ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВУЮЩИХ ГИБРИДНЫХ ПРОЦЕССОВ

В. М. Шпаков

Санкт-Петербургский институт информатики и автоматизации РАН

199178, Санкт-Петербург, 14-я линия В.О., д.39

<vlad@iias.spb.su>

УДК 681.3.06

В. М. Шпаков. **Программирование взаимодействующих гибридных процессов** // Труды СПИИРАН. Вып. 2, т. 1. — СПб.: СПИИРАН, 2004.

Аннотация. *Обсуждаются основанные на правилах исполняемые спецификации взаимодействующих процессов различной динамики. Приводятся возможные структуры правил трансформации состояний процессов и рассматриваются особенности основанного на этих правилах программирования простых гибридных процессов.* — Библ. 4 назв.

UDC 681.3.06

V. M. Shpakov. **Programming of interacting hybrid processes** // SPIIRAS Proceedings. Issue 2, vol. 1. — SPb.: SPIIRAS, 2004.

Abstract. *Rule based executable specifications of interacting different dynamic processes are discussed. Possible structures of process state transition rules are presented and peculiarities of simple hybrid process programming, based on these rules, are considered*. — Bibl. 4 items.

1. Введение

Процессом называется последовательная смена состояний чего-либо. Различают процессы непрерывные, дискретно-событийные и гибридные. Все процессы представляются с помощью переменных состояния. В случае непрерывного процесса изменения носят количественный характер и в качестве переменных состояния используются вещественные переменные, которые за бесконечно малый промежуток времени изменяются на бесконечно малые величины. В случае дискретно-событийных процессов изменения происходят мгновенно в дискретные моменты времени и носят качественный характер. Для их описания используются символьные или лингвистические переменные, например, такие, как "включен", "выключен", "открыт", "закрыт" и т.п. В гибридных процессах происходят изменения обоих указанных типов, причем изменения одного типа могут влиять на характер изменений другого типа. Событийные изменения в гибридных процессах называются режимами. При изменении режима может происходить изменение динамики непрерывной составляющей процесса или скачкообразное изменение величин некоторых непрерывных состояний, или и то и другое. С другой стороны, выход некоторого непрерывного состояния за границу заданного диапазона изменений может представлять собой событие и вызывать изменение режима данного или другого гибридного процесса, или состояния дискретно-событийного процесса.

Для спецификации непрерывных процессов разработан и широко используется математический аппарат обыкновенных дифференциальных уравнений и уравнений в частных производных. Для реализации непрерывного процесса необходимо составить дифференциальное уравнение в соответствии с естественнонаучными законами соответствующей динамики (механики, электро-, газо- или гидродинамики) и найти его аналитическое или численное решение. В 90-е

годы прошлого века в научных кругах высказывались пожелания разработать аппарат аналогичный дифференциальным уравнениям для спецификации дискретно-событийных процессов [1]. В настоящее время эта задача признается невыполнимой, но, к счастью, и необходимость в её решении в значительной степени отпала в основном вследствие многократного увеличения быстродействия современных компьютеров. Разработаны формализмы спецификации дискретно-событийных процессов, которые эффективно реализуются на современных компьютерах. Они основаны на использовании транзитивных моделей, которые предполагают представление процесса с помощью вектора состояния и формирование совокупности продукционных правил для задания переходов из одного состояния в другое.

Проблема возникает вновь при необходимости специфицировать совокупности взаимодействующих процессов различной динамики, когда для представления разнородных процессов желательно использовать один и тот же формализм. Оказалось, что в большинстве случаев является приемлемым дискретное представление непрерывных составляющих гибридных процессов [2], при котором формируются правила аналогичные продукционным, которые определяют смену состояний непрерывного процесса. Эти правила определяют отношение следования между текущим значением непрерывного состояния x и следующим его значением x', отстоящим от текущего на известный (заданный или измеренный) интервал времени. Появляющаяся при этом погрешность дискретизации зависит от размерности процессов, эффективности исполняющей процедуры и быстродействия компьютера.

2. Правила исполняемых спецификаций гибридных процессов

Все правила трансформации состояний процессов имеют в своей основе структуру правил `if … then` и могут интерпретироваться как отношения вида *условие* $\rightarrow$ *действие*. Конкретные структуры правил могут в значительной мере отличаться друг от друга. От структуры правил зависят выразительные возможности формализма и надёжность получаемых на его основе программ.

В [3,4] рассмотрен основанный на правилах формализм спецификации процессов, ориентированный на разработку исполняемых спецификаций совокупностей взаимодействующих дискретно-событийных и гибридных процессов. В этом формализме состояния дискретных процессов и режимы гибридных процессов представляются не символьными, а логическими (двоичными) переменными, что сделано для повышения эффективности исполняющей процедуры при реализации формализма. Такое представление всегда возможно, так как любой символ может быть закодирован двоичным числом. Основные особенности основанного на правилах программирования процессов рассмотрим на примерах использования указанного формализма.

В этом формализме различаются 4 типа дискретных логических переменных состояния и 3 типа непрерывных состояний, образующих соответствующие подмножества:

Σ — переменные входных логических процессов (команды);

Q — состояний дискретно-событийных процессов (состояния);

V — логических переменных для задания динамических режимов гибридных процессов (режимы);

Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

G — предикатов нахождения непрерывных состояний гибридных процессов в заданных диапазонах изменения (предикаты диапазонов);

X_i — переменных входных непрерывных процессов (сигналы);

X_v — переменных непрерывных процессов, задаваемых скоростью их изменения;

X_f — переменных непрерывных процессов, задаваемых функциональными зависимостями.

Входные воздействия являются независимыми переменными, состояние $q \in Q$ и режим $v \in V$ в общем случае могут зависеть от любой логической переменной и, наконец, значения предикатов диапазонов $g \in G$ определяются значениями непрерывных состояний процессов $X = X_i \cup X_v \cup X_f$.

В случае использования логических переменных для представления дискретных состояний и режимов условная часть продукционных правил может быть представлена логической формулой этих переменных. В принципе структура формулы может быть произвольной, ориентированной на конкретную совокупность процессов, но наиболее удобной для широкого применения, на наш взгляд, является использование в качестве условной части правил элементарной конъюнкции логических состояний. Дело в том, что использование конъюнкции делает возможной удобную для пользователя ситуационную интерпретацию условной части правила. В этом случае можно предложить следующее индуктивное определение динамической ситуации:

1. любая логическая переменная (вход, состояние, режим, предикат диапазона) или её отрицание являются динамической ситуацией;
2. любая элементарная конъюнкция динамических ситуаций также является динамической ситуацией.

Введём обозначение W для множества всех логических переменных совокупности процессов $W = \Sigma \cup Q \cup V \cup G$ и обозначение S_j — для произвольной ситуации. После этого определению динамической ситуации можно придать следующий вид:

$S_j = s_{j_1},...s_{j_i},...,s_{j_n}$, где $s_{j_i} = w_{j_i}$ или $s_{j_i} = \neg w_{j_i}$, (1)

где $w_{j_i} \in W, i \le n, n = 1...N, N = |W|$;

Из определения видно, что ситуация имеет истинностное значение, которое определяется каждой входящей в неё переменной.

С учётом сделанного определения и введённых обозначений правило трансформации состояний дискретных переменных и режимов непрерывных переменных будет иметь вид:

$S_j \to w'_{j_1},...,w'_{j_i},...,w'_{j_m}$, где $w'_{j_i} \in Q \cup V, m \in N$. (2)

Штрихи в обозначении переменных состояния означают непосредственное следование во времени, т.е. сначала ситуация в левой условной части правила принимает значение `True`, а затем значения `True` принимают переменные правой части правила. Механизм продвижения времени может быть различным, он определяется исполняющей процедурой.

Поскольку в левую условную часть правил (2) может входить любая логическая переменная любого процесса, то правила этого типа позволяют задать зависимость состояния дискретного процесса или режима непрерывного процесса от переменных любого процесса и тем самым специфицировать взаимозависимость и координацию процессов между собой. Из структуры правила (2)

видно, что состояния $q_j \in Q$ и режимы $v_j \in V$ могут входить как в условную так и в исполнительную части правила, т.е. являться как причиной, так и следствием изменений. Из этого следует, во-первых, то, что с помощью правил (2) можно специфицировать циклические процессы и, во-вторых, что первопричиной изменения установившегося процесса могут быть только входные воздействия или изменения значений предикатов диапазонов $g_j \in G$ непрерывных состояний.

В рассматриваемом формализме нет правил, присваивающих состояниям и режимам значения `False`. Значения `False` присваиваются этим переменным по умолчанию, если на данном шаге не срабатывает ни одно из правил, присваивающих им `True`, т.е. отсутствие трактуется как отрицание. Это сделано для того, чтобы исключить появление конфликтующих правил и повысить надёжность получаемых программ. Указанное ограничение необходимо учитывать при разработке программ процессов. С учётом сделанного замечания правило (2) может интерпретироваться так, что переменные правой части правила имеют значения `True` до тех пор, пока значение `True` имеет ситуация в его левой части.

Правило (2) позволяет присваивать имена ситуациям. Для этого необходимо ввести логическую переменную состояния с именем ситуации и затем связать её при помощи правила (2) с соответствующей ситуацией. После этого эту переменную можно использовать для формирования других ситуаций, получая тем самым составные ситуации.

С каждой ситуацией может быть связано два события: одно – с её возникновением, а второе — с её исчезновением. Поскольку скачкообразные изменения некоторых непрерывных состояний могут происходить в моменты изменения некоторой ситуации, то для их спецификации удобно использовать связанные с этой ситуацией события. Событие можно трактовать как ситуацию минимальной длительности, т.е. длительности равной одному шагу алгоритма обновления состояния. Для формирования такой “событийной” ситуации может быть использован следующий приём. Допустим, надо создать событие, связанное с возникновением ситуации `Sit1`. Создадим переменную состояния `Event1` для представления этого события и вторую вспомогательную переменную `Sit2`, которую определим с помощью правила (2) следующим образом:

`Sit1` $\rightarrow$ `Sit2` (3)

При таком определении `Sit2` повторяет значение `Sit1` с отставанием на один шаг. Тогда `Event1` можно задать с помощью следующего правила:

`Sit1,(not Sit2)` $\rightarrow$ `Event1` (4)

В момент возникновения `Sit1` будет иметь значение `True`, а `Sit2` – значение `False` и, значит, составная ситуация в условной части правила (4) будет иметь значение `True` , и, следовательно, переменная `Event1` на следующем шаге будет иметь значение `True`. Но на следующем шаге в соответствии с правилом (3) `Sit2` примет значение `True` и, следовательно, в соответствии с (4) `Event1` примет значение `False`. Таким образом, `Event1` будет иметь значение `True` только в течение одного шага, непосредственно следующего за возникновением ситуации `Sit1`.

Для спецификации диапазонов изменения непрерывных состояний могут быть использованы любые предикаты от соответствующих переменных. Для

определённости будем использовать правило, предложенное в указанном выше формализме:

$$((x_{j_1} \geq a_k + x_{j_2}) \wedge (x_{j_3} \leq b_k + x_{j_4})) \rightarrow g'_k, \quad (5)$$

где $x_{j_1}, x_{j_2}, x_{j_3}, x_{j_4} \in X, g'_k \in G$, a_k и b_k — константы, соответствующие некоторому диапазону. Более сложные предикаты диапазонов изменения непрерывных переменных могут быть заданы совокупностью правил вида (5) и (2).

Рассматриваемый формализм имеет два типа правил трансформации непрерывных состояний, одно задаёт текущую скорость изменения состояния, а второе непосредственно задаёт следующие значения состояний на основании функциональных или интегро-дифференциальных зависимостей следующих состояний от текущих состояний. Правило спецификации скоростей имеет следующий вид:

$$S_j \rightarrow (\dot{x}'_k = (c_{jk} + a_{jk} x_m)), \; j,k,m \in N, \quad (6)$$

где c_{jk}, a_{jk} — коэффициенты, соответствующие изменению k-ой переменной при данном режиме, определяемом ситуацией S_j, $\dot{x}'_k$ — скорость изменения (производная) специфицируемой данным правилом переменной на следующем шаге. Совокупность правил такого типа эквивалентна системе линейных разностных уравнений и позволяет описывать линейные динамические структуры произвольного вида. Для реализации этого правила исполняющая процедура должна на каждом шаге производить вычисление координаты на основании определяемой правилом (6) скорости.

$x'_k = x_k + \dot{x}_k \Delta t_i$, где x'_k — значение x_k на следующем шаге.

Правило спецификации функциональных состояний на каждом шаге производит вычисление значения специфицируемой непрерывной переменной из своей правой части в соответствии с заданной функциональной зависимостью и текущими значениями аргументов. Вычисленные значения присваиваются состоянию на следующем шаге.

$$S_j \rightarrow (x'_k = f_{jk}(x_m)), \; j,k,m \in N, \quad (7)$$

где x'_k — значение x_k на следующем шаге, f_{jk} — функция, задающая значение k-ого непрерывного состояния в ситуации S_j. С помощью правил этого типа могут быть специфицированы интегро-дифференциальные и произвольные функциональные зависимости.

В следующем параграфе рассмотрим особенности программирования процессов на примерах использования приведённых выше правил.

3. Примеры основанных на правилах программ простых процессов

3.1. Затухающий колебательный процесс

Моделирование динамических процессов выше первого порядка производится с помощью наборов правил, описывающих элементарные динамические зависимости. Формирование модели производится путем соединений динамических звеньев, задаваемых отдельными правилами, между собой. Необходи-

Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

мые соединения реализуются за счёт использования одних переменных в качестве аргументов правил, описывающих другие переменные. Методика аналогична составлению схем моделей при использовании аналоговых компьютеров. Разработку программы удобно начинать с составления блок схемы динамической системы. На рис. 1 представлена схема системы, содержащей два последовательно соединенных интегратора, охваченных общей отрицательной обратной связью, и демпфирующей обратной связью по скорости.

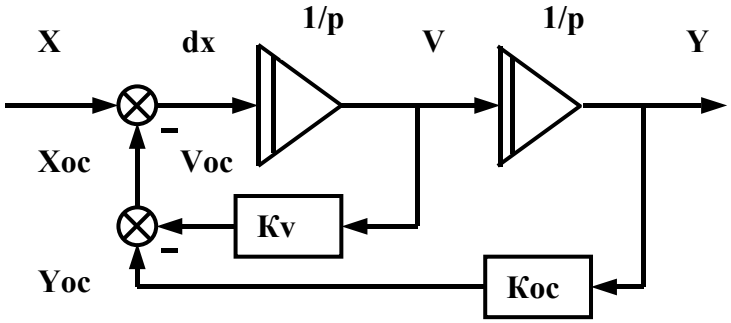

Рис. 1. Схема динамической системы 2-го порядка

Программа модели протекающих в системе процессов использует два правила специфицирующих интегральные зависимости и, в соответствии со схемой, правила, описывающие пропорциональные зависимости и сумматоры. Имена функциональным непрерывным состояниям присвоим соответствующие обозначениям на схеме, заключенным в кавычки. В данном случае система имеет один режим работы, поэтому в качестве переменной режима можно использовать логическую константу `True`. Конкретная форма правил определяется соответствующим редактором правил. Мы будем при написании правил использовать синтаксис близкий к языку Pascal. При этом процесс в рассматриваемой системе может быть представлен следующей совокупностью правил типа (7):

```
if True then "dx":=-1.0*Сумма("X","Xoc");
if True then "V":=1.0*Интеграл("dx");
if True then "Y":=1.0*Интеграл("V");
if True then "Yoc":=Koc*"Y";
if True then "Voc":=Kv*"V";
if True then "Xoc":=1.0*Сумма("Yoc","Voc").
```

Как нетрудно видеть правила специфицируют преобразования, показанные на схеме: сигнал ошибки с сумматора поступает на вход первого интегратора, с его выхода — на вход второго интегратора и далее на сумматор формирования обратной связи. На рис. 2 изображён график процесса, задаваемого приведёнными правилами и возникающего при начальном отклонении `Y=300` и нулевых значениях остальных состояний. На графике представлены изменения выходного состояния и скорости. Декремент затухания определяется значением коэффициента отрицательной обратной связи по скорости, а частота — коэффициентами передачи прямой и обратной цепей.

Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

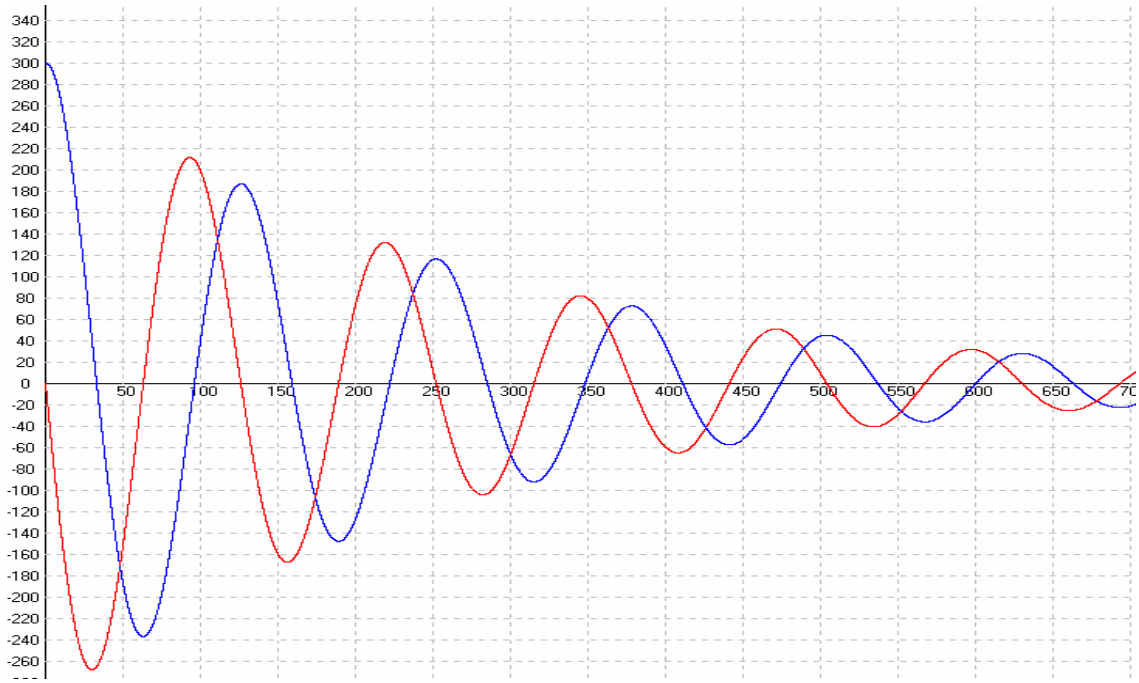

Рис. 2. Координата и скорость затухающего колебательного процесса

3.2. Простейшая гибридная система — прыгающий мяч

Прыгающий мяч представляет собой простейшую гибридную динамическую систему. Движение свободного мяча определяется действием силы тяжести. Пока высота мяча над поверхностью больше нуля его движение является равномерно ускоренным. В момент, когда высота равна нулю, происходит событие, в результате которого скорость мяча мгновенно меняет знак и немного уменьшается по величине. Дальнейшее движение опять определяется только действием силы тяжести, т.е. происходит с ускорением свободного падения.

Для спецификации описанного процесса используем две непрерывные переменные состояния “высота_x1”, “скорость_x2”, предикат диапазона с именем “x1=0” и логическую событийную переменную, например, с именем “событие_x1=0”. Предикат “x1=0” вычисляется с помощью следующего правила типа (5):

```
if “высота_x1” ≤ 0.05 then “x1=0”:=True;                    (8)
```

Он определяет ситуацию, в которой высота близка к нулю. Эта переменная может быть истинной в течение более чем одного шага сканирования базы правил. Поэтому её нельзя использовать в правиле (7) для изменения скорости, так как скорость может поменять знак больше одного раза, что недопустимо. Логическую переменную “событие_x1=0” формируем следующим образом. Вначале с помощью продукционного правила типа (2) формируем вспомогательную переменную состояния “Re(x1=0)”, которая должна повторять значения предиката “x1=0” с отставанием по времени на один шаг (9). Тогда правило (10) будет определять переменную “событие_x1=0” истинной только в ситуации, когда уже “x1=0” = True и ещё “Re(x1=0)” = False, т.е. точно в течение одного шага сканирования правил.

```
if “x1=0” then Re(x1=0):=True;                              (9)
if “x1=0” and not(“Re(x1=0)”) then “событие_x1=0” =True;    (10)
```


Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

С учётом этого правила для непрерывных состояний можно представить в виде (11) – (13).

```
if True then "высота_x1":=1.0*Интеграл("скорость_x2");      (11)
if True then "скорость_x2":=1.0*Интеграл("ускорение_g");    (12)
if "событие_x1=0" then "скорость_x2":=-0.9*"скорость_x2";   (13)
```

Совокупность правил (8) – (13) составляют программу процесса изменения состояний прыгающего мяча. На рис. 3 приведены графики изменения высоты и скорости прыгающего мяча, соответствующие приведённым правилам.

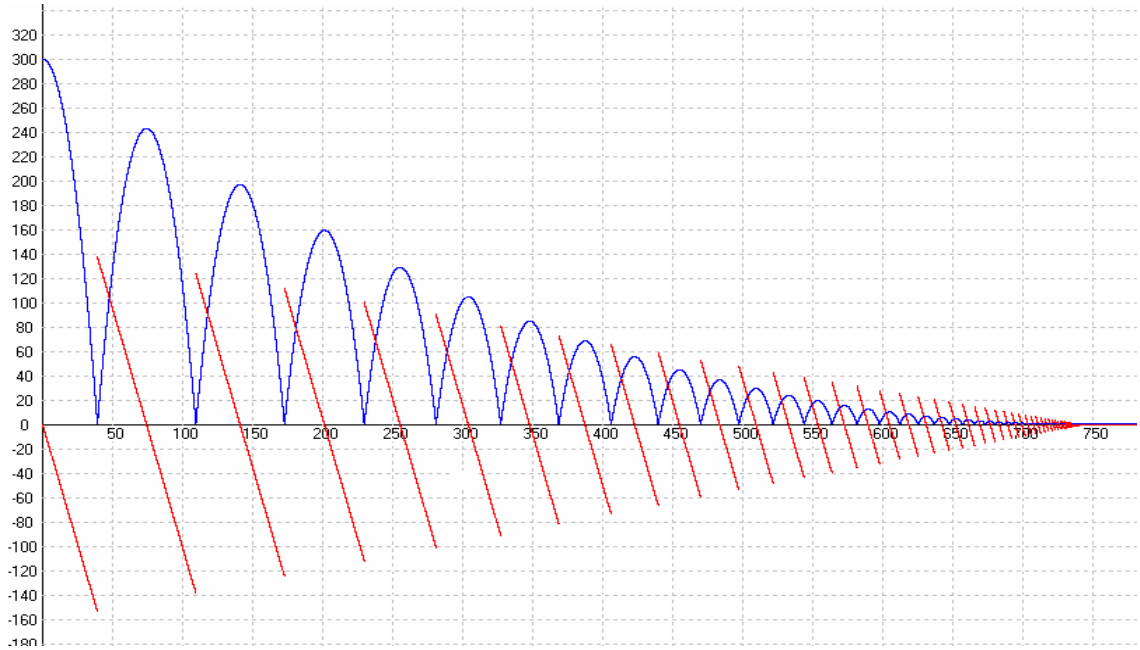

Рис. 3. Высота и скорость прыгающего мяча

3.3. Процесс релейной стабилизации уровня

В качестве второго примера модели простой гибридной динамической системы рассмотрим модель релейного стабилизатора уровня. Система представляет собой ёмкость, содержащую впускной и выпускной клапаны (Vin, Vout), и два реле уровня, одно из которых срабатывает при достижении уровнем верхнего предельного значения, а второе — нижнего. Задача системы поддерживать уровень в заданных пределах, открывая впускной клапан при достижении уровнем нижнего предельного значения (Ldn) и закрывая его при достижении уровнем верхнего предельного значения (Lup). Схематическое представление системы приведено на рис. 4.

При моделировании данной системы была сформирована специфицируемая скоростью непрерывная переменная состояния `"Уровень"` для представления уровня жидкости и функциональная переменная `"скорость"` для представления скорости изменения уровня. Для спецификации логики работы системы были сформированы следующие логические переменные: внешняя команда `"Пуск"` для задания режима работы, состояния `"Vin_открыт"`, и `"Vout_открыт"` для представления состояния клапанов, и `"L < 160"`, `"L > 240"` для представления предикатов диапазонов (срабатывания реле уровней).

Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

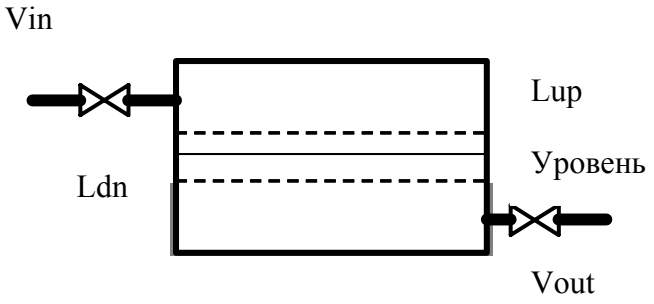

Рис. 4. Функциональная схема релейного стабилизатора уровня

Изменения уровня специфицируем с помощью правил (6) трансформации скоростей изменения непрерывного состояния:

```
if "Vin_открыт" then ∂/∂t ("Уровень"):=25;
if "Vout_открыт" then ∂/∂t ("Уровень"):="скорость";
```

Первое из этих правил задает скорость повышения уровня 25 ед. в секунду при условии, если открыт впускной клапан Vin. Второе правило задает скорость изменения уровня равной функциональной переменной "скорость" при условии, если открыт выпускной клапан Vout. При открытых обоих клапанах скорость изменения уровня будет равна алгебраической сумме скоростей, задаваемых обоими правилами. Переменная непрерывного состояния "скорость" определяет функциональную (в данном случае пропорциональную) зависимость скорости изменения уровня от величины уровня и задается правилом типа (7).

```
if True then "скорость":=-0.10 * "Уровень";
```

Предикаты диапазонов "L < 160", "L > 240" задаются правилами типа (5). Первый из них принимает значение True, когда "Уровень" становится меньше 160, а вторая – когда уровень превышает 240.

```
if ("Уровень" > 240) then "L > 240":=True;
if ("Уровень" < 160) then "L < 160":=True;
```

С помощью правил трансформации ситуаций (2)) определим логику управления клапанами.

```
if "Пуск" then "Vout_открыт":=True;
if "Пуск" and "L < 160" then "Vin_открыт":=True;
if "Vin_открыт" and not("L > 240") then "Vin_открыт":=True;
```

Первое правило устанавливает отношение между командой и состоянием клапана, а именно, если команда "Пуск" имеет значение True, то выпускной клапан открыт, логическая переменная "Vout_открыт", представляющая его состояние, имеет значение True. Следующие два правила специфицируют автоматический режим работы системы. Второе правило задаёт открытое состояние впускного клапана, если значение уровня не превышает 160. Это правило обеспечивает открытие впускного клапана, когда уровень падает до значения 160. Третье правило блокирует впускной клапан в открытом состоянии до тех пор, пока уровень не поднимется до 240. При достижении уровнем этого значения блокировка снимается и впускной клапан закрывается. Тем самым осуществляется автоматическое поддержание уровня в заданных пределах.

График процесса изменения уровня, возникающего при подаче команды "Пуск" и нулевом начальном значении уровня, представлен на рис. 5.

Труды СПИИРАН. 2004. Вып. 2. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2004. Issue 2. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

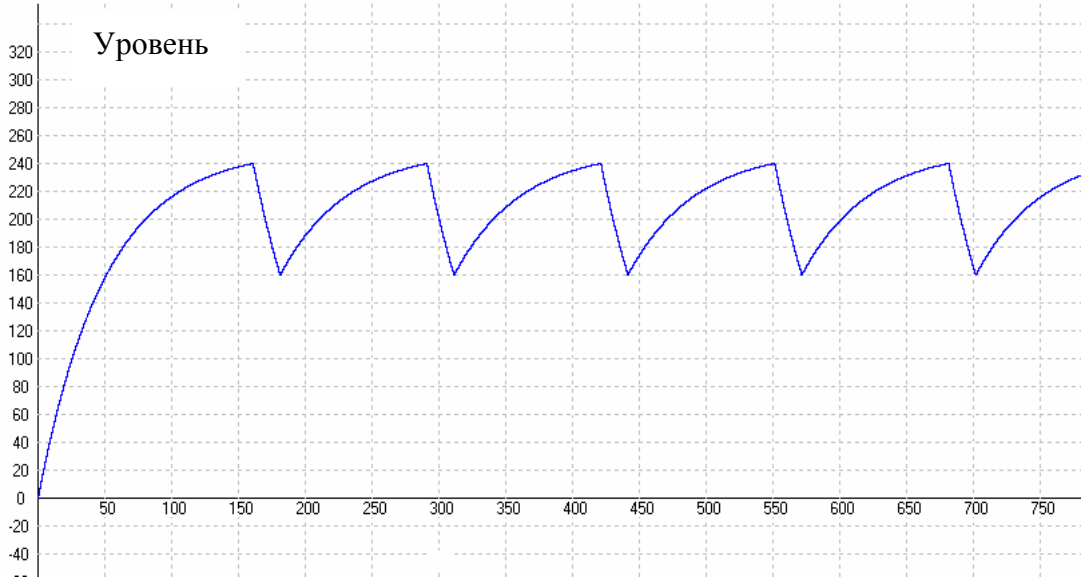

Рис. 5. График процесса стабилизации уровня

Изменения уровня, как при нарастании, так и при понижении происходят по отрезкам экспонент. Так как скорость вытекания жидкости растёт при увеличении уровня, то при закрытом впускном клапане скорость понижения уровня уменьшается, а при открытом впускном клапане скорость повышения уровня уменьшается.

4. Заключение

Как вектора состояний процессов и списки правил, так и исполняющие процедуры обработки правил могут быть эффективно реализованы с помощью универсальных алгоритмических языков программирования. Полученные программные модули могут служить основой создания компьютерных сред или компонентов моделирования процессов, встраиваемых в различные программные продукты. Полученные таким путём приложения предоставляют пользователю возможность самостоятельно программировать модели процессов на языке близком к предметной области, что в ряде случаев позволяет расширить области применения программных средств и длительности их жизненных циклов.

Литература

[1] *Хо Ючзи.* Динамика систем с дискретными событиями // ТИИЭР: Труды института по электротехнике и радиоэлектронике. Перевод с английского М: Мир, 1989. Том 77, №1. С. 4–9.

[2] *Alur R., Henzinger T. A., Lafferriere G., Pappas G. J.* Discrete Abstractions of Hybrid Systems // Proceedings of the IEEE.2000. Vol. 88. P. 971–984.

[3] *Шпаков В. М.* Ситуационные спецификации имитационных моделей гибридных реактивных систем // Труды СПИИРАН. Вып. 1, том 2 . СПб: СПИИРАН, 2002.С. 212–222.

[4] *Shpakov V.* Situation Based Transition Model for Discrete-Continuous Production Processes Simulation // Proceedings of 9th International Multi-Conference Advanced Computer Systems ACS'02, Supply Chain Management SCM'2002, Miedzyzdroje, Poland, October 23-25, 2002, Part 2. P. 507–514.