ будем обозначать семейство всех *присваиваний* — конечных множеств элементарных присваиваний с попарно различными левыми частями. Также для заданного множества $Chan$ каналов связи записью $Sync$ будем обозначать множество *сигналов* вида $!c$ (широковещательная посылка), $?c$ (широковещательный прием) и $none$ (отсутствие посылки и приема), где $c \in Chan$.

Иерархический временной автомат (далее просто автомат) — это система $H = (Var, Clock, Chan, L, L_0, \eta, Type, I, T)$, где

- Var, $Clock$, $Chan$ — множества *переменных*, *таймеров* и *каналов* соответственно;
- L — множество *состояний*, содержащее множество *инициальных* состояний L_0;
- $\eta : L \to 2^L$ — функция *вложенности* состояний;
- $Type : L \to \{BASIC, XOR, AND, ENTRY, EXIT\}$ — функция *типов* состояний;
- $I : L \to Inv$ — разметка состояний *инвариантами*;
- $T \subseteq L \times Guard \times Sync \times Reset \times L$ — множество *переходов*.

Для перехода (l_1, g, s, a, l_2) будем использовать запись $l_1 \xrightarrow{g,s,a} l_2$. Если какая-либо из компонент g, s, a отсутствует (равна $true$, $none$, $\emptyset$), при записи она будет

опускаться. Состояния типов AND и XOR будем называть метасостояниями, типа $BASIC$ — простыми, типа $ENTRY$ — входами, типа $EXIT$ — выходами, входы и выходы — псевдосостояниями, метасостояния и простые состояния — существенными. Записью $\eta^*(l)$, где $l \in L$, будем обозначать множество состояний, включающее в себя $l, \eta(l), \eta(\eta(l)), \ldots$. Записью $\eta^{-1}(l)$ будем обозначать такое состояние l', что $l \in \eta(l')$ (далее на автомат налагается условие, согласно которому такое состояние определяется единственным образом либо не существует). Состояния множества $\eta^*(l)$ далее будем называть внутренними для l, множества $\eta(l)$ — вложенными в l.

Для корректного функционирования на автомат налагаются следующие ограничения. Функция η вместе с множеством L определяет корневое дерево, множество листьев которого совпадает с множеством простых состояний и псевдосостояний. Корень этого дерева будем записывать как $root$. Поддерево этого дерева, начинающееся в вершине l, будем обозначать записью $\mathcal{T}^\eta(l)$ и называть *деревом вложенности* для состояния l. В AND-состояние могут быть вложены только метасостояния. В XOR-состоянии ровно один вход и ровно один выход. Из каждого входа исходит ровно один переход и не входит ни одного перехода; из каждого выхода не исходит ни одного перехода. Множество L_0 содержит только существенные состояния и обязательно содержит состояние $root$. Если AND-состояние входит в L_0, то все вложенные в него состояния входят в L_0. Если XOR-состояние входит в L_0, то ровно одно вложенное в него состояние входит в L_0. Множество T состоит только из переходов, соединяющих состояния одного уровня (т.е. вложенные в одно и то же метасостояние). Переходы, исходящие из входов и входящие в выходы, помечены предусловием $true$, синхронизацией $none$ и присваиванием $\emptyset$. Если переход помечен синхронизацией $?c$, $c \in Chan$, то он также помечен присваиванием $\emptyset$.

Ограничения на структуру автомата строже предложенных в работе [10] по двум причинам. Во-первых, для простоты описания и обоснования корректности; так, обработку каналов связи типа точка-точка (handshake), срочных (urgent) переходов и возможности вложения множества входов и выходов в одно метасостояние можно посмотреть в [10]. Во-вторых, ограничения на выражения, описанные в [10], не гарантируют корректности алгоритма трансляции; так, например, если допустить произвольные присваивания на переходах, помеченных синхронизацией $?c$, трансляция перестает быть корректной по причине того, что в автомате оба присваивания совершаются одновременно, тогда как в транслируемой сети возможно вычисление, в котором одно из присваиваний уже выполнено, тогда как другое — нет.

Опишем семантику автомата. Подробное ее описание, за исключением переходов, использующих широковещательные каналы, можно найти в работе [10].

Конфигурацией автомата является тройка (ρ, μ, ν), где $\rho \subseteq L$ — множество активных состояний автомата, в которое могут входить только существенные состояния, $\mu : Var \to \mathbb{Z}$ — оценка переменных, $\nu : Clock \to \mathbb{R}^+$ — оценка таймеров. Начальная конфигурация состоит из множества L_0 и оценок, ставящих в соответствие всем переменным и таймерам ноли.

Определим понятие *входного дерева* $\mathcal{ET}(l)$ состояния l автомата. Корнем дерева $\mathcal{ET}(l)$ является состояние l. Если AND-состояние v входит в дерево $\mathcal{ET}(l)$, то все вложенные в v состояния входят в $\mathcal{ET}(l)$ и отстоят от v на одну дугу. Если XOR-

состояние v входит в дерево $\mathcal{ET}(l)$, то на одну дугу от него отстоит состояние, достижимое по дуге автомата из вложенного в v входа.

Множество вершин дерева $\mathcal{T}$ далее будет записываться как $\rho_{\mathcal{T}}$. Обозначим записями μ^a, ν^a оценки переменных и таймеров, получаемые из оценок μ, ν заменой значений переменных из левых частей элементарных присваиваний множества a на значения правых частей при оценках μ, ν. Истинность формулы f при оценках μ, ν будем обозначать записью $\mu, \nu \models f$. Для заданной конфигурации (ρ, μ, ν) совокупность переходов $l'_i \xrightarrow{g_i, s_i, a_i} l''_i$, $1 \le i \le k$ будем считать *активной*, если выполнены следующие условия:

• $l'_i \in \rho$, $1 \le i \le k$;

• ни для какой пары состояний l'_i, l'_j, где $1 \le i, j \le k$ и $i \ne j$, не выполнено соотношение $l'_i \in \eta^*(l'_j)$;

• $\mu, \nu \models g_i$, $1 \le i \le k$;

• либо $s_1 = none$ и $k = 1$, либо $s_1 = !c$, $s_2 = \ldots = s_k = ?c$ и к данной совокупности нельзя добавить ни одного ребра с сохранением предыдущих условий.

• совокупность присваиваний $a = \bigcup_{i=1}^{k} a_i$ также является присваиванием;

• $\mu^a, \nu^a \models inv$ для всех инвариантов inv состояний из множества $\rho' = \rho \setminus \{\nu^*(l'_i) \mid 1 \le i \le k\} \cup \bigcup_{i=1}^{k} \rho_{\mathcal{ET}(l''_i)}$;

Пусть $t_i = l'_i \xrightarrow{g_i, s_i, a_i} l''_i$, $1 \le i \le k$ — совокупность переходов, активная для текущей конфигурации $c = (\rho, \mu, \nu)$, множества a и ρ' определены согласно последним двум пунктам определения активности и $c' = (\rho', \mu^a, \nu^a)$. Тогда смена конфигурации происходит по одному из следующих правил.

Обычный переход. Если $k = 1$ и $s_1 = none$, то c' получается из c в результате выполнения перехода t_1.

Переход с широковещательным сигналом. Если $s_1 = !c$, то c' получается из c в результате выполнения совокупности переходов t_i, $1 \le i \le k$.

Продвижение времени. Если для инвариантов inv всех состояний множества ρ выполняются соотношения $\mu, \nu^{+d} \models inv$, где $\nu^{+d}(t) = \nu(t) + d$, то конфигурация (ρ, μ, ν^{+d}) получается из конфигурации (ρ, μ, ν) в результате продвижения времени на константу d.

Трассой автомата является максимальная последовательность конфигураций, в которой первой конфигурацией является начальная, а каждая следующая конфигурация получается из предыдущей согласно одному из описанных правил.

3. Сеть временных автоматов

Сеть временных автоматов [12, 13] является математической моделью, реализация которой используется в средстве верификации UPPAAL [8, 9].

Сетью временных автоматов (далее просто *сетью*) будем называть систему $N = (Var, Clock, Chan, PPChan, \mathcal{A})$, где

• множества Var, $Clock$, $Chan$ таковы же, как и в иерархическом автомате;

• $PPChan$ — множество *каналов типа точка-точка* (Peer-to-Peer Channels),

• $\mathcal{A} = (A_1, A_2, \ldots, A_k)$ — вектор временных автоматов.

Обозначим записью $TASync$ множество сигналов вида $!c$, $?c$ и $none$, $c \in Chan \cup PPChan$. Каждый временной автомат (далее, для краткости, *процесс*) имеет вид $A_i = (L_i, l_i^0, Type_i, I_i, T_i)$, где

- L_i — множество *состояний*, включающее *инициальное* состояние l_i^0;
- $Type_i : L_i \to \{o, c\}$ — *типизация* состояний (обычные, срочные);
- $I_i : L_i \to Inv$ — разметка состояний *инвариантами*;
- $T_i \subseteq L_i \times Guard \times TASync \times Reset \times L_i$ — множество *переходов*.

Сеть представляет собой параллельную композицию процессов с двумя видами синхронизации: широковещательным и типа точка-точка.

Конфигурацией сети является тройка $(\vec{l}, \mu, \nu)$, где $\vec{l} = (l_1, l_2, \ldots, l_k)$ — вектор активных состояний сети, $l_i \in L_i$, $\mu : Var \to \mathbb{Z}$ — оценка переменных, $\nu : Clock \to \mathbb{R}^+$ — оценка таймеров. Начальная конфигурация сети состоит из вектора инициальных состояний процессов и оценок, обнуляющей значения всех переменных и таймеров.

Запись $(l_1, \ldots, l_k)[l_i \leftarrow l_i']$ будет обозначать вектор $(l_1, \ldots, l_{i-1}, l_i', l_{i+1}, \ldots, l_k)$. Для произвольной конфигурации $(\vec{l}, \mu, \nu)$, $\vec{l} = (l_1, \ldots, l_k)$ смена конфигураций происходит по одному из следующих правил.

Обычный переход. Если в сети есть переход $t = l_i \xrightarrow{g,a} l_i'$, где $\mu, \nu \models g$, $\mu^a, \nu^a \models inv$ для инвариантов inv всех состояний вектора $\vec{l'} = \vec{l}[l_i \leftarrow l_i']$ и либо состояние l_i имеет тип c, либо остальные активные состояния имеют тип o, то в результате выполнения перехода t получается конфигурация $(\vec{l'}, \mu^a, \nu^a)$.

Переход с сигналом точка-точка. Если в сети содержатся два перехода $t_1 = l_i \xrightarrow{g_1,!c,a_1} l_i'$, $t_2 = l_j \xrightarrow{g_2,?c,a_2} l_j'$, для которых $c \in PPChan$, $i \neq j$, $\mu, \nu \models g_1 \wedge g_2$, $a = a_1 \cup a_2$ — присваивание, $\mu^a, \nu^a \models inv$ верно для инвариантов inv всех состояний вектора $\vec{l'} = \vec{l}[l_i \leftarrow l_i'][l_j \leftarrow l_j']$, и при этом либо одно из состояний l_i, l_j имеет тип c, либо остальные активные состояния имеют тип o, то в результате выполнения пары переходов t_1, t_2 получается конфигурация $(\vec{l'}, \mu^a, \nu^a)$.

Переход с широковещательным сигналом. Если в сети содержатся такие переходы $t = l_i \xrightarrow{g,!c,a} l_i'$, $t_m = l_{j_m} \xrightarrow{g_m,?c,a_m} l_{j_m}'$, для которых $c \in Chan$, $1 \leq m \leq p$, числа $i, j_1, \ldots, j_p$ попарно различны, $\hat{a} = a \cup a_1 \cup \ldots \cup a_p$ — присваивание, $\mu, \nu \models g \wedge g_1 \wedge \ldots \wedge g_p$, нельзя добавить других переходов сети с сохранением перечисленных выше условий, $\mu^{\hat{a}}, \nu^{\hat{a}} \models inv$ для инвариантов inv всех состояний вектора $\vec{l'} = \vec{l}[l_i \leftarrow l_i'][l_{j_1} \leftarrow l_{j_1}'] \ldots [l_{j_p} \leftarrow l_{j_p}']$, и при этом либо среди состояний $l_1, l_{j_1}, \ldots, l_{j_p}$ есть состояния типа c, либо остальные активные состояния имеют тип o, то в результате выполнения переходов $t, t_1, \ldots, t_p$ получается конфигурация $(\vec{l'}, \mu^{\hat{a}}, \nu^{\hat{a}})$.

Продвижение времени. Если все активные состояния имеют тип o и $\mu, \nu^{+d} \models inv$ для всех инвариантов inv состояний вектора $\vec{l}$, то в результате продвижения времени на константу d получается конфигурация $(\vec{l}, \mu, \nu^{+d})$.

Трассой сети является максимальная последовательность конфигураций, в которой первой конфигурацией является начальная, а каждая следующая конфигурация получается из предыдущей согласно одному из описанных правил.

4. Алгоритм трансляции

В этом разделе описан алгоритм преобразования произвольного иерархического автомата $H = (Var, Clock, Chan, L, L_0, \eta, Type, I, T)$ в сеть временных автоматов $N = (Var', Clock', Chan', PPChan', \mathcal{A})$, обеспечивающий возможность верификации диаграмм состояний UML с помощью средства UPPAAL. Во избежание коллизии наименований в описании автомата и сети, состояния процесса (т.е. временного автомата) будем называть вершинами, а переходы процесса — дугами.

Вначале $Var' = Var$, $Clock' = Clock$, $Chan' = Chan$, $PPChan' = \emptyset$, $\mathcal{A} = \emptyset$. Алгоритм обрабатывает каждое метасостояние l автомата H и ставит ему в соответствие процесс P_l в сети N. В начале обработки метасостояния процесс P_l состоит из одной инициальной вершины *idle* типа o, соответствующей неактивности состояния l. По ходу обработки в процесс P_l добавляются вершины, соответствующие активности вложенных в l компонентов; они соединяются дугами между собой и с вершинами типа c, обеспечивающими корректную активацию и деактивацию компонентов.

Все встречающиеся далее новые каналы имеют тип точка-точка и добавляются в множество $PPChan'$. Записью $i(a)$, где i — инвариант, будем обозначать инвариант, полученный в результате подстановки правых частей элементарных присваиваний семейства a вместо соответствующих левых в инвариант i. При описании вершин по умолчанию подразумевается инвариант *true*.

Обработка AND**-состояния**. В процесс P_l добавляются вершина *active* типа o, соответствующая активности состояния l и всех вложенных в него компонентов, и дуга $active \xrightarrow{?exit[l]} idle$. Если в AND-состояние l вложены метасостояния $l_1, \ldots, l_n$, то в процесс вводятся вершины $stat[l_i]$ типа c, $1 \le i \le n$, и следующие дуги:

$$idle \xrightarrow{?enter[l]} start[l_1] \xrightarrow{!enter[l_1]} start[l_2] \xrightarrow{!enter[l_2]} \ldots start[l_n] \xrightarrow{!enter[l_n]} active.$$

Обработка XOR**-состояния**. Для каждого простого состояния l', вложенного в XOR-состояние l, в процесс P_l добавляется вершина $active[l']$, соответствующая активности состояния l'. Для каждого метасостояния l', вложенного в l, в процесс P_l вводится фрагмент $aux[l'] \xrightarrow{!enter[l']} active[l']$, соответствующий активации и последующей активности состояния l'. Вершина $active[l']$ помечается инвариантом состояния l', вершина $aux[l']$ — конъюнкцией инвариантов вершин входного дерева $\mathcal{ET}(l')$.

Запись $entry[l]$ будет обозначать вершину $active[l]$ для простого состояния, вершину $aux[l]$ для метасостояния и вершину *idle* для выхода. Запись $exit[l]$ будет обозначать вершину $active[l]$ для существенного состояния и вершину *idle* для входа. Каждый переход $l_1 \xrightarrow{g,s,a} l_2$ между состояниями автомата, вложенными в l, в котором l_1 не является метасостоянием или l_2 не является существенным состоянием, порождает дугу $exit[l_1] \xrightarrow{g,s,a} entry[l_2]$ в процессе P_l. Метка s заменяется на $?enter[l]$, если l_1 — вход, и на $?exit[l]$, если l_2 — выход. Если l_1 — метасостояние и l_2 — существенное состояние, то одной такой дуги недостаточно для корректного моделирования перехода автомата, т.к. при совершении перехода нужно деактивировать все метасостояния, внутренние для l_1. Этот случай разобран далее.

Деактивация состояний. Для корректного моделирования деактивации метасостояний автомата в сеть добавляются служебные переменные с областью определения $\{0, 1\}$. Каждая такая переменная $ready[l]$ отвечает возможности совершения

перехода из простого состояния, вложенного в XOR-состояние l, в выход. В процессе P_l выделяется множество $EX(l)$ вершин вида $active[l']$ для простых состояний l', из которых существует переход в выход. Если дуга e в процессе P_l исходит из вершины множества $EX(l)$ и не ведет ни в какую вершину этого множества, то к меткам дуги добавляется присваивание $ready[l] := 0$. Если дуга e в процессе P_l входит в вершину множества $EX(l)$ и не исходит ни из одной вершины этого множества, то к меткам дуги добавляется присваивание $ready[l] := 1$.

Переход вида $tr = l_1 \xrightarrow{g,s,a} l_2$ для метасостояния l_1 и существенного состояния l_2, вложенных в XOR-состояние l, обрабатывается следующим образом. Для метасостояния l_1 перебираются всевозможные множества простых состояний, одновременный переход из которых в выходы завершает работу состояний, внутренних для l_1. Для каждого из таких множеств в процесс P_l добавляются вершины типа c, последовательный переход по которым из состояния $exit[l_1]$ в $entry[l_2]$ деактивирует все внутренние для l_1 состояния. При этом необходимо обеспечить одинаковые условия совершения описанного перехода в автомате и соответствующей последовательности переходов в сети.

Рассмотрим дерево вложенности $\mathcal{T}^\eta(l_1)$. Исключим из него все псевдосостояния и все состояния, из которых не существует переходов в выходы, вложенные в XOR-состояния. Затем, пока это возможно, будем исключать из дерева XOR-состояния, ставшие листьями, и такие AND-состояния, хотя бы одно вложенное состояние которых было исключено. Если получающееся в результате дерево пусто, обработка перехода tr завершена. Иначе после исключения состояний получается некоторое непустое дерево $\mathcal{T}$. Пусть $\mathcal{T}'$ — поддерево дерева $\mathcal{T}$, удовлетворяющее следующим свойствам: корни деревьев $\mathcal{T}$ и $\mathcal{T}'$ совпадают; если AND-состояние входит в дерево $\mathcal{T}'$, то все вложенные в него состояния входят в это дерево; если XOR-состояние входит в дерево $\mathcal{T}'$, то ровно одно вложенное в него состояние входит в это дерево. Множество всех таких поддеревьев будем обозначать записью $AND-OR(\mathcal{T})$.

Для каждого дерева $\mathcal{T}'$, $\mathcal{T}' \in AND-OR(\mathcal{T})$, производятся следующие действия. Пусть $v_1, \ldots, v_n$ — все внутренние вершины дерева $\mathcal{T}'$ и $v'_1, \ldots, v'_k$ — те из них, которые отстоят на одну дугу от листьев. Тогда вершины $exit[l_1]$ и $entry[l_2]$ соединяются следующим образом:

$$exit[l_1] \xrightarrow{g_0,s} deact[v_1] \xrightarrow{?exit[v_1]} deact[v_2] \xrightarrow{?exit[v_2]} \ldots deact[v_n] \xrightarrow{?exit[v_n],a} entry[l_2],$$

где $g_0 = g \wedge trig \wedge inv(a)$, $trig = \bigwedge_{i=1}^{k}(ready[v'_i] = 1)$, inv совпадает с инвариантом вершины $entry[l_2]$ и все вершины $deact[v_i]$ имеют тип c.

Инициализация сети. Для корректной инициализации сети, т.е. достижения конфигурации сети, соответствующей начальной конфигурации автомата, в сеть добавляется процесс $Launch$ следующего вида:

$$v_1 \xrightarrow{!init[l_1]} v_2 \xrightarrow{!init[l_2]} \ldots v_n \xrightarrow{!init[l_n]} done.$$

При этом вершины $v_1, \ldots, v_n$ имеют тип c, вершина $done$ имеет тип o, вершина v_1 является инициальной и предполагается, что множество инициальных состояний автомата имеет вид $L_0 = \{l_1, l_2, \ldots, l_n\}$.

Если AND-состояние l является инициальным, то в процесс P_l добавляется ду-

га $idle \xrightarrow{?init[l]} active$. Если XOR-состояние l_1 и состояние l_2, $l_2 \in \eta(l_1)$, являются инициальными, то в процесс P_{l_1} добавляется дуга $idle \xrightarrow{?init[l_1]} active[l_2]$.

5. Корректность алгоритма трансляции

Чтобы обосновать корректность алгоритма трансляции автоматов в сети, достаточно для некоторого класса темпоральных формул, допустимых в рамках средства UPPAAL, показать их равновыполнимость на множествах вычислений автомата и соответствующей ему сети. Для этого введем понятие размеченной системы переходов, удобное для описания поведения автомата и сети. Затем выделим класс формул, проверка которых обеспечивается средством UPPAAL, и покажем, что оценка истинности этих формул одинакова для систем переходов автомата и соответствующей ему сети. Для этого воспользуемся отношением эквивалентности по прореживанию [14] и убедимся в том, что системы переходов, описывающие поведение автомата и сети, получаемой в результате работы алгоритма, эквивалентны.

Поведение автомата и сети может быть описано с помощью размеченных систем переходов. *Размеченная система переходов* (labelled transition system, LTS) над заданным множеством A — это система $M = (S, s_0, \mathcal{L}, \Rightarrow)$, где

- S — некоторое множество состояний, содержащее инициальное состояние s_0;
- $\mathcal{L} : S \to 2^A$ — разметка состояний;
- $\Rightarrow \subseteq S \times S$ — отношение переходов.

Путем в LTS M называется максимальная последовательность состояний $tr = s_0, s_1, \ldots, s_n, \ldots$, для которой верно соотношение $s_i \Rightarrow s_{i+1}$ для всех i, $i \geq 0$.

LTS $M_Z = (S^Z, s_0^Z, \mathcal{L}^Z, \Rightarrow^Z)$, описывающая поведение системы Z (автомата или сети), строится над множеством $EGuard$ элементарных ограничений данных и таймеров автомата. Заметим, что множества таймеров автомата и сети совпадают и множество переменных автомата вложено в множество переменных сети. При этом S^Z есть множество конфигураций Z, s_0^Z — начальная конфигурация Z, каждая конфигурация помечена множеством ограничений из $EGuard$, истинных в ней, и переход $s_1 \Rightarrow s_2$ допустим в том и только в том случае, если s_2 может быть получена из s_1 согласно одному из правил в описании семантики Z. Заметим также, что множество трасс системы Z совпадает с множеством путей LTS M_Z.

В качестве языка запросов в UPPAAL используется подмножество формул логики TCTL (timed computational tree logic), которое порождается грамматикой

$$\Phi ::= AG\ \varphi \mid AF\ \varphi \mid EG\ \varphi \mid EF\ \varphi \mid \varphi \rightsquigarrow \varphi$$
$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi,$$

где $p \in EGuard \cup \{deadlock\}$. Формула называется допустимой, если она не содержит символа $deadlock$ или имеет следующий вид: $AF\ deadlock$, $AG\ \neg deadlock$, $EF\ deadlock$, $EG\ \neg deadlock$, $\varphi \rightsquigarrow deadlock$, где φ не содержит символа $deadlock$.

Выполнимость формулы Φ на LTS M (обозначение $M \models \Phi$) определена в статье [8] обычным для логики CTL* образом.

Обозначим записью $Alg(H)$ сеть, получаемую из автомата H в результате трансляции. Корректность алгоритма трансляции подразумевает справедливость соотношения $M^H \models \Phi \Leftrightarrow M^{Alg(H)} \models \Phi$ для любого автомата H и любой допустимой фор-

мулы Ф. Для обоснования корректности введем на множестве путей LTS отношение эквивалентности по прореживанию.

Рассмотрим пару путей $tr = s_0, s_1, \ldots$ и $tr' = s'_0, s'_1, \ldots$ в LTS $M = (S, s_0, \mathcal{L}, \Rightarrow)$ и $M' = (S', s'_0, \mathcal{L}', \Rightarrow')$ соответственно, причем либо оба пути конечные, либо оба бесконечные. Пути tr, tr' *эквивалентны по прореживанию*, если найдутся такие возрастающие последовательности целых чисел $\alpha = i_0, i_1, \ldots$ и $\beta = j_0, j_1, \ldots$ одинаковой длины, что:

• $i_0 = j_0 = 1$;

• если пути tr и tr' конечны и имеют длины n и m, то последовательности α, β также конечны и завершаются числами $n + 1$ и $m + 1$ соответственно;

• для любых i, j, k, $1 \le k < |\alpha| = |\beta|$, $i_k \le i < i_{k+1}$, $j_k \le j < j_{k+1}$, справедливо равенство $\mathcal{L}(s_i) = \mathcal{L}'(s'_j)$.

Иначе говоря, пути эквивалентны по прореживанию, если их можно разбить на блоки так, чтобы значение функции разметки на блоках с одинаковыми номерами было одинаково. Две LTS M_1 и M_2 назовем эквивалентными (и запишем этот факт как $M \sim M'$), если для любого пути в LTS M_i найдется эквивалентный ему по прореживанию путь в LTS M_{3-i}, $i \in \{1, 2\}$.

Корректность алгоритма трансляции иерархических автоматов в сети временных автоматов, описанного в предыдущем разделе статьи, непосредственно следует из двух приведенных ниже теорем.

Теорема 1. *Для любых эквивалентных LTS M и M' и любой допустимой формулы* Ф *верно соотношение* $M \models \Phi \Leftrightarrow M' \models \Phi$.

Доказательство. Воспользовавшись соотношениями

• $M \models EG\ \varphi \Leftrightarrow M \not\models AF\ \neg\varphi$,

• $M \models EF\ \varphi \Leftrightarrow M \not\models AG\ \neg\varphi$,

• $M \models \varphi \rightsquigarrow \psi \Leftrightarrow M \models AG(\varphi \to F\ \psi)$,

легко заметить, что рассматриваемый фрагмент TCTL тесно взаимосвязан с темпоральной логикой LTL_{-X}: каковы бы ни были LTS M_1 и M_2, если на них выполняется одно и то же множество формул LTL_{-X}, то на LTS M_1 и M_2 также выполняется одно и то же множество допустимых формул TCTL, не содержащих *deadlock*. Поэтому для допустимых формул, не содержащих символа *deadlock*, утверждение теоремы следует из аналогичного утверждения для формул LTL_{-X}, доказательство которого представлено в работе [14]. Для завершения доказательства теоремы остается лишь заметить, что допустимые формулы, содержащие символ *deadlock*, выражают требования конечности или бесконечности путей, и эквивалентность по прореживанию, очевидно, сохраняет выполнимость этих требований. ■

Теорема 2. *Для любого автомата H верно соотношение $M^H \sim M^{Alg(H)}$.*

Доказательство. Рассмотрим автомат $H = (Var, Clock, Chan, L, L_0, \eta, Type, I, T)$ и соответствующую ему сеть $Alg(H) = (Var \cup Var_{add}, Clock, Chan, PPChan, \vec{A})$. Значения переменных множества V_{add} однозначно определяются значениями переменных множества Var, таймеров множества $Clock$ и вектором активных вершин сети. Поэтому для удобства переменные V_{add} в записи конфигураций будем опускать. Сопоставим конфигурации $c = (\rho, \mu, \nu)$ автомата H конфигурацию $\xi(c) = (\vec{v}, \mu, \nu)$, где

• вершина *done* процесса *Launch* входит в вектор $\vec{v}$ состояний процессов сети $Alg(H)$;
• если $l \in \rho$ и l — AND-состояние, то вершина *active* процесса P_l входит в вектор $\vec{v}$;
• если $l, l' \in \rho$, l — XOR-состояние и $l' \in \eta(l)$, то вершина $active[l']$ процесса P_l входит в вектор $\vec{v}$.

Построим для произвольной трассы $tr = s_1 s_2 \ldots$ автомата H эквивалентную по прореживанию трассу $tr' = s'_1 s'_2 \ldots$ сети $Alg(H)$. Из описания процесса *Launch*, приоритета в выходе из вершин типа c и правила для перехода с сигналом точка-точка в описании семантики сети видно, что любая трасса сети начинается одной и той же последовательностью конфигураций $s'_1, s'_2, \ldots, s'_k$ с неизменными значениями переменных и таймеров, т.е. $s'_k = \xi(s_1)$.

Предположим, что по префиксу $s_1, s_2, \ldots, s_p$ трассы tr уже построен префикс $pr' = s'_1, s'_2, \ldots, s'_k$, для которого $s'_k = \xi(s_p)$, и его конфигурации можно разбить на p блоков так, что значение функций разметки на i-м блоке и в конфигурации s_i совпадают, $1 \le i \le p$. Если $|tr| = p$, то в конфигурации s_p автомата H нет активных переходов. Но тогда из описания процессов сети $Alg(H)$ следует, что построенный префикс pr' нельзя продолжить, и искомая трасса $tr' = pr'$ построена.

Если s_{p+1} получается из s_p продвижением времени на константу d, то к построенному префиксу tr' также может быть добавлена конфигурация s'_{k+1}, получающаяся из s'_k продвижением времени на константу d. Если s_{p+1} получается из s_p выполнением множества E активных переходов, то к построенному префиксу tr' можно добавить последовательность конфигураций, отвечающую следующим действиям: деинициализация метасостояний, из которых исходят активные переходы; затем активация состояний, в которые ведут активные переходы. В любом случае к построенному префиксу добавляются конфигурации $s''_1, s''_2, \ldots, s''_l, s''_{l+1}, \ldots, s''_q$, для которых верно следующее: значения функций разметки на конфигурациях $s''_1, \ldots, s''_l$ и s_p совпадают, и значения функций разметки на конфигурациях $s''_{l+1}, \ldots, s''_q$ и s_{p+1} также совпадают.

Аналогичным образом по трассе tr' сети $Alg(H)$ строится эквивалентная ей по прореживанию трасса tr автомата H с учетом того факта, что в любой трассе сети бесконечно часто встречаются ξ-образы конфигураций автомата. ■

Следствие (корректность алгоритма трансляции). *Для любой допустимой формулы φ и любого автомата H верно соотношение $M^H \models \varphi \Leftrightarrow M^{Alg(H)} \models \varphi$.*

6. Заключение

Нам удалось реализовать приведенный в разделе 4 алгоритм трансляции иерархических автоматов в сети временных автоматов и построить на его основе систему верификации СРВ, описанных в терминах диаграмм состояний UML. Система получает на вход диаграмму состояний в формате XMI, строит по ней промежуточное представление — иерархический временной автомат, транслирует это представление в сеть временных автоматов и применяет средство UPPAAL для проверки темпоральных свойств этой сети. Возможности разработанного средства верификации были проверены на модели системы бортового оборудования, разработанной в проекте DrTesy [15]. Описание этой СРВ состоит из 20 диаграмм, включающих свыше

200 композитных состояний. Для построенного описания СРВ был проверен ряд логических и темпоральных свойств системы, таких как достижимость заданных конфигураций, соблюдение расписаний выполнения задач и обеспечение взаимно исключающего доступа к ресурсам системы. Результаты проведенного эксперимента показывают, что построенное средство верификации способно очень эффективно проводить проверку поведения довольно сложных моделей СРВ, представленных в виде диаграмм состояний UML.

Авторы выражают благодарность анонимному рецензенту за замечания, позволившие улучшить облик статьи.

Список литературы

1. David A., Moller M.O. From HUppaal to Uppaal: a translation from hierarchical timed automata to flat timed automata // Research Series RS-01-11, BRICS, Department of Computer Science, University of Aarhus, March 2001.

2. Lakhnech M.E., Siegal M. Hierarchical automata as model for statechart // Lecture Notes in Computer Science. 1997. V. 1345. P. 187–196.

3. Chen Hai-yan, Dong Wei, Wang Huo-wang. Verify UML statechart with SMV // *Wuhan University Journal of Natural Science*. 2001. V. 6, №1–2. P. 183–190.

4. Jussila T., Dubrovin J., Junttila T., Latvala T., Pores I. Model checking dynamic and hierarchical UML state machines // *Proc. of the 3rd Workshop on Model Design and Validation*. 2006.

5. Latella D., Majzik I., Massink M. Automatic verification of a behavioural subset of UML statechart diagrams using SPIN model-checker // *Formal Aspects of Computations*. 1999. V. 11.

6. Lilius J., Paltor I. vUML: a Tool for Verifying UML Models // *Technical Report TUCS-272. Turku Centre for Computer Science*. 1999.

7. Ober I., Graf S., Ober I. Validating timed UML models by simulation and verification // *Proc. of the Workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS 2003)*. 2003.

8. Behrmann G., David A., Larsen K.G. A Tutorial on Uppaal // Lecture Notes in Computer Science. 2004. V. 3185. P. 200–236.

9. Bengtsson J., Larsen K.G., Larsson F., Pettersson P., Yi W. UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems // Lecture Notes in Computer Science. 1996. V. 1066. P. 232–243.

10. David A., Moller M.O., Yi W. Verification of UML statechart with real-time extensions // IT Tech. Rep. 2003-009, Uppsala: Dep. of Information Technology, Uppsala University. 2003.

11. Muniz A.L.N., Andrade A.M.S., Lima G. Integrating UML and UPPAAL for designing, specifying and verifying component-based real-time systems // *Innovation in Software and System Engineering*. 2009.

12. Alur R., Dill D.L. Automata for modeling real-time systems // Lecture Notes in Computer Science. 1990. V. 443/1990. P. 322–335.

13. Alur R., Dill D.L. A theory of timed automata // Theoretical Computer Science. 1994. V. 126. P. 183–235.

14. Browne M.C., Clarke M.C., Grumberg O. Characterizing finite Kripke structures in propo-sitional temporal logics // Theoretical Computer Science. 1988. V. 59 (1–2). P. 115–131.

15. Chistolinov M.V., Epatko I.V., Bahmurov A.G., Smelyansky R.L., Zakharov V.A., Winter K., Usenko Y. Towards a unified toolset for embedded systems development // *Proc. of the conference UKRPROG-2000 «Problems of Programming»*. 2000. №1–2. P. 316–322.

On the Designing of Model Checkers for Real-Time Distributed Systems

Volkanov D.Yu., Zakharov V.A., Zorin D.A., Konnov I.V., Podymov V.V.

Keywords: verification, real time systems, statecharts, hierarchical automaton, timed automaton

To verify real-time properties of UML statecharts one may apply a UPPAAL, toolbox for model checking of real-time systems. One of the most suitable ways to specify an operational semantics of UML statecharts is to invoke the formal model of Hierarchical Timed Automata. Since the model language of UPPAAL is based on Networks of Timed Automata one has to provide a conversion of Hierarchical Timed Automata to Networks of Timed Automata. In this paper we describe this conversion algorithm and prove that it is correct w.r.t. UPPAAL query language which is based on the subset of Timed CTL.

Сведения об авторах: **Волканов Дмитрий Юрьевич**,
Московский государственный университет им. М.В. Ломоносова, ассистент.
Захаров Владимир Анатольевич,
Московский государственный университет им. М.В. Ломоносова, доцент.
Зорин Даниил Александрович,
Московский государственный университет им. М.В. Ломоносова, аспирант.
Коннов Игорь Владимирович,
Венский технологический университет, ассистент.
Подымов Владислав Васильевич,
Московский государственный университет им. М.В. Ломоносова, аспирант.