Российская академия наук
Сибирское отделение
Институт систем информатики
имени А. П. Ершова

В.А. Непомнящий, Е.В. Бодин, С.О. Веретнов

ЯЗЫК СПЕЦИФИКАЦИЙ РАСПРЕДЕЛЕННЫХ СИСТЕМ DYNAMIC-REAL

Препринт
147

Новосибирск 2007

В настоящей работе описан язык спецификаций распределенных систем Dynamic-REAL (dREAL), который расширяет разработанный ранее язык Basic-REAL посредством динамических конструкций порождения и уничтожения экземпляров процессов. Язык dREAL включает подъязыки выполнимых и логических спецификаций. В работе представлены синтаксис и формальная семантика языка dREAL. В качестве примера рассматривается протокол управления сетью касс-терминалов.

Работа частично поддержана грантами РФФИ № 07-07-00173а и 06-01-00464а.

© Институт систем информатики имени А. П. Ершова СО РАН, 2007

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

V.A. Nepomniaschy, E.V. Bodin, S.O. Veretnov

DISTRIBUTED SYSTEMS SPECIFICATION LANGUAGE DYNAMIC-REAL

Preprint
147

Novosibirsk 2007

The present paper describes a distributed systems specification language Dynamic-REAL (dREAL) which extends the language Basic-REAL developed previously by dynamic constructs for generating and removing process instances. The dREAL language includes sublanguages of executable and logic specifications. Syntax and formal semantics of the dREAL language are represented in this paper. As an example, a protocol for control of a booking terminals net is considered.

© A. P. Ershov Institute of Informatics Systems, 2007

1. ВВЕДЕНИЕ

Последние годы заметно возрастает роль формальных методов, применяемых для разработки распределенных систем, таких как, например, коммуникационные протоколы. Это связано с тем, что для современных распределенных систем усложняется документирование, анализ и верификация. Для преодоления указанных трудностей используются языки выполнимых спецификаций SDL, Estelle, LOTOS, принятые в качестве стандарта международной организацией ITU (International Telecommunication Union). Среди этих языков наиболее активно на практике применяется язык SDL [1,11]. Верификация выполнимых спецификаций, представленных на языке SDL, заключается в проверке корректности их ключевых свойств и является известной открытой проблемой современного программирования.

Развитие формальной семантики языка SDL необходимо для разработки формальных методов анализа и верификации. Интересные формализмы для описания формальной семантики SDL были предложены в [5, 6, 7, 9, 10]. Однако при использовании этих формализмов формальная семантика языка SDL остается громоздкой.

Идея нашего подхода к проблеме верификации SDL-спецификаций состоит в разработке модельных языков, ориентированных на верификацию, которые были бы комбинированными, т.е. включали подъязыки выполнимых и логических спецификаций. Такой комбинированный язык спецификации Basic-REAL (bREAL) был представлен в [3, 8], где описана полная структурная операционная семантика выполнимых спецификаций в виде систем переходов, которая позволила доказать важные семантические свойства. Упрощенная версия этого языка, названная Elementary-REAL, представлена в [2]. В [4] этом описана система SRPV (SDL/REAL Protocol Verifier) для моделирования, анализа и верификации статических SDL-спецификаций распределенных систем, которая базируется на трансляции языка SDL в язык bREAL.

Цель настоящей работы — описать язык Dynamic-REAL (dREAL), полученный расширением языка Basic-REAL посредством динамических конструкций порождения и уничтожения экземпляров процессов. Данная работа состоит из 6 разделов. В разд.2 дается обзор языка выполнимых спецификаций dREAL. В разд. 3 представлена формальная семантика динамических конструкций языка dREAL. Разд. 4 посвящен описанию логического подъязыка dREAL, предназначенного для представления свойств. В разд. 5 в качестве примера рассматривается протокол управления сетью терминалов выдачи билетов неограниченному числу пассажиров. В разд. 6

обсуждаются достоинства языка dREAL и перспективы развития этого направления.

2. ОБЗОР ЯЗЫКА СПЕЦИФИКАЦИЙ DYNAMIC-REAL. ПРЕДСТАВЛЕНИЕ СИСТЕМ

В языке dREAL определены две синтаксические формы описания систем. Одна форма — текстовая, другая — графическая, в которой система описывается в виде диаграмм, состоящих из графических символов.

Система состоит из одного или нескольких блоков, соединенных между собой и с окружающей средой каналами. Содержимым каналов являются сигналы, возможно, с параметрами. Каналы могут иметь следующие структуры: очереди, магазины, мультимножества. Время жизни сигнала с параметрами в канале может быть ограничено числом запросов на чтение этого сигнала и/или непосредственно интервалом времени. Из окружения система получает сигналы и в окружение может возвращать ответные сигналы, причем запуск системы возможен только по сигналу извне. Каждый блок и канал в системе имеют уникальное имя.

Средствами dREAL обеспечивается многоуровневое описание системы. Блок может быть разбит на более мелкие единицы — подблоки. Все подблоки, объединенные в блок, работают параллельно и взаимодействуют друг с другом и с внешней средой посредством посылки сигналов в каналы.

На самом нижнем уровне этой иерархии блоки содержат процессы, являющиеся функциональными компонентами системы и определяющие ее поведение. Внутри блока каналы связывают процессы между собой. Процесс языка dREAL описывает последовательность таких действий, как изменение переменных, чтение сигналов из каналов, запись сигналов в каналы, очистка каналов. С каждым действием в dREAL ассоциируется временной интервал, который задает продолжительность выполнения действия. Допускается использование оператора недетерминированного перехода.

В языке dREAL предлагается новая концепция времени — асинхронные мультичасы, синхронизированные посредством линейных неравенств на их скорости хода. Множество поведений выполнимой спецификации можно ограничить посредством условий справедливости. В результате мы будем рассматривать только так называемые справедливые поведения, т.е. такие поведения, в которых каждое из условий справедливости реализуется бесконечно часто.

Большинство статических конструкций языка dREAL аналогичны соответствующим конструкциям языка bREAL [3]. Различие состоит в том, что в языке dREAL каждому экземпляру процесса присваивается личный идентификатор, а каналы, которые являются входными для данного экземпляра процесса, индексируются посредством идентификатора этого экземпляра процесса. В язык dREAL включены две следующие конструкции (без указания идентификаторов экземпляров процесса):

- послать сигнал всем экземплярам процесса через заданный канал;
- очистить заданный канал для всех экземпляров данного процесса.

В отличие от языка bREAL, в язык dREAL включены две динамические конструкции, которые позволяют каждому процессу порождать или уничтожать экземпляры самого себя или других процессов в своем блоке.

В качестве иллюстративного примера Рис.1, В представляет результат порождения экземпляра процесса P2 из конфигурации, представленной на Рис.1, А.

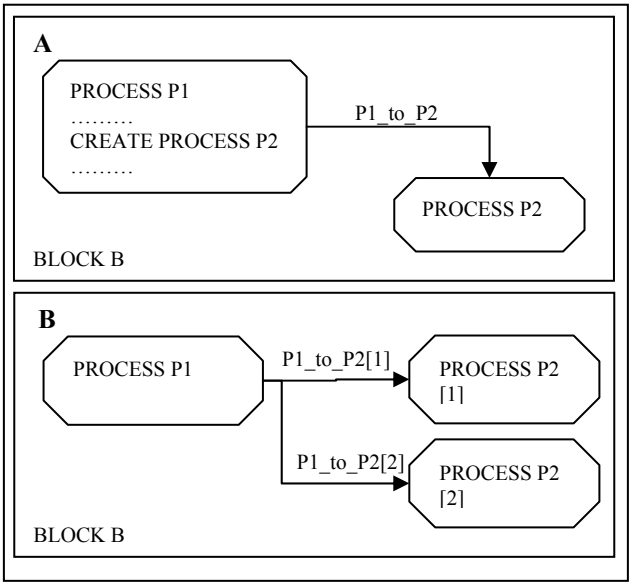


Рис. 1. Создание экземпляра процесса

В качестве второго примера Рис.2, С представляет результат уничтожения экземпляра процесса P2 из конфигурации, представленной на Рис.2, А, причем на Рис. 2, В показана передача сигнала уничтожения kill по соответствующему каналу.

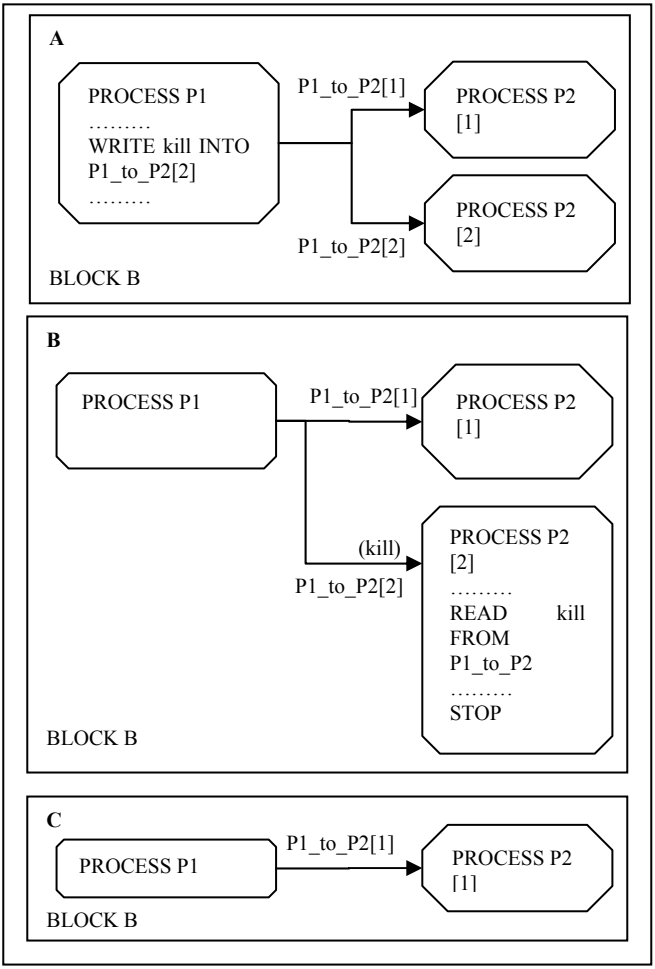


Рис. 2. Уничтожение экземпляра процесса

Всякая спецификация (как выполнимая, так и логическая) носит иерархическую структуру и состоит из заголовка, шкалы, контекста, схемы, подспецификаций.

Заголовок спецификации определяет ее имя и вид — блок или процесс, формула или предикат.

Шкала спецификации есть конечное множество линейных равенств и неравенств, в которых в качестве переменных выступают неинтерпретированные единицы времени, а в качестве коэффициентов — целые положительные числа, расширенные символами «сейчас» (NOW) и «бесконечность» (INF). Каждая единица времени, которая встречается в шкале, есть такт независимых часов, а сама шкала задает совокупность ограничений для синхронизации хода часов таким образом, что все равенства и неравенства шкалы выполняются.

Для формального описания семантики времени обычно применяется подход, близкий к модели фиктивного такта. Для границ интервала используются следующие обозначения: AFTER — для левой открытой, UNTIL — для правой открытой, FROM — для левой замкнутой, UPTO — для правой замкнутой.

Контекст спецификации есть конечное множество определений типов и описаний переменных и каналов. Каждый объект, описанный в контексте, имеет только одно описание, хотя идентификаторы у разных объектов могут совпадать.

В языке dREAL есть как стандартные, так и нестандартные структуры каналов. Список стандартных структур каналов содержит ограниченные и неограниченные по емкости очереди, стеки и мультимножества, а значит и элементарный буфер в качестве частного случая одноэлементного мультимножества.

Всякий объект, объявляемый в контексте, должен содержать атрибут локации (location), определяющий место использования — в схеме самой спецификации (значение OWN) или в определенной подспецификации (имя подспецификации в качестве значения).

Схема выполнимой спецификации состоит из условий справедливости и диаграммы.

Диаграмма блока состоит из маршрутов, каждый из которых является маршрутом канала и связывает имя подблока с именем другого подблока или со средой. Диаграмма блока может иметь как графическую, так и линейную формы. Графическая форма диаграммы блока — это размеченный граф, вершины которого помечены именами подблоков или символом внешней среды, а дуги соответствуют каналам. В графическом синтаксисе

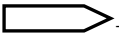
диаграмм блоков каналы изображаются сплошными стрелками, а связанные ими имена процессов и среда заключаются в прямоугольные рамки. Линейная форма диаграммы блока — это просто перечисление в некотором порядке смежных вершин и связывающих их дуг описанного графа.

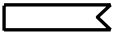
Диаграмма процесса в языке dREAL представляет собой некоторое обобщение понятия программы и состоит из переходов, каждый из которых, в свою очередь, состоит из управляющего состояния («метки»), тела, временного интервала и (недетерминированного) скачка («перехода») на следующие управляющие состояния. Тело перехода определяет одно из следующих действий:

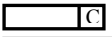
- прочитать определенный сигнал из входного канала и одновременно занести значения его параметров в определенные переменные;
- записать определенный сигнал в выходной канал и одновременно передать ему в качестве значений параметров значения определенных переменных;
- в соответствии с определенной программой изменить значения программных переменных процесса.

Любое из этих действий исполняется мгновенно, но исполнить его можно только в интервале времени, предусмотренном для этого перехода, т.е., когда с момента передачи управления состоянию, которое метит переход, пройдет время, укладывающееся во временной интервал перехода.

Диаграмма процесса может иметь как графическую, так и линейную формы. Графическая форма диаграммы процесса представляет собой размеченный граф с вершинами 2-х типов: управляющее состояние и тело состояния, а дуги соответствуют передаче управления. Для диаграмм процессов используется следующий графический синтаксис: все состояния (в зависимости от сортов помеченных переходов) помещаются в

 — «вымпел», если тело состояния метит переход с посылкой сигнала,

 — «флажок», если тело состояния метит переход с приемом сигнала,



— «знамя», если тело состояния метит переход с чисткой канала,

 — «ящик», если тело состояния метит переход с исполнением программы,

 — «ромб», если тело состояния метит переход с выбором,

 — «овал», имя управляющего состояния.

Графическая форма диаграммы процесса дублирует линейную форму диаграммы процесса и может отсутствовать в спецификации процесса, так как играет вспомогательную роль. Линейная форма диаграммы процесса — это полное описание всех переходов. Она должна присутствовать в спецификации процесс. Линейная форма диаграммы процесса — это полное описание всех переходов.

Неформально говоря, процесс выполняется недетерминировано. С внешней средой процесс обменивается сигналами с параметрами через входные/выходные каналы. Это означает, что в канале, связанном с внешней средой, возможно возникновение и исчезновение сигнала.

Подспецификации — это спецификации подблоков и подформул, имена которых используются в диаграмме самой спецификации. Как уже было сказано, процессы и предикаты являются элементарными спецификациями и не имеют подспецификаций. Имя каждой подспецификации должно быть описано в контексте спецификации и явно присутствовать в схеме спецификации. Шкала каждой подспецификации является расширением шкалы спецификации. Аналогично, множество определений типов каждой подспецификации содержит все определения типов самой спецификации. Если какой-либо объект описан в контексте спецификации с указанием имени какой-либо подспецификации в качестве его локации, то этот же объект должен быть описан и в этой подспецификации с другой локацией, а в остальном — аналогично.

В языке dREAL иерархия многоуровневая — блок может содержать другие блоки любой вложенности, а формула может содержать подформулы.

3. СЕМАНТИКА ДИНАМИЧЕСКИХ КОНСТРУКЦИЙ ЯЗЫКА DYNAMIC-REAL

3.1 Формальная семантика. Предварительные понятия

Полная формальная семантика языка dREAL состоит из 14 правил шага (одно для блока и 13 — для процессов).

Конфигурация спецификации — снятая в одно мгновение совокупность, состоящая из

- показаний мультичасов,
- значений всех переменных,
- содержащихся в каналах сигналов со значениями параметров,

– характеристик активности и значений задержки для всех состояний всех процессов спецификации.

Семантика выполнимых спецификаций будет определяться в терминах событий и правил шага.

Есть восемь видов событий:

– отправление сигнала с параметрами в канал (WriTing и WriTing ALL);
– получение сигнала с параметрами из канала (ReaDiNg);
– чистка входного канала (CLeaNing INput);
– чистка выходного канала (CLeaNing OUTput и CLeaNing OUTput ALL);
– исполнение программы (EXEcuting);
– создание экземпляра процесса;
– уничтожение экземпляра процесса;
– невидимое событие INV (INVisible).

Правило шага имеет вид CND |= CNF1<EVN>CNF2, где

CNF1 — текущая конфигурация,

CNF2 — результирующая конфигурация,

CND — условие на конфигурации CNF1, CNF2 и на событие EVN.

Интуитивная семантика правила шага следующая: если условие CND выполняется, то выполнимая спецификация может преобразовать событием EVN конфигурацию CNF1 в конфигурацию CNF2. Счетная последовательность конфигураций является поведением спецификации тогда и только тогда, когда для любой последовательной пары конфигураций CNF1 и CNF2 из этой последовательности существует событие EVN и условие CND, такие, что CND |= CNF1<EVN>CNF2 есть частный случай соответствующего правила шага. Поведение выполнимой спецификации называется справедливым тогда и только тогда, когда каждое из условий справедливости выполняется бесконечно часто в конфигурациях, которые встречаются в этом поведении.

Зафиксируем модель со структурами для каналов M=(DOM, INT, DTS) и масштаб времени SCL., где непустое множество DOM — область модели, INT — интерпретация символов отношений и операций над DOM, а DTS — структуры для каналов, т.е. отображение, сопоставляющее каждому каналу (его расширенному имени) структуру этого канала.

Пусть CNF' <EVN> CNF" означает выражение “*срабатывание*”, которое представляет возможный переход из конфигурации CNF' в конфигурацию CNF" посредством события EVN.

Если EVN — не INV, то срабатывание называется *активным* срабатыванием.

3.2 Условия, используемые в правилах семантики динамических конструкций

12 правил семантики всех статических конструкций даны в Приложении 2.

Опишем здесь условия, используемые в правилах для динамических конструкций.

Пусть CNF1 = (*T*1,*V*1,*C*1,*S*1,*P1*) и CNF2 = (*T*2,*V*2,*C*2,*S*2,*P2*) — пара конфигураций, где Si=(ACTi, DELi).

Здесь *Ti* — таймеры (значения текущих задержек для активных состояний каждого процесса), *Vi* — значения переменных,*Ci* — содержимое каналов, *Pi* — идентификаторы экземпляров процесса, *Si* –значение активности состояний в соответствующей конфигурации, а именно: *ACTi* определяет, какие состояния являются активными в процессах, а *DELi* — значения задержек в состояниях (задержки равну нулю в неактивных состояниях).

- TIME.CONST: $T1=T2$
  (показания часов не изменяются);
- DEL.ZER: $\forall state.DEL2(state)=0$.
  (в следующем состоянии таймеры начинают считать с нуля);
- DEL.IN(*state*,*interval*): $DEL1(state) \in interval$.
  (задержка в состоянии *state* не выходит за пределы интервала *interval*);
- ACT.ACT(*state*): $state \in ACT1$.
  (состояние *state* активно перед срабатыванием перехода);
- ACT.NEXT(*next*): $next \in ACT2$.
  (состояние *next* становится активным после срабатывания перехода);
- CrTR(*state*,*process*,*interval*,*next*):
  диаграмма процесса *process* содержит переход «*state* `CREATE` *process interval* `JUMP` *Set*», где *Set* — множество состояний таких, что $next \in Set$;
- StTR(*state*,*process*): диаграмма процесса *process* содержит переход «*state* `STOP`»;

– VAR.CREATE(*process*): $V2=V1\cup new(vars(process),max(P2(process))$ (конфигурация расширена переменными нового экземпляра процесса *process, а* функция *new* описана в разделе 3.3);
– CHAN.CREATE(*process*): $C2=C1\cup new(channels(process),max(P2(process)))$ (конфигурация расширена каналами нового экземпляра процесса *process*);
– ACT.CREATE(*process*): $S2=S1\cup new(states(process),max(P2(process)))$, $T2=T1\cup new(timers(process),max(P2(process)))$ (конфигурация расширена значениями активности состояний и задержек нового экземпляра процесса *process*);
– PROC.CAN_CREATE(*process*): $|P1(process)| <$ max_instances(*process*), где max_instances(*process*) — максимально возможное количество экземпляров данного процесса, которые могут существовать одновременно (указывается в спецификации в заголовке процесса; если оно не указано — процесс может иметь только один экземпляр), $|P1(process)|$ — количество экземпляров процесса в первой конфигурации;
– PROC.CREATE(*process*): $P2(process)=P1(instances(process))\cup\{max(P1(process))+1\}$ (конфигурация расширена новым экземпляром процесса *process*);
– VAR.DELETE(*process*): $V2=V1 \setminus vars(process,PId)$ (из конфигурации удалены переменные уничтожаемого экземпляра процесса *process*);
– CHAN.DELETE(*process*): $C2=C1 \setminus channels(process,PId)$ (из конфигурации удалены каналы уничтожаемого экземпляра процесса *process*);
– ACT.DELETE(*process*): $S2=S1 \setminus states(process,PId)$, $T2=T1 \setminus timers(process,PId)$ (из конфигурации удалены значения активности и задержек уничтожаемого экземпляра процесса *process*);
– PROC.DELETE(*process*): $P2(instances(process))=P1(instances(process)) \setminus \{PId\}$ (из конфигурации удалён уничтожаемый экземпляр процесса *process*);
– CHAN.WRT_ALL(*chan,sig,x*): $\forall\ pid \in C2(chan).\ PUT(C1(chan,pid),sig,V1(x))=C2(chan,pid)$.

(сигнал *sig* записан в выходной канал *chan* для всех экземпляров процесса, к которому он направлен);

– CHAN.CLNOUT_ALL(*chan*): ∀ *pid* ∈ *C*2(*chan)*. EMP*C*2(*chan,pid*). (очищен выходной канал *chan* для всех экземпляров процесса, к которому он направлен).

3.3 Порождение экземпляра процесса

Порождение происходит переходом с телом “CREATE PROCESS имя-процесса”.

При этом в специальную переменную OFFSPRING заносится PId (последнего) созданного процесса.

Когда создаётся экземпляр процесса, конфигурация “обогащается” переменными, каналами, состояниями и таймерами, проиндексированными новым PId'ом соответствующего процесса.

Например,

$$V2 = V1 \cup new(vars(process), max(P2(process))$$

означает, что множество переменных во второй конфигурации расширено по сравнению с первой конфигурацией посредством имён переменных процесса process (понимаемого как “тип процесса”), проиндексированных PId'ом соответствующего процесса.

Функция **new(entity, newPId)** является «полиморфной», т.е. она создаёт экземпляр соответствующей сущности (процесса, канала, переменной), индексированный данным новым идентификатором процесса PId. Этот новый PId на единицу больше, чем максимальный PId всех процессов, существовавших до момента создания (включая уже уничтоженные процессы, чтобы избежать возможных коллизий).

Функция *max,* применяемая к множеству чисел имеет обычный смысл — возвращает максимальное значение из множества.

Правило порождения экземпляра процесса

RULE 12 (Creating a process instance)
TIME.CONST, DEL.ZER,
∃ state, process, interval, next :
VAR.CREATE(*process*), CHAN.CREATE(*process*),
ACT.CREATE(*process*), PROC.CAN_CREATE(*process*)
PROC.CREATE(*process*), ACT.ACT(*state*),
ACT.NEXT(*next*), DEL.IN(*state, interval*),
CrTR(state, prog, interval, next)

|=
CNF1<CREATE(process)>CNF2

3.4 Уничтожение экземпляра процесса

Уничтожение происходит переходом с телом STOP. Этот переход находится в диаграмме уничтожаемого процесса.

Процесс может уничтожить другой процесс, только послав ему сигнал, получив который, другой процесс сам закончит работу. Когда процесс уничтожается, переменные, каналы, состояния и таймеры, проиндексированные PId'ом уничтожаемого процесса, удаляются из конфигурации. При этом содержимое выходных каналов уничтоженного процесса становится недоступным для других процессов.

Правило уничтожения экземпляра процесса
RULE 13 (STOP of a process instance)
TIME.CONST, DEL.ZER,
∃ state, process, interval, next :
PROC.DELETE(*process*), ACT.DELETE(*process*),
VAR.DELETE(*process*), CHAN.DELETE(*process*),
ACT.ACT(*state*), ACT.NEXT(*next*),
DEL.IN(state, interval), StTR(state)
|=
CNF1<STOP(*process, PId*)>CNF2.

3.5 Отправка сигнала всем экземплярам процесса через заданный канал

RULE 14 (Writing of a signal to all)
TIME.CONST, DEL.ZER, VAR.CONST, ∃ *state*, *sig*, *x*, *chan*, *interval*, *next* :
CHAN.WR(*chan*, *sig*, *x,pid*), CHAN.STEP(*chan*), ACT.ACT(*state*),
ACT.NEXT(*next*), DEL.IN(*state*,*interval*), WTR(*state, sig, x, chan, interval, next)* |= *CNF*1<WRT_ALL(*chan*,*sig*,*x*)>*CNF*2

3.6 Очистка заданного канала для всех экземпляров процесса

RULE 15 (Cleaning all output channels)
TIME.CONST, DEL.ZER, VAR.CONST, ∃ *state, chan, interval, next* :

CHAN.OUTPUT(*chan,pid*), CHAN.CLEAN(*chan*), CHAN.STEP(*chan,pid*), ACT.ACT(*state*), ACT.NEXT(*next*),

DEL.IN(*state*,*interval*), CTR (*state, sig, x, chan, interval, next*) |= *CNF*1<CLNOUT__ALL >*chanCNF*2

4. ПРЕДСТАВЛЕНИЕ СВОЙСТВ В ЯЗЫКЕ DYNAMIC-REAL.

Языком представления свойств служит подъязык логических спецификаций REAL, который значительно расширяет средства логики ветвящегося времени CTL за счет использования временных интервалов и средств динамических логик.

Формулы этого языка строятся из предикатов с помощью булевских операций, кванторов по выделенным переменным, а также трёх видов модальностей. Первый вид — модальности по моментам времени из мультиинтервала, семантика которых означает «для всех моментов времени» и «существует момент времени». Второй вид — модальности по возможным последовательностям действий (т.е. поведениям) выполнимых спецификаций, семантика которых означает «для всех поведений» и «существует поведение». Кроме того, используется квалификатор (своего рода квантор) по экземплярам процессов: «для всех экземпляров» и «существует экземпляр».

Предикаты бывают четырех видов:

- отношения между значениями переменных и параметров сигналов;
- локаторы управляющих состояний в процессах;
- контроллеры пустоты и переполнения для каналов;
- чекеры наличия/готовности сигналов в каналах.

В отношениях используются расширенные имена переменных и параметров сигналов. В локаторах контроля управления также используются расширенные имена состояний процессов. Контроллеры могут быть контроллерами как пустоты, так и полноты каналов (конечной длины). Диаграмма формулы строится из имен предикатов с помощью пропозициональных связок ($\neg$, &, $\wedge$, $\Rightarrow$, $\Leftrightarrow$), кванторов ($\forall$, $\exists$) над кванторными переменными и идентификаторами процессов, модальностей по моментам времени («всегда» $\Box$ и «иногда» $\Diamond$) и модальностей по поведениям («для всех» EACH и «для некоторых» SOME). Пропозициональные связки, кванторы и скобки используются обычным образом. Динамические и временные мо-

дальности используются в качестве модального префикса только в следующих комбинациях: EACH□ , EACH◊ , SOME□ и SOME◊.

Семантика логических спецификаций будет определяться в терминах истинности в конфигурациях.

Для любой конфигурации CNF и любой логической спецификации SPC тот факт, что конфигурация принадлежит множеству истинности логической спецификации, обозначается CNF |= SPC, а его отрицание — CNF |≠ SPC.

Чтобы сократить описание семантики логических спецификаций, зафиксируем конфигурацию CNF=(*T*,*V*,*C*,*S*=(*SAC*,*STD*)). Отношение CNF |= определяется индукцией по структуре диаграммы логической спецификации SPC.

Базис индукции: SPC — предикат. Если SPC — отношение, то его диаграмма имеет вид R(t1, ..., t2), где R — символ отношения, а t1, ..., t2 — выражения, построенные из символов операций, расширенных имен переменных и параметров. Тогда CNF |= SPC ⇔ значения выражений t1, ..., t2 в конфигурации CNF удовлетворяют отношению, соответствующему R.

Если SPC — локатор, то его диаграмма имеет вид `AT` *state*, где *state* — некоторое состояние или расширенное имя состояния. Тогда CNF |= `AT` *state* ⇔ SAC(*state*)=true.

Если SPC — контроллер, то его диаграмма имеет вид `EMP` *channel* или `FUL` *channel*, где *channel* — некоторый канал или его экземпляр. Тогда, соответственно,

CNF |= `EMP` *channel* ⇔ отношение *EMP*(C(*channel*)) истинно,

CNF |= `FUL` *channel* ⇔ отношение *FUL*(C(*channel*)) истинно.

Если SPC — чекер, то его диаграмма имеет вид *signal* IN *channel* для чекера наличия или *signal* IN *channel* для чекера готовности, где *signal* — некоторый сигнал, а *channel* — некоторый канал. Тогда

CNF |= *signal* IN *channel* ⇔ C(*channel*) содержит вершину, помеченную сигналом *signal* с некоторым значением параметра;

CNF |= *signal* `RD` *channel* ⇔ *GET*(C(*channel*)) = (*state* , *signal* , *vl*) при некоторых значениях *state* и *vl*, т.е. сигнал *signal* доступен для чтения в C(*channel*).

Шаг индукции. Если диаграмма спецификации SPC есть пропозициональная комбинация, то ее значение определяется естественным образом. Например, если диаграмма имеет вид A & B, где A и B — диаграммы формул, то CNF |= SPC ⇔ CNF |= SPA и CNF |= SPB, где SPA

и SPB отличаются от SPC только диаграммами, которые суть A и B соответственно.

Если диаграмма спецификации SPC есть ∀ *variable* A (или ∃ *variable* A), где *variable* — кванторная переменная, а A — диаграмма формулы, то CNF |= SPC ⇔ для каждой (соответственно некоторой) конфигурации CNF', отличающейся от CNF не более чем интерпретацией переменной *variable*, выполняется CNF' |= SPA, где SPA отличается от SPC только диаграммой, которая есть A.

Если диаграмма спецификации SPC есть

M1 множество поведений M2 мультиинтервал A,

где M1 — модальность EACH или SOME, M2 — модальность □ или ◊, а A — диаграмма формулы, то модальность EACH означает «для всех поведений из данного множества поведений», SOME означает «существует поведение в данном множестве поведений», □ означает «для всех моментов времени из данного мультиинтервала», ◊ означает «существует момент времени в данном мультиинтервале».

5. ПРИМЕР СПЕЦИФИКАЦИИ ПРОТОКОЛА УПРАВЛЕНИЯ СЕТЬЮ КАСС-ТЕРМИНАЛОВ

5.1 Описание протокола

Протокол обслуживания пассажиров автоматической сетью кассой состоит в следующем. Сеанс начинается с того, что центральный компьютер (main_machine) включает все терминалы. Затем в очередь к определенному терминалу приходит пассажир, который нажимает кнопку клавиатуры с нужной ему станцией. Получив от пассажира название станции, терминал запрашивает у центрального компьютера стоимость проезда до данной станции и по получению ответа показывает на первом индикаторе цену билета. Далее начинается следующий цикл: пассажир смотрит на второй индикатор и, если на нем оказывается сумма, отличная от стоимости билета, берет монету из кошелька и опускает ее в щель-монетоприемник. Терминал, получив монету, складывает ее номинал с суммой денег, уже введенных пассажиром, и показывает новое значение суммы на втором индикаторе. Если в определенный момент времени пассажир обнаруживает, что у него денег уже не осталось, а цена билета еще больше, чем оказалось денег в

монетоприемник, то пассажир нажимает кнопку return. Терминал, получив команду при нажатии этой кнопки, выдает все ранее введенные деньги в лоток для сдачи и переходит в состояние ожидания следующего в очереди пассажира. Если пассажир в определенный момент времени перестаёт работать с терминалом, запросив билет до станции, но не до конца заплатив за него, то терминал ждет 2 минуты, после чего переходит в состояние ожидания следующего пассажира. Когда второй индикатор показывает сумму, равную цене билета с первого индикатора, пассажир нажимает на клавиатуре кнопку выдачи билета. Терминал, получив такую команду с клавиатуры, отправляет данные о полученной сумме в центральный компьютер и если все верно, печатает билет с указанием станции, а пассажир забирает этот билет в билетном окне и завершает сеанс, освобождая место для следующего пассажира. После выдачи билета терминал завершает сеанс обслуживания пассажира и ждет, пока к нему не подойдет другой пассажир.

Выполнимая спецификация этого протокола на языке dREAL представлена на рисунках 3, 4, 5.1, 5.2, 6.1, 6.2, а спецификация этого протокола на языке SDL представлена в Приложении 4.

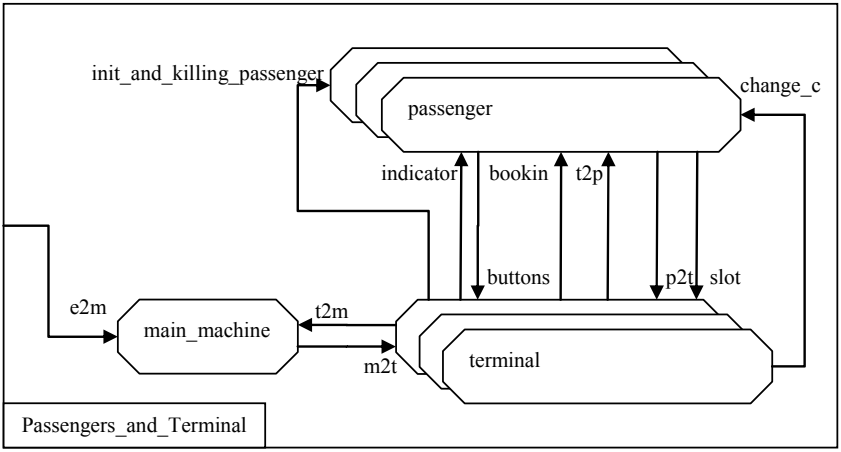


Рис. 3. Взаимодействие процессов в примере «Сеть Касс»

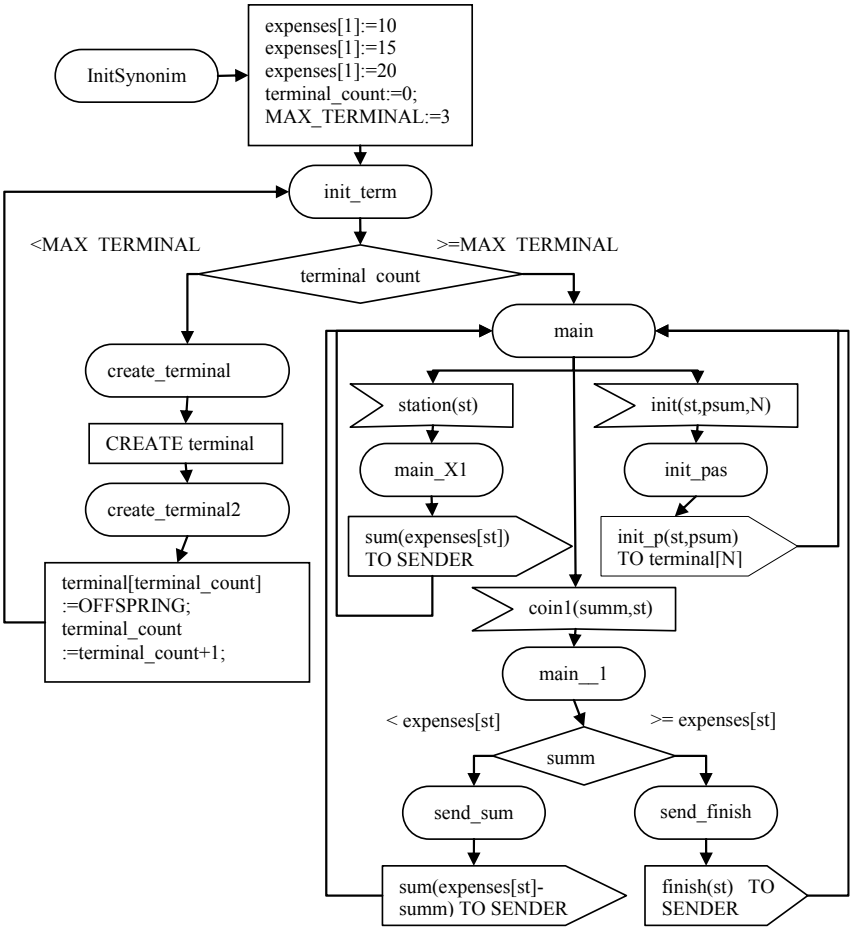


Рис. 4. Блок-схема процесса main-machine

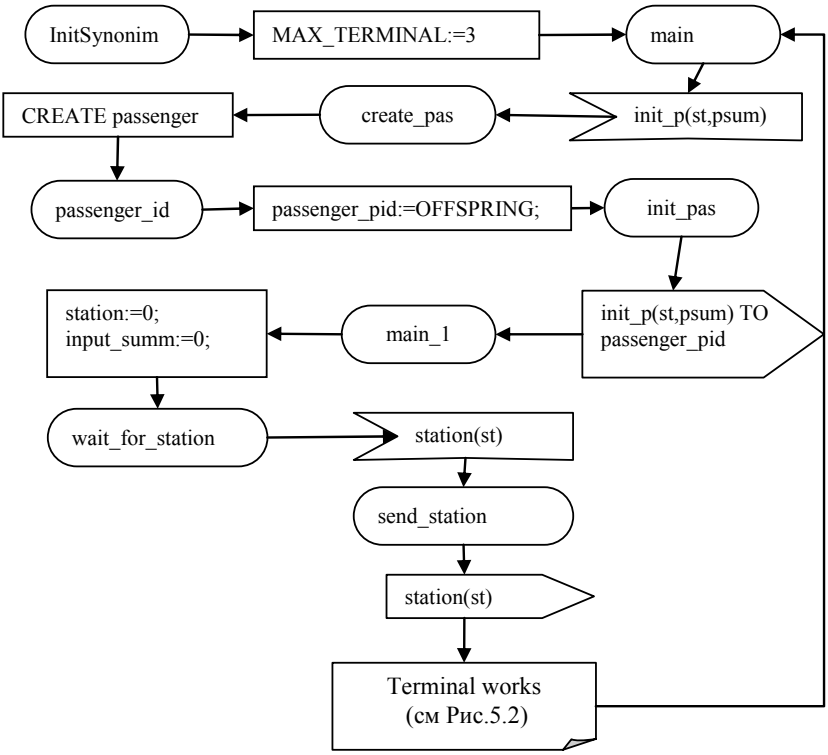


Рис. 5.1. Блок-схема процесса terminal

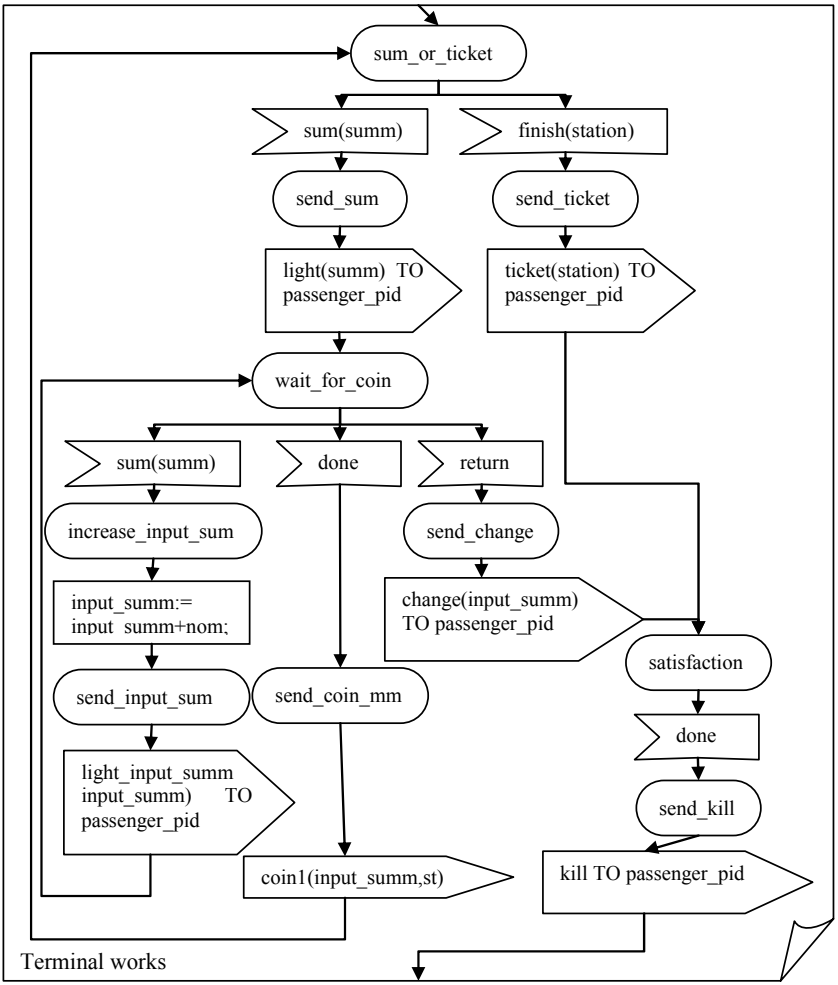


Рис. 5.2. Блок-схема фрагмента процесса terminal

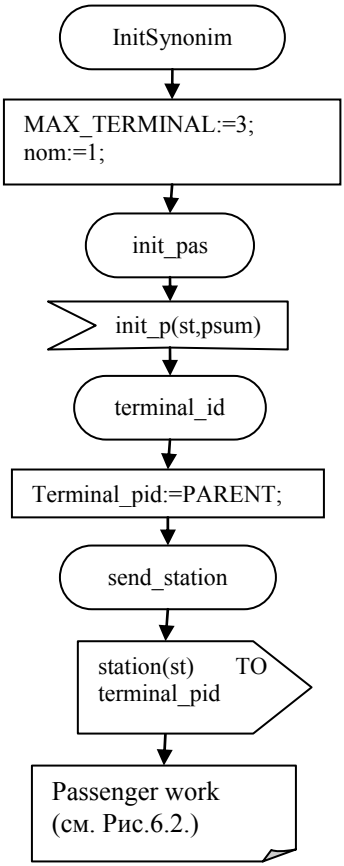


Рис. 6.1. Блок-схема процесса Passenger

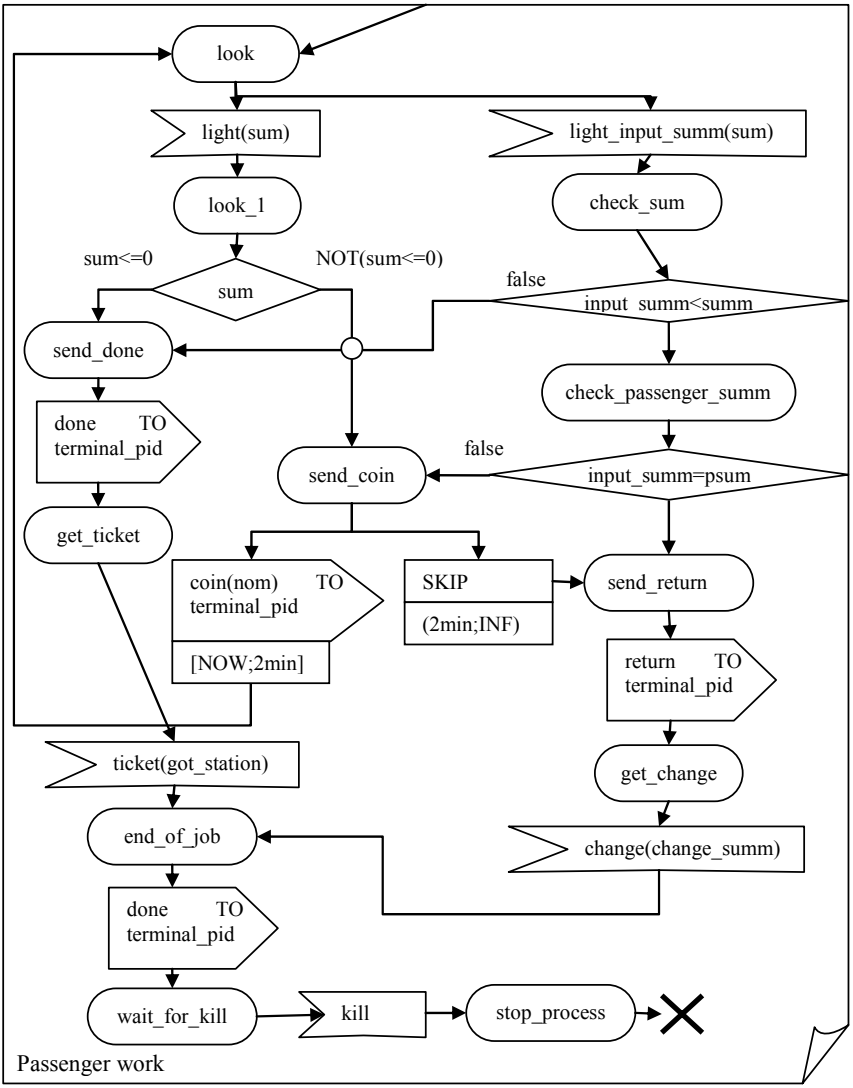


Рис. 6.2. Блок-схема фрагмента процесса Passenger

5.2 Логическая спецификация протокола

Приведенное свойство системы заключается в том, что пассажир получит билет до нужной ему станции в случае, если он действует в соответствии с протоколом, описанным в разделе 5.1.

property : FORM

```
1 min = 60 sec;
1 sec <= 60 sec;
1 ing <= 1 min <= 20 ing;

/*.....контекст спецификации см. в Приложении 3.....*/
/**/
./*свойство спецификации*/
Passengers_and_Terminals
 (AL pid.
(AL terminal[pid].expenses.(AL passenger[pid].station.
((start_of_machine & start_of_passenger &
  no_commands_on_buttons & no_information_on_indicator &
  no_ticket_in_booking & no_money_in_slot)
       =>
(EACH ET (end_of_passenger & get_ticket))))))).
/*описание предикатов*/
/*терминал находится в начальном состоянии*/
start_of_machine : PRED
AT terminal[pid].Start
/*процесс passenger находится в начальном состоянии*/
start_of_passenger : PRED
AT passenger[pid].Start
/*ни одна кнопка не нажата*/
no_commands_on_buttons : PRED
buttons IS EMPTY
/*на индикаторе ничего не отображается*/
no_information_on_indicator : PRED
indicator IS EMPTY
/*лоток выдачи билета пуст*/
no_ticket_in_booking : PRED
booking IS EMPTY
/*в монетоприемнике нет монет*/
```

```
no_money_in_slot : PRED
slot IS EMPTY
/*пассажир завершил работу*/
end_of_passenger : PRED
AT passenger[pid].end_of_process

/*билет до нужной станции получен */
get_ticket : PRED
passenger[pid].st=passenger[pid].station
```

6. ЗАКЛЮЧЕНИЕ

Насколько нам известно, язык Dynamic-REAL является единственным комбинированным языком спецификаций как телекоммуникационных систем, так и их свойств. Преимущества языка dREAL обусловлены следующими характерными чертами:

- простым синтаксисом, допускающим графическое представление выполнимых спецификаций;
- полной компактной операционной семантикой;
- выразительностью языка свойств, которая обеспечивается средствами как временных, так и динамических логик.

Пример, представленный в разд. 5, является оригинальным примером, который иллюстрирует преимущества нашего подхода.

В отличие от языка SDL, в языке выполнимых спецификаций dREAL допускаются недетерминированные действия, и все действия сопровождаются временными интервалами, что позволяет преодолеть известные недостатки концепции таймера в языке SDL.

Предполагается расширить нашу систему SRPV [4] с целью анализа и верификации SDL-спецификаций, использующих динамические конструкции. В качестве промежуточного языка в этой системе SRPV предполагается использовать язык dREAL. Предполагается применить язык dREAL и систему SRPV к спецификации, анализу и верификации телекоммуникационных систем из таких проблемных областей, как коммуникационные протоколы, телефонные сети, системы управления железнодорожным транспортом.

СПИСОК ЛИТЕРАТУРЫ

1. Карабегов А.В., Тер-Микаэлян Т.М. Введение в язык SDL. — М.: Радио и связь. — 1993.
2. Непомнящий В.А., Шилов Н.В., Бодин Е.В Спецификация и верификация распределенных систем средствами языка Elementary-REAL // Программирование. — 1999. — № 4. — С. 54–67.
3. Непомнящий В.А., Шилов Н.В., Бодин Е.В REAL: язык для спецификации и верификации систем реального времени // Системная информатика. — Новосибирск: Наука, 2000. — Т. 7. — С. 174–224.
4. Непомнящий В.А., Бодин Е.В, Веретнов С.О., Тюрюшкин М.В. Симуляция и верификация статических SDL-спецификаций распределенных систем с помощью промежуточного языка REAL. — Новосибирск, 2007. — 69 с. — (Препр. / ИСИ СО РАН; № 142).
5. Broy M. Towards a formal foundation of the specification and description language SDL // Formal Aspects of Computing. — 1991. — Vol. 3, N 1. — P. 21–57.
6. Eschbach R. et al. On the formal semantics of SDL-2000: A compilation approach based on an abstract SDL machine// Lect. Notes Comp. Sci. — 2000. — Vol. 1912. — P. 242–265.
7. Mery D., Mokkedem A. CROCOS: An integrated environment for interactive verification of SDL specifications // Lect. Notes Comp. Sci. — 1993. — Vol. 663. — P. 343–356.
8. Nepomniaschy V.A., Shilov N.V., Bodin E.V., Kozura V.E. Basic-REAL: integrated approach for design, specification and verification of distributed systems // Lect. Notes Comp. Sci. — 2002. — Vol. 2335. — P. 69–88.
9. Orava F. Formal semantics of SDL specifications // Proc. of 8-th IFIP Intern. Symp. on Protocol Spec. Test., and Verif. — 1988. — P. 143-157.
10. Prinz A., Lowis M. Engineering the SDL formal language definition, Proc. FMOODS 2003 // Lect. Notes Comp. Sci. — 2003. — Vol. 2884. — P. 47–63.
11. Specification and Description Language (SDL). — CCITT, Recommendation Z.100, 1988.

ФОРМАЛЬНЫЙ СИНТАКСИС ЯЗЫКА DYNAMIC-REAL

specification:: executional-specification | logical-specification

executional-specification:: process | block

logical-specification:: predicate| formula

process:: process-head scale context fairness-conditions process-diagram

block:: block-head scale context fairness-conditions block-diagram subblocks

predicate:: predicate-head scale context systems predicate-diagram

formula:: formula-head scale context systems formula-diagram subformulae

process-head:: name : `PROCESS` [(init-count, max-count)]

block-head:: name : `BLOCK`

predicate-head:: name : `PREDICATE`

formula-head:: name : `FORMULA`

scale:: {linear-equality-over-time-units | linear-inequality-over-time-units }

context:: {type-definition | object-declaration }

type-definition:: `TYPE` type `IS` type-expression

type-expression:: predefined-type | enumerated-type | type-expression `ARRAY OF` type-expression | type-expression `QUEUE OF` type-expression | type-expression `STACK OF` type-expression | type-expression `BAG OF` type-expression

predefined-type:: `INT` | `STR` | `FLOAT`

enumerated-type::name | name, enumerated-type | name; enumerated-type

subblocks:: { executional-specification }

subformulae:: { logical-specification }

object-declaration:: variable-declaration | channel-declaration

variable-declaration:: location назначение `VAR` variable `OF` type

channel-declaration:: location role organization `CHN` channel `FOR` signal {`WITH PAR` parameter `OF` type-expression }

role:: `INP` | `OUT` | `INN`

organization:: capacity structure | `ELEMENTARY`

capacity:: number-`ELM` | `UNB`

structure:: QUE | STACK | BAG

process-diagram:: { transition }

transition:: TRANSITION state [**:**] [WHEN condition] body interval jump

body:: EXE program | READ signal-with-parameters-1 FROM channel | WRITE signal-with-parameters-2 INTO channel_inst | CLEAN channel_inst |

CREATE [PROCESS] name [ELSE state] | KILL [PROCESS] name [ELSE state] { WRITE Kill INTO channel_inst }

program:: operator, {; operator }

operator:: variable :=expression | SKIP | ABRT | IF condition THEN program [ELSE program] FI | WHILE condition DO program OD | CASE program [ALT program] ESAC | LOOP program POOL

signal-with-parameters-1:: signal [**(** variable-list **)**]

variable-list:: variable {, variable }

signal-with-parameters-2:: signal [**(** value-list **)**]

value-list:: expression {, expression }

expression :: constant | variable | **(** expression **)** | expression operation-sign expression

operation-sign:: **+** | **-** | ***** | **/** | APPLY | UPDATE

interval:: left-bound linear-time-expression right-bound linear-time-expression

left-bound:: AFTER | FROM |

right-bound:: UNTIL | UPTO |

jump:: JUMP state-list.

state-list:: state {, state }.

block-diagram:: { route }

route:: source CHN channel destination

source:: name | ENV

destination:: name | ENV

predicate-diagram:: relation | locator | controller | checker

relation:: expression relation-sign expression

locator:: AT state

controller:: EMP channel_inst | channel_inst IS EMPTY | FUL channel_inst | channel_inst IS OVERFULL

checker:: signal IN channel_inst | signal RD channel_inst

formula-diagram:: name | (propositional-combination) | (quantifier variable formula-diagram) | (behavioural-modality system temporal-modality interval formula-diagram)

quantifier:: ∀ | ∃
behavioural-modality:: EACH | SOME
temporal-modality □ | ◊
state:: name
channel:: name
channel_inst:: channel | channel [process-PId-value]
parameter:: name

Приложение 2

ФОРМАЛЬНАЯ СЕМАНТИКА СТАТИЧЕСКИХ КОНСТРУКЦИЙ ЯЗЫКА DYNAMIC-REAL

Условия

Пусть CNF1 = ($T1$,$V1$,$C1$,$S1$=($ACT1$,$DEL1$)) и
CNF2 = ($T2$,$V2$,$C2$,$S2$=($ACT2$,$DEL2$)) — пара конфигураций.

Для краткости в правилах используются условия общие для нескольких правил.

TIME.CONST $T1=T2$

TIME.STEP $T1<T2$

VAR.CONST $\forall x.V1(x)=V2(x)$.

VAR.STEP(x) $\forall y\neq x.V1(y)=V2(y)$.

VAR.IO(*prog*) $(V1,V2)\in IO(prog)$.

CHAN.CONST $\forall chan.C1(chan)=C2(chan)$.

CHAN.STEP(*chan*) $\forall\ chan' \neq chan: C1(chan')=C2(chan')$.

CHAN.HEAD(*chan*,*sig*,*x*,*pid*) $GET(C1(chan,pid))=(C2(chan,pid),sig,V2(x))$.

CHAN.PUT(*chan*,*pid*) $\exists sig,val.PUT(C1(chan,pid),sig,val)=C2(chan,pid)$.

CHAN.GET(*chan*,*pid*) $\exists sig,val.GET(C1(chan,pid))=C2(chan,sig,val,pid)$.

CHAN.WRT(*chan*,*sig*,*x*,*pid*) $PUT(C1(chan,pid),sig,V1(x))=C2(chan,pid)$.

CHAN.WRT_ALL(*chan*,*sig*,*x*) $\forall\ pid \in C2(chan)$. $PUT(C1(chan,pid),sig,V1(x))=C2(chan,pid)$.

CHAN.CLNIN(*chan*,*pid*) EMP$C2(chan,pid)$.

CHAN.CLNOUT(*chan*,*pid*) EMP$C2(chan,pid)$.

CHAN.CLNOUT_ALL(*chan*) $\forall\ pid \in C2(chan)$. EMP$C2(chan,pid)$.

CHAN.INPUT(*chan*) *chan* — входной;

CHAN.OUTPUT(*chan*) *chan* — выходной;

LEG.R(*sig*,*par*,*chan*,*pid*) Сигнал *sig* с параметром *par* может быть прочитан из канала *chan*.

LEG.W(*sig*,*par*,*chan*,*pid*) Сигнал *sig* с параметром *par* может быть записан в канал *chan*

DEL.CONST ∀*state*.*DEL*1(*state*)=*DEL*2(*state*).

DEL.ZER ∀*state*.*DEL*2(*state*)=0.

DEL.OUT ∀*state*,*transition* если *state*∈*ACT*1 и *state* метит переход *transition*, то *DEL*1(*state*) выходит за правую границу интервала *interval* этого перехода.

DEL.NOT-OUT

∀*state*. если *state*∈*ACT*1, то в диаграмме процесса есть переход state body interval jump такой, что that *DEL*1(*state*) не выходит за правую границу интервала *interval*.

DEL.IN(*state*,*interval*) *DEL*1(*state*)∈*interval*.

DEL.PROGR ∀*state*. if *state*∈*ACT*1, то *DEL*2(*state*)=*DEL*1(*state*)+*T*2−*T*1, иначе *DEL*2(*state*)=0.

ACT.CONST ∀*state*.*state*∈*ACT*1⇔*state*∈*ACT*2.

ACT.ACT(*state*) *state*∈*ACT*1.

ACT.NEXT(*next*) *next*∈*ACT*2.

STATE.FIN ∀*state*. если *state*∈*ACT*1, то в диаграмме процесса нет перехода, помеченного состоянием *state*.

RTR(*state, sig, x, chan, interval, next*)

диаграмма процесса содержит переход *state* `READ` *sig(x)* `FROM` *chan interval* `JUMP` *Set*, где *Set* — множество состояний таких, что *next*∈*Set*.

WTR(*state, sig, x, chan, interval, next*)

диаграмма процесса содержит переход *state* `WRITE` *sig(x)* `INTO` *chan interval* `JUMP` *Set*, где *Set* — множество состояний, таких что *next*∈*Set*.

CTR(*state, sig, x, chan, interval, next*)

диаграмма процесса содержит переход *state* `CLEAN` *chan* `JUMP` *Set*, где *Set* — множество состояний, таких что *next*∈*Set*.

PTR(*state*,*prog*,*interval*,*next*)

диаграмма процесса содержит переход*state* `EXE` *prog interval* `JUMP` *Set*, где *Set* — множество состояний, таких что *next*∈*Set*.

CrTR(*state*,*process*,*interval*,*next*)

диаграмма процесса содержит переход *state* `CREATE` *process interval* `JUMP` *Set*, где *Set* — множество состояний, таких что *next*∈*Set*.

StTR(*state*)

диаграмма процесса содержит переход *state* `STOP`.

VAR.CREATE(*process*) $V2=V1 \cup new(vars(process),max(P2(process))$

CHAN.CREATE(*process*) $C2=C1 \cup new(channels(process),max(P2(process)))$

ACT.CREATE(*process*) $S2=S1 \cup new(states(process),max(P2(process)))$, $T2=T1 \cup new(timers(process),max(P2(process)))$

PROC.CAN_CREATE(*process*) $|P1(process)| < \max\_instances(process)$

PROC.CREATE(*process*) $P2(process)=P1(instances(process)) \cup \{max(P1(process))+1\}$

VAR.DELETE(*process*) $V2=V1 \setminus vars(process,PId)$

CHAN.DELETE(*process*) $C2=C1 \setminus channels(process,PId)$

ACT.DELETE(*process*) $S2=S1 \setminus states(process,PId)$, $T2=T1 \setminus timers(process,PId)$

PROC.DELETE(*process*) $P2(instances(process))=P1(instances(process)) \setminus \{PId\}$

Правило композиции для блока B с подблоками B1...Bk выглядит следующим образом.

RULE 0 (Composition)

For each $i = 1, \ldots, k$, CNF1/B <*EVN*/B > CNF2/B

|=

*CNF*1<*EVN*>*CNF*2.

RULE 1 (Stuttering)

TIME.CONST, VAR.CONST, CHAN.CONST, ACT.CONST, DEL.PROGR, DEL.NOT-OUT |= *CNF*1<INV>*CNF*2

RULE 2 (Stabilization)

TIME.STEP, VAR.CONST, CHAN.CONST, ACT.CONST, DEL.PROGR, STATE.FIN |= *CNF*1<INV>*CNF*2

RULE 3 (Reading of a signal)

TIME.CONST, DEL.ZER, ∃ *state*, *sig*, *x*, *chan*, *interval*, *next*: VAR.STEP(*x*), CHAN.HEAD(*chan*,*sig*,*x*,*pid*), CHAN.STEP(*chan*,*pid*), ACT.ACT(*state*), ACT.NEXT(*next*), DEL.IN(*state*,*interval*), RTR(*state, sig, x, chan, interval, next*) |= *CNF*1<RDN(*chan*,*sig*,*x*)>*CNF*2

RULE 4 (Writing of a signal)

TIME.CONST, DEL.ZER, VAR.CONST, ∃ *state*, *sig*, *x*, *chan*, *interval*, *next* : CHAN.WR(*chan*, *sig*, *x*,*pid*), CHAN.STEP(*chan*,*pid*), ACT.ACT(*state*), ACT.NEXT(*next*), DEL.IN(*state*,*interval*), WTR(*state, sig, x, chan, interval, next)* |= *CNF*1<WRT(*chan*,*sig*,*x*)>*CNF*2

RULE 5 (Appearing of a signal)

TIME.CONST, VAR.CONST, ACT.CONST, DEL.CONST, ∃ *state*, *sig*, *x*, *chan*, *interval*, *next* : CHAN.PUT(*chan,pid*), CHAN.STEP(*chan,pid*), LEG.R(*sig, x, chan,pid*) |= *CNF*1<INV>*CNF*2

RULE 6 (Disappearing of a signal)

TIME.CONST, VAR.CONST, ACT.CONST, DEL.CONST, ∃ *state, sig, chan, interval, next* : CHAN.GET(*chan,pid*), CHAN.STEP(*chan,pid*), LEG.W(*sig, x, chan,pid*) |= *CNF*1<INV>*CNF*2

RULE 7 (Cleaning input channel)

TIME.CONST, DEL.ZER, VAR.CONST, ∃ *state, chan, interval, next* : CHAN.INPUT(*chan,pid*), CHAN.CLEAN(*chan,pid*), CHAN.STEP(*chan,pid*), ACT.ACT(*state*), ACT.NEXT(*next*),

DEL.IN(*state,interval*), CTR(*state, sig, x, chan, interval, next*) |= *CNF*1<CLNIN>*chanCNF*2

RULE 8 (Cleaning output channel)

TIME.CONST, DEL.ZER, VAR.CONST, ∃ *state, chan, interval, next* : CHAN.OUTPUT(*chan,pid*), CHAN.CLEAN(*chan,pid*), CHAN.STEP(*chan,pid*), ACT.ACT(*state*), ACT.NEXT(*next*),

DEL.IN(*state,interval*), CTR(*state, sig, x, chan, interval, next*) |= *CNF*1<CLNOUT>*chanCNF*2

RULE 9 (Program Execution)

TIME.CONST, CHAN.CONST, DEL.ZER, ∃ *state, prog, interval, next* : VAR.IO(*prog*), ACT.ACT(*state*), ACT.NEXT(*next*), DEL.IN(*state, interval*), PTR(*state, prog, interval, next*) |= *CNF*1<*EXE*(*prog*)>*CNF*2

RULE 10 (Clock)

TIME.STEP, VAR.CONST, CHAN.CONST, ACT.CONST, DEL.PROGR, DEL.NOT-OUT |= *CNF*1<INV>*CNF*2

RULE 11 (Starvation)

TIME.STEP, VAR.CONST, CHAN.CONST, ACT.CONST, DEL.PROGR, DEL.OUT |= *CNF*1<INV>*CNF*2

Приложение 3

КОНТЕКСТ СПЕЦИФИКАЦИИ ПРИМЕРА «СЕТЬ КАСС»

```
    All: BLOCK
     Passengers_and_Terminals: BLOCK
    /*Число терминалов*/
      PR VAR MAX_TERMINAL OF integer;

    /*канал от main_machine к терминалу*/
      INN UNB QUEUE CHN m2t
    /*сигнал, по которому терминал выдает билет пасса-
жиру*/
    /*параметр1: номер станции до которой выдается би-
лет*/
        FOR finish
        WITH PAR p1 OF integer;
    /*сигнал, сообщающий стоимость билета*/
    /*параметр1: цена билета*/
        FOR sum
        WITH PAR p1 OF integer;
    /*сигнал инициализации пассажира*/
    /*параметр1: номер станции, до которой пассажир
намерен взять билет*/
    /*параметр2: количество денег в кошельке пассажи-
ра*/
        FOR init_p
        WITH PAR p1 OF integer,
        WITH PAR p2 OF integer.

    /*канал от терминала к main_machine*/
      INN UNB QUEUE CHN t2m
    /*сигнал, передающий main_machine информацию,
сколько денег ввел пассажир*/
    /*параметр1: сумма денег*/
    /*параметр2: номер станции*/
        FOR coin1
        WITH PAR p1 OF integer,
        WITH PAR p2 OF integer;
    /*сигнал с номером станции, до которой пассажир
намерен взять билет*/
```

```
    /*параметр1: номер станции*/
        FOR station
        WITH PAR p1 OF integer.

    /*канал от пассажира к терминалу*/
      INN UNB QUEUE CHN p2t
    /*сигнал успешного завершения работы пассажира*/
       FOR done.

    /*канал для выдачи сдачи или денег обратно пассажиру*/
      INN UNB QUEUE CHN change_c
    /*сигнал выдачи сдачи*/
    /*параметр1: сумма выдаваемых денег*/
       FOR change
       WITH PAR p1 OF integer.

    /*канал для выдачи билета пассажиру*/
      INN UNB QUEUE CHN booking
    /*сигнал выдачи билета*/
    /*параметр1: номер станции, до которой выдается билет*/
       FOR ticket
       WITH PAR p1 OF integer.

    /*канал для передачи информации при нажатии кнопок на терминале*/
      INN UNB QUEUE CHN buttons
    /*сигнал, требующий вернуть все введенные деньги*/
       FOR return;
    /*сигнал с номером станции, до которой пассажир намерен взять билет*/
    /*параметр1: номер станции*/
       FOR station
       WITH PAR p1 OF integer.

    /*канал монетоприемника*/
      INN UNB QUEUE CHN slot
    /*сигнал, передающий сумму, которую пассажир положил в монетоприемник*/
    /*параметр1: сумма*/
       FOR coin
       WITH PAR p1 OF integer.
```

```
    /*канал индикатора*/
      INN UNB QUEUE CHN indicator
    /*индикатор, показывающий уже введенную сумму */
    /*параметр1: сумма*/
        FOR light_input_summ
        WITH PAR p1 OF integer;
    /*индикатор, показывающий стоимость билета*/
    /*параметр1: сумма*/
        FOR light
        WITH PAR p1 OF integer.

    /*канал для инициализации и деинициализации пасса-
жира*/
      INN UNB QUEUE CHN init_and_killing_passenger
    /*сигнал инициализации пассажира*/
    /*параметр1: номер станции, до которой пассажир
намерен взять билет*/
    /*параметр2:количество денег в кошельке пассажи-
ра*/
       FOR init_p
       WITH PAR p1 OF integer,
       WITH PAR p2 OF integer.
    /*сигнал деинициализации пассажира*/
       FOR kill.

    /*канал из внешней среды к main-machine*/
      INP UNB QUEUE CHN e2m
    /*сигнал, несущий информацию о подошедшем пассажи-
ре*/
    /*параметр1: номер станции, до которой пассажир
намерен взять билет*/
    /*параметр2: количество денег, которыми располага-
ет пассажир*/
    /*параметр3: номер терминала, к которому пассажир
подошел*/
       FOR init
       WITH PAR p1 OF integer,
       WITH PAR p2 OF integer,
       WITH PAR p3 OF integer.

       FROM main_machine CHN m2t TO terminal.
       FROM terminal CHN t2m  TO main_machine.
```

```
       FROM terminal CHN t2p  TO passenger.
       FROM passenger CHN p2t TO terminal.
       FROM terminal CHN change_c  TO passenger.
       FROM terminal CHN booking  TO passenger.
       FROM passenger CHN buttons TO terminal.
       FROM passenger CHN slot TO terminal.
       FROM terminal CHN indicator  TO passenger.
       FROM  terminal  CHN  init_and_killing_passenger
TO passenger.
       FROM ENV CHN e2m  TO main_machine.
    /*.............описание                      процес-
сов.................*/
    END; /* All */
```

Приложение 4

ПРИМЕР «СЕТЬ КАСС» НА ЯЗЫКЕ SDL

```
SYSTEM All;
BLOCK Passen-
gers_and_Terminals;
SYNONYM MAX_TERMINAL inte-
ger= 3;
SIGNAL
station(integer),
coin(integer),
coin1(integer, integer),
done,
sum(integer),
finish(integer),
kill,
init(integer, integer,
integer),
light(integer),
light_input_summ(integer)
,
ticket(integer),
ready_for_job,
passenger_id(PId),
return,
change(integer),
press_start(integer);

SIGNALROUTE e2q
    FROM ENV TO queue
WITH init;

SIGNALROUTE
init_and_killing_passeng
er
    FROM queue TO pas-
senger WITH init;
    FROM passenger TO
queue WITH kill;

SIGNALROUTE start-
ing_data
    FROM passenger TO
dispatcher WITH
press_start;

SIGNALROUTE indicator
```

```
    FROM terminal TO pas-
senger WITH light,
light_input_summ;

SIGNALROUTE slot
    FROM passenger TO ter-
minal WITH coin;

SIGNALROUTE buttons
    FROM passenger TO ter-
minal      WITH      sta-
tion,return;

SIGNALROUTE booking
    FROM terminal TO pas-
senger WITH ticket;

SIGNALROUTE change_c
    FROM terminal TO pas-
senger WITH change;

SIGNALROUTE d2t
    FROM   dispatcher   TO
terminal   WITH   passen-
ger_id;

SIGNALROUTE p2t
    FROM passenger TO ter-
minal WITH done;

SIGNALROUTE t2p
    FROM terminal TO pas-
senger                WITH
ready_for_job;

SIGNALROUTE t2m
    FROM    terminal    TO
main_machine   WITH   sta-
tion,coin1;

SIGNALROUTE m2t
    FROM main_machine TO
terminal WITH sum, fin-
ish;

PROCESS queue     REFER-
ENCED;
PROCESS     main_machine
    REFERENCED;
PROCESS       dispatcher
    REFERENCED;
PROCESS   passenger(0,3)
    REFERENCED;
PROCESS    terminal(0,3)
    REFERENCED;

ENDBLOCK         Passen-
gers_and_Terminals;
ENDSYSTEM All;

PROCESS queue;
DCL
st,psum,N,count integer;

START;
    TASK count:=0;
    NEXTSTATE look;

STATE look;
    INPUT
init(st,N,psum);
    CREATE passenger;
    OUTPUT
init(st,N,psum)  TO  OFF-
SPRING;
    TASK count:=count+1;
    NEXTSTATE look;

    INPUT kill;
    TASK count:=count-1;
    NEXTSTATE look;
ENDSTATE;
ENDPROCESS queue;
```

```
PROCESS passenger(0,3);
DCL
N,    psum,    summ,    in-
put_summ,     nom,     st,
change_summ,       station
integer,
terminal_pid   PId;
START;
    TASK nom:= 1;
    NEXTSTATE init_pas;

STATE init_pas;
    INPUT init(st,N);
    OUTPUT press_start(N);
    NEXTSTATE look;
ENDSTATE;

STATE look;
    INPUT ready_for_job;
    TASK            termi-
nal_pid:=SENDER;
    OUTPUT  station(st)  TO
terminal_pid;

    INPUT light(summ);
DECISION summ;
(<=0):   OUTPUT   done   TO
terminal_pid;
     NEXTSTATE get_ticket;
(>psum): OUTPUT return;
     NEXTSTATE get_change;
ELSE: OUTPUT coin(nom) TO
terminal_pid;
     NEXTSTATE look;
ENDDECISION;



    INPUT light_input_summ
(input_summ);
DECISION input_summ;
(<summ):
    DECISION input_summ;
    (=psum):OUTPUT return;
            NEXTSTATE
get_change;
    ELSE:OUTPUT
coin(nom);
         NEXTSTATE look;
    ENDDECISION;
ELSE:  OUTPUT  done  TO
terminal_pid;
      NEXTSTATE
get_ticket;
ENDDECISION;
ENDSTATE;

STATE get_change;
    INPUT          change
(change_summ);
    OUTPUT done;
    OUTPUT kill;
    STOP;
ENDSTATE;

STATE get_ticket;
    INPUT
ticket(station)
    OUTPUT  done  TO  ter-
minal_pid;
    OUTPUT kill;
    STOP;
ENDSTATE;

ENDPROCESS passenger;



PROCESS dispatcher;
NEWTYPE arr
    array (integer, PId)
ENDNEWTYPE arr;

DCL
N integer,
terminal_count integer,
terminal arr;
```

```
START;
    TASK            termi-
nal_count:=0;
1:
DECISION terminal_count;
(<MAX_TERMINAL):
    CREATE terminal;
    TASK            termi-
nal(terminal_count):=
OFFSPRING,
    termnal_count:= termi-
nal_count+1;
    JUMP 1;
ELSE: NEXTSTATE main;
ENDDECISION;
ENDSTATE;

STATE main;
    INPUT press_start(N);
    OUTPUT    passenger_id
(SENDER) TO terminal(N);
    NEXTSTATE main;
ENDSTATE;
ENDPROCESS dispatcher;

PROCESS terminal(0,3);
DCL
nom,st,station,summ    in-
teger,
passenger_pid PId;

START;
    NEXTSTATE main;

STATE main;
    INPUT     passenger_id
(passenger_pid);
    TASK station:=0,
         input_summ:=0;
    OUTPUT   ready_for_job
TO passenger_pid;
    NEXTSTATE
wait_for_station;
    ENDSTATE;
STATE wait_for_station;
    INPUT station(st);
    OUTPUT station(st);
    NEXTSTATE
sum_or_ticket;
ENDSTATE;

STATE sum_or_ticket;
    INPUT sum(summ);
    OUTPUT    light(summ)
TO passenger_pid;
    NEXTSTATE
wait_for_coin;

    INPUT            fin-
ish(station);
    OUTPUT
ticket(station)  TO  pas-
senger_pid;
    NEXTSTATE   satisfac-
tion;
ENDSTATE;

STATE wait_for_coin;
    INPUT coin(nom);
    TASK     input_summ:=
input_summ+nom;
    OUTPUT
light_input_summ(input_s
umm) TO passenger_pid;
    NEXTSTATE
wait_for_coin;

    INPUT done;
     OUTPUT
coin1(input_summ,st);
     NEXTSTATE
sum_or_ticket;

     INPUT return;
     OUTPUT
change(input_summ);
```

```
    NEXTSTATE   satisfac-
tion;
ENDSTATE;

STATE satisfaction;
    INPUT done;
    NEXTSTATE main;
ENDSTATE;
ENDPROCESS terminal;

PROCESS main_machine;
NEWTYPE int_arr
    array  (integer,  inte-
ger)
ENDNEWTYPE int_arr;

DCL
st, summ    integer,
expences       int_arr,

START;
    TASK
     expences(1):=10,
     expences(2) := 15,
     expences(3) := 20;
     NEXTSTATE main;

STATE main;
    INPUT station(st);
    OUTPUT
sum(expenses(st))      TO
SENDER;
    NEXTSTATE main;
    INPUT
coin1(summ,st);
DECISION summ;
(<expenses(st)):  OUTPUT
sum(expenses(st)-summ)
TO SENDER;
    NEXTSTATE main;
ELSE:  OUTPUT  finish(st)
TO SENDER;
       NEXTSTATE main;
ENDDECISION;
ENDSTATE;
ENDPROCESS main_machine;
```

В.А. Непомнящий, Е.В. Бодин, С.О. Веретнов

ЯЗЫК СПЕЦИФИКАЦИЙ РАСПРЕДЕЛЕННЫХ СИСТЕМ DYNAMIC-REAL

Препринт
147

Рукопись поступила в редакцию 22.12.07
Рецензент Н. В. Шилов
Редактор Т. М. Бульонкова

Подписано в печать 28.12.07
Формат бумаги 60 × 84 1/16 Объем 2.5 уч.-изд.л., 2.9 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35