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

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

ПРИМЕНЕНИЕ ЯЗЫКА DYNAMIC-REAL ДЛЯ АНАЛИЗА И ВЕРИФИКАЦИИ РАСПРЕДЕЛЕННЫХ СИСТЕМ, СПЕЦИФИЦИРОВАННЫХ НА ЯЗЫКЕ SDL

Препринт
161

Новосибирск 2011

Описана новая версия программного комплекса SRDSV2 (SDL/REAL Distributed Systems Verifier), предназначенного для моделирования, анализа и верификации распределенных систем, специфицированных на стандартном языке SDL. Этот комплекс включает транслятор из языка SDL в язык Dynamic-REAL (dREAL), систему автоматического моделирования dREAL-спецификаций и транслятор из языка dREAL в язык Promela, который является входным языком известной системы верификации методом проверки моделей SPIN. В этой версии программного комплекса SRDSV2 язык dREAL был расширен за счет введения процедур, что позволило уменьшить размер dREAL-спецификаций и их моделей в языке Promela. Описано применение этого комплекса для анализа и верификации динамической системы управления сетью касс-терминалов.

Эта работа частично поддержана грантом РФФИ № 11-07-90412-Ukr_f_a.

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

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

APPLICATION OF DYNAMIC-REAL LANGUAGE FOR ANALYSIS AND VERIFICATION OF DISTRIBUTED SYSTEMS SPECIFIED IN THE LANGUAGE SDL

Preprint
161

Novosibirsk 2011

A new version of the program complex SRDSV2 (SDL/REAL Distributed Systems Verifier) intended for modeling, analysis and verification of SDL-specifications of distributed systems is described. This complex includes a translator from the SDL language into the Dynamic-REAL language (dREAL), a tool for automatic modeling of dREAL- specifications and a translator from dREAL into the input language Promela of the SPIN verifier. In the SRDSV2 version dREAL has been extended by introducing procedures. This extension allows to reduce the amount of dREAL- specifications and their models in Promela. An application of the SRDSV2 tool-set to analysis and verification of a dynamic system for control of booking terminals net is described.

1. ВВЕДЕНИЕ

На практике широко используется стандартный язык выполнимых спецификаций SDL [1,17], предназначенный для разработки распределенных и телекоммуникационных систем. Этот язык часто используется для разработки современных коммуникационных протоколов большого размера [9,15]. Верификация выполнимых спецификаций, представленных на языке SDL, заключается в проверке корректности их ключевых свойств и является актуальной задачей современного программирования. Для доказательства свойств SDL-спецификаций применяется метод проверки моделей [2,3]. Однако непосредственное применение этого метода часто связано со значительными трудностями ввиду громоздкости моделей SDL-спецификаций. Для преодоления этих трудностей используется трансляция SDL-спецификаций в модельный язык IF [11]. С помощью языка IF была проведена верификация используемого на практике коммуникационного протокола Mascara [10,16].

Идея нашего подхода к проблеме верификации SDL-спецификаций состоит в разработке модельных языков, ориентированных на верификацию, которые были бы комбинированными, т.е. включали бы подъязыки выполнимых и логических спецификаций. Такой комбинированный язык спецификаций Basic-REAL (bREAL), базирующийся на статическом подмножестве SDL, был представлен в [8,14], где описана полная структурная операционная семантика выполнимых спецификаций в виде систем переходов, которая позволила доказать важные семантические свойства. Упрощенные версии этого языка, названные Real92 и Elementary-REAL, представлены в [6] и [7]. В [4] описан язык Dynamic-REAL (dREAL), полученный расширением языка bREAL посредством динамических конструкций порождения и уничтожения экземпляров процессов. В [5] описан программный комплекс SRDSV (SDL/REAL Distributed Systems Verifier), предназначенный для моделирования, анализа и верификации SDL-спецификаций распределенных систем, который использует язык выполнимых спецификаций dREAL в качестве промежуточного языка.

Цель настоящей работы – описать новую версию SRDSV2 программного комплекса SRDSV и эксперименты с ней. В этой версии программного комплекса SRDSV2 язык dREAL был расширен за счет введения процедур, что позволило уменьшить размеры dREAL-спецификаций и их моделей в языке Promela. Также в этой версии SRDSV2 устранены коллизии, связан-

ные с различными механизмами передачи сигналов между процессами в языках SDL и dREAL. По сравнению с [5] в данной работе описаны эксперименты с новой версией динамической системы управления сетью касс-терминалов.

Данная работа состоит из шести разделов. В разделе 2 дается обзор новой версии языка dREAL. Раздел 3 посвящен описанию программного комплекса SRDSV2. В разделе 4 представлена SDL-спецификация динамической системы управления сетью касс-терминалов. В разделе 5 описано применение программного комплекса SRDSV2 к моделированию и верификации этой динамической системы. В разделе 6 обсуждаются достоинства программного комплекса SRDSV2 и перспективы развития нашего подхода.

2. ЯЗЫК DYNAMIC-REAL

Язык Dynamic-REAL (dREAL) состоит из языка выполнимых спецификаций для представления распределённых систем и языка логических спецификаций для представления их свойств. Будем обозначать язык выполнимых спецификаций через dREAL.ex, а язык логических спецификаций – через dREAL.log.

2.1. Язык выполнимых спецификаций

Для описания систем в языке dREAL.ex определены две синтаксические формы текстовая и графическая. Система состоит из блоков, соединенных между собой и с окружающей средой каналами. Средствами языка dREAL обеспечивается многоуровневое описание системы. Блок может содержать другие блоки и процессы, которые работают параллельно и взаимодействуют друг с другом и с внешней средой посредством обмена сигналами через каналы. С каждым действием процесса ассоциируется временной интервал, который задает продолжительность выполнения действия. В отличие от [4], мы используем единственную единицу времени, называемую *tick*. Таким образом, временной интервал содержит, кроме **`NOW`** (“сейчас”) и **`INF`** («не ограничен справа»), только время, выраженное в этих единицах. Также добавлена возможность использования специальной служебной переменной **`TICK`** (значение глобальных часов) в выражениях.

Каждый экземпляр процесса может порождать или уничтожать экземпляры процессов в своем блоке. В отличие от [4], уничтожение экземпляра

процесса осуществляется посылкой специального сигнала Kill в соответствующий канал, т.е. тело перехода имеет вид **WRITE Kill INTO** *имя_канала* **[pid]**, где **pid** – уникальный идентификатор экземпляра процесса. При этом сигналы, посланные уничтожаемым экземпляром процесса, не уничтожаются и могут быть прочитаны получателем сигналов, т.е. остаются в его входном канале (если сам получатель не был уничтожен). В то же время сигналы, адресованные уничтожаемому экземпляру, пропадают.

Процессы, имеющие экземпляры, не могут выполнять операцию **CLEAN** (очистку выходного канала), так как это может приводить к уничтожению сигналов, посланных другим экземпляром этого процесса.

Спецификация имеет иерархическую структуру и состоит из заголовка, контекста, диаграммы и подспецификаций. Заголовок определяет имя и вид спецификации: блок или процесс. Контекст — множество определений типов и описаний переменных и каналов.

Диаграмма блока состоит из маршрутов, соединяющих подблоки между собой и с внешней средой. Диаграмма процесса состоит из переходов, каждый из которых состоит из имени состояния, тела, временного интервала и списка (возможно, пустого) следующих состояний.

Имя состояния служит меткой, причём одно состояние может метить несколько переходов.

Также на переход могут накладываться условия выполнения (**WHEN**), такие как проверка наличия сигнала в канале для чтения сигнала, непустота канала для записи сигнала, логическое выражение, использующее значения переменных.

Возможные варианты тела перехода:

1. **READ** – чтение сигнала (с параметрами) из канала.
2. **WRITE** – запись сигнала (с параметрами) в канал.
3. **CLEAN** – очистка выходного канала.
4. **EXE** – исполнение программы, изменяющей значения переменных, хотя содержимое каналов при этом не меняется (подробно понятие программы описано в [4]).
5. **STOP** – прекращение работы данного процесса или его экземпляра.

По сравнению с [4] в язык выполнимых спецификаций dREAL были добавлены следующие операторы для отладки и тестирования спецификаций:

- ASSERT (*условие*) – это условие должно быть выполнено в данном месте спецификации, а при невыполнении условия во время отладки происходит прерывание;

- PRINT(*"строка-сообщение"*,список_*выражений*) – при отладке пользователю выдаётся эта ***строка-сообщение*** (с указанием номера строки исходной спецификации), а если **список_*выражений*** не пуст, то строка считается форматной строкой (в стиле функции **printf** языка С), причём выражения могут содержать переменные, включая переменную **TICK**.

В спецификации оператор **ASSERT(**условие**)** может входить только в переход следующего вида:

состояние:
ASSERT(условие)
тело_перехода
интервал
JUMP ***список-состояний*****.**

Этот переход означает, что при невыполнении этого условия исполнение процесса прерывается.

Эта версия языка dREAL расширена за счёт добавления макроопределений-процедур, не содержащих обращений к каналам.

PROCEDURE *имя* **(** *список_переменных* **);**
(LOCAL VAR *список_переменных***;)***
программа
END PROCEDURE.

Здесь конструкция «*()**» означает повторение 0 и более раз.

Процедура может быть вызвана из программы, которая находится в теле перехода вида **EXE.**

Вызов процедуры осуществляется посредством оператора **CALL** *имя* **(***список_выражений***).**

2.2. Язык логических спецификаций

Подъязык логических спецификаций dREAL (dREAL.log) расширяет конструкции логики ветвящегося времени CTL [2,3] за счет использования временных интервалов и средств динамических логик. Формулы этого языка строятся из предикатов с помощью булевских операций, модальностей по моментам времени из интервала и по возможным поведениям выполнимых спецификаций, а также кванторов (существования и всеобщности) по выделенным переменным и по экземплярам процессов. Предикаты бывают четырех видов: отношения между значениями переменных и параметров сигналов, локаторы управляющих состояний в процессах, контроллеры

пустоты и переполнения для каналов, индикаторы наличия и готовности сигналов в каналах.

3. ПРОГРАММНЫЙ КОМПЛЕКС SRDSV2

3.1. Общая схема программного комплекса SRDSV2

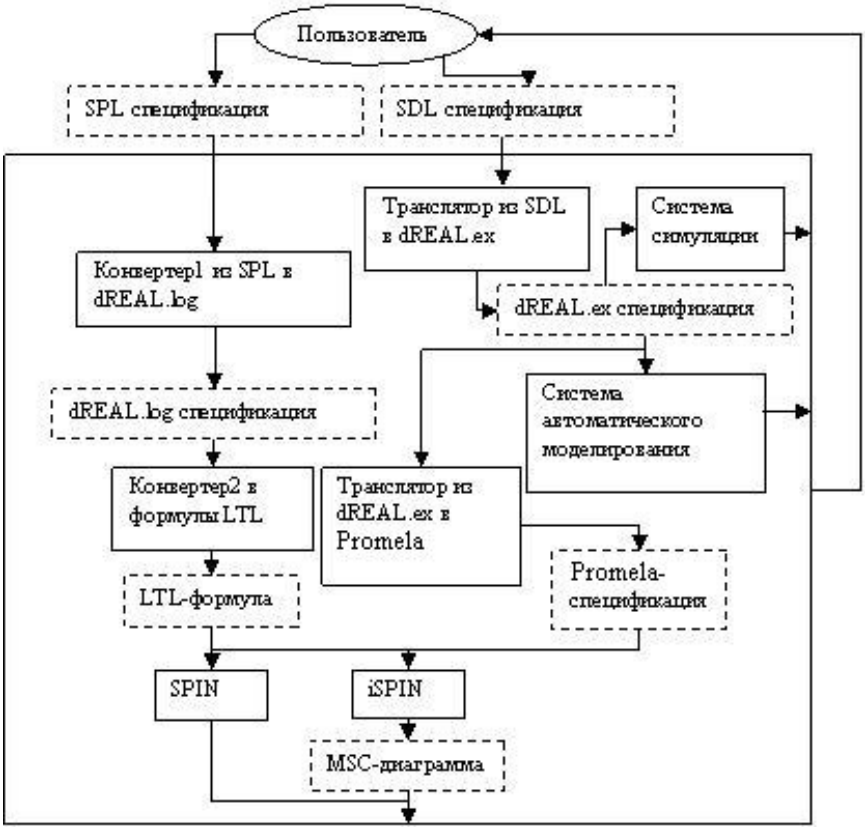

Рис.1. Программный комплекс SRDSV2

Программный комплекс SRDSV2 предназначен для моделирования и верификации спецификаций, представленных на языках SDL и dREAL.ex. Для представления свойств SDL-спецификаций был введен язык SPL (SDL Property Language), близкий к подъязыку логических спецификаций dREAL.log [5]. Единственное различие состоит в том, что в языке SPL кванторы и модальности применяются к экземплярам SDL-процессов и их поведениям. На рис. 1 представлена схема программного комплекса, где прямоугольники со сплошной границей означают компоненты системы, а со штриховой – файлы данных.

Комплекс SRDSV2 состоит из следующих компонентов:

1. Транслятор из языка SDL в язык dREAL переводит SDL-спецификацию в эквивалентную dREAL.ex-спецификацию.
2. Конвертер1 переводит SPL-спецификацию, описывающую свойства SDL-спецификации, в эквивалентную dREAL.log-спецификАцию.
3. Система симуляции позволяет визуально проследить пошаговое выполнение процессов dREAL.ex-спецификации.
4. Система автоматического моделирования позволяет проводить анализ dREAL.ex -спецификации с помощью пользовательских запросов [5].
5. Конвертер2 переводит dREAL.log-спецификации в эквивалентные формулы логики линейного времени LTL [2,3], если это возможно.
6. Транслятор из языка dREAL.ex во входной язык Promela системы SPIN [2,13] переводит dREAL.ex-спецификацию в спецификацию на языке Promela.
7. Система SPIN [2,13] проводит верификацию спецификации на языке Promela относительно свойств на языке логики линейного времени LTL методом проверки моделей [2,3]. Система iSPIN [13] используется для графической симуляции Promela-спецификации и построения соответствующих этой спецификации MSC-диаграмм.

3.2. Транслятор из языка SDL в язык dREAL-ex

Транслируемое подмножество языка SDL включает в себя основные конструкции языка SDL88, такие как BLOCK, PROCESS, SIGNALROUTE, CHANNEL, STATE и другие.

3.2.1. Общая схема транслятора

Процесс трансляции проходит в две основных стадии:

1. Генерация внутреннего представления, соответствующего тексту входного файла. Во время этого этапа происходит проверка входного текста, что он является синтаксически корректной SDL-спецификацией. Результатом является внутреннее представление, приближенное по структуре к dREAL-спецификации.
2. Генерация dREAL-текста (выходного файла), соответствующего внутреннему представлению. По заданным шаблонам происходит построение dREAL-текста.

Результатом работы транслятора является dREAL-текст, соответствующий входной SDL-спецификации, если она не содержала синтаксических ошибок и представима в языке dREAL. Также выдаются сообщения об ошибках, показывающие причину, по которой построение выходного файла провести не удалось.

Опишем внутреннее представление SDL спецификации.

Объект самого высокого уровня – это служебный «глобальный» блок. С точки зрения внутреннего представления он является обычным SDL-блоком и содержит предопределенные объекты с глобальной областью видимости такие, как например предопределенные типы данных.

Непосредственным потомком этого блока является *блок-система*, потомками которой являются блоки и далее – процессы. Таким образом, иерархия внутреннего представления практически полностью совпадает с иерархией языка SDL. С точки зрения внутреннего представления процесс это блок, имеющий граф состояний.

Атрибутами блока являются следующие компоненты:

- список типов. Типы «глобального» блока являются предопределенными, а их определение задается транслятором до начала работы;
- списки переменных, констант;
- список сигналов и список списков сигналов;
- список каналов и временный список соединений.

Процессы содержат информацию о графе состояний и списке условий, если таковые имеются. При трансляции может использоваться ряд служебных свойств, как например список сигналов, которые может получать процесс.

3.2.2. Построение dREAL-спецификации

Блоки/процессы, типы данных и переменные языка SDL имеют прямые аналоги в dREAL, поэтому эта часть процесса трансляции не представляет сложности. Трансляция динамических конструкций и таймеров приведена в [5].

Каналы

Построение ***каналов*** происходит в два этапа: сначала создаются списки ***каналов*** и ***соединений***, в которых запоминается декларативная информация. Затем, после построения дерева блоков, происходит обработка ***соединений***: проверяется непротиворечивость указанной в них информации и связывания их с конкретными объектами внутреннего представления (блоками/процессами или каналами). Это необходимо, так как SDL допускает использование объектов до их определения.

Семантика каналов в SDL и dREAL имеет ряд существенных отличий: SDL-каналы могут разветвляться или сходиться на границе блоков, в то время как dREAL-каналы присоединяются строго один к одному. Двусторонние каналы преобразуются в два с множеством сигналов, которые передаются в одном и в противоположном направлении. Возможность разветвления SDL-каналов преодолевается следующим образом: каналы, которые разделяются или соединятся в один, преобразуются таким образом, чтобы образовавшиеся в результате этой операции каналы были односторонними.

Одним из основных отличий языка SDL от языка dREAL является различный механизм передачи сигналов от процесса к процессу. В языке SDL все входные сигналы хранятся в порядке их прихода в едином порту процесса. В языке dREAL сигналы, относящиеся к разным каналам одного процесса, хранятся независимо от других каналов, так что порядок их чтения не зависит от порядка получения сигналов в других каналах. То есть становится возможной ситуация, что если один процесс отправил другому два сигнала по разным каналам, то пришедший вторым сигнал будет прочитан раньше, чем первый (если их можно прочитать в одном состоянии) и выполнение процесса пойдет в отличном направлении, чем пошло бы в SDL. Для разрешения этой коллизии были предложены следующие методы:

1. Все каналы от одного процесса к другому объединяются в один канал. Это сделано из следующих соображений. В языке dREAL чтение сигналов из каналов происходит недетерминировано, т.е. если в двух каналах в данный момент времени находятся сигналы, то они воспринимаются в случайном порядке. Когда эти сигналы идут по каналам от разных процессов, то так и должно быть. Но если каналы от

одного процесса, то может возникнуть коллизия, если запись сигналов в каналы происходит мгновенно. Первый сигнал, записанный в один сигнал, может восприняться как второй. Когда сигналы исходят из одного процесса, то их порядок может быть важен. Введение единого канала от процесса к процессу исключит такую неоднозначность.

2. В каждом блоке создается процесс-буфер. Для каждого процесса все входные каналы от остальных процессов к этому перенаправляются от этих процессов к буферу. Буфер читает приходящие сигналы и записывает их в единый канал, ведущий к рассматриваемому процессу. Таким образом получается, что в единственном входном канале процесса все входящие сигналы расположены в порядке их прихода в буфер. В случае одновременного прихода сигналы записываются в случайном порядке, как и в SDL.

На рис. 2 и рис. 3 представлен механизм передачи сигналов до и после трансляции. Как видно из схем, в процессе трансляции добавляется служебный процесс буфер. Каналы C1 и C2 переходят в связку каналов P1toP3_ch от процесса P1 до буфера и buffertoP3_ch от буфера к процессу P3. Двусторонний канал C3 переходит в связки каналов: P2toP3_ch + buffertoP3_ch и P2toP3_ch_inv+buffertoP2_ch. Каналы C5 и C6 переходят в каналы P2toP1_ch и buffertoP1_ch. Заметим, что от буфера к каждому процессу идет ровно по одному каналу, каждый из которых может передавать все сигналы, которые способен воспринять процесс. Например канал buffer2P3_ch передает сигналы s1, s2, s3 от процесса P1 и сигнал s4 от процесса P2.

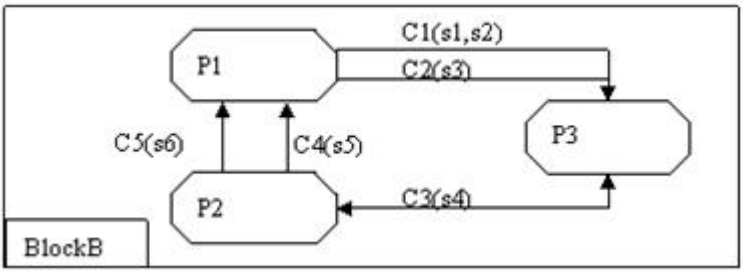

Рис. 2. Передача сигналов на SDL

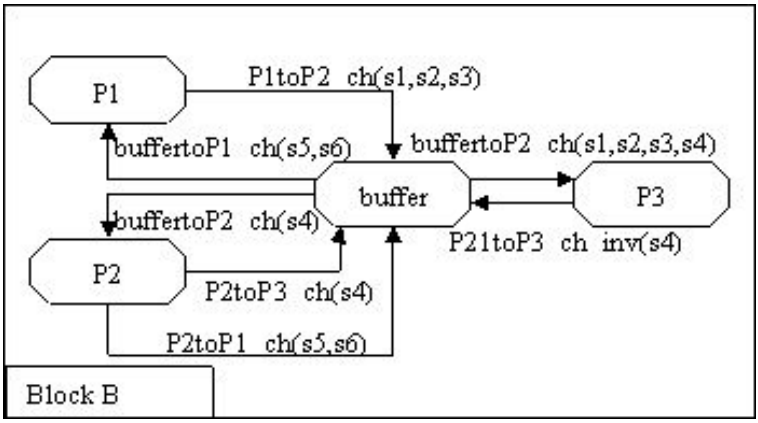

Рис. 3. Передача сигналов на dREAL после трансляции

Тело состояния

Последовательность операторов, образующих тело SDL-состояния, переводится в один или несколько dREAL-переходов по следующим правилам.

- каждый SDL-оператор **OUTPUT** переходит в соответствующий ему dREAL-оператор **WRITE**, помещаемый в новый переход;
- для каждого оператора **INPUT** в одном состоянии создается dREAL-переход с тем же именем, что и исходное состояние. Сработает тот из них, сигнал которого придет первым. Если оператор **INPUT** имеет разрешающее условие, перед состоянием, содержащим оператор **READ**, вставляется dREAL-переход с оператором **WHEN condition**, с нулевым временем ожидания и переходом к чтению сигнала;
- тело оператора **TASK** (список присваиваний), по возможности, помещается в один dREAL-переход (это невозможно сделать, если выражения содержат операторы экспорта/импорта или обозреваемые переменные). При этом, если текущий dREAL-переход содержит оператор **READ** или **WRITE**, создается новый переход;
- если SDL-выражения содержат операторы **EXPORT**, **IMPORT** или **VIEW**, перед dREAL-переходом, использующим это выражение, вставляется группа служебных переходов, реализующих эти конструкции[5];

- если левая часть оператора присваивания является переменной, объявленной в секции **VIEWED** (обозреваемой), после этого выражения вставляются служебные переходы, осуществляющие действия, необходимые для отправки нового значения этой переменной адресатам [5];
- для SDL-оператора **DECISION** создается группа альтернативных dREAL-переходов, помеченных одним состоянием;
- SDL-операторы SET, RESET переходят в dREAL-операторы WRITE.

Оператор DECISION

Операторы ветвления языков SDL и dREAL имеют значительные отличия. Кроме того, ограничение на единственность операторов в dREAL-переходе создает дополнительные проблемы при трансляции этих конструкций.

Условный оператор SDL имеет следующий вид:

```
DECISION условие;
   { {вариант:}+ переход }+
   [ELSE: переход]
ENDDECISION;
```

где **условие** – выражение произвольного типа,

вариант – ограничение на множество значений ответа: константа (или множество констант) или один или несколько интервалов (открытых или закрытых),

переход – последовательность произвольных команд (за исключением операторов **INPUT**) языка SDL.

Условный оператор dREAL имеет вид

WHEN выражение-условие программа

где **выражение-условие –** обычное выражение, результат которого имеет логический тип,

программа – последовательность из одного или нескольких операторов dREAL.

Для каждого SDL-варианта создается собственный альтернативный переход – начало ветвления. Оно содержит оператор IF или WHEN с условием, полученным из SDL-условия.

Выражение-вариант для варианта-ELSE строится как отрицание дизъюнкции всех вариантов оператора DECISION.

Кроме того, создаются состояния и для конца условия (end_decision).

REAL-состояние начала ветвления выглядит так:

```
TRANSITION begin_decision_N
```

```
 EXE SKIP
 FROM NOW TO INF
JUMP state_name_T.

TRANSITION state_name_T
 WHEN условие вариант
 FROM NOW TO INF
JUMP state2.
```

Оно срабатывает, только если условие из оператора **WHEN** истинно и осуществляет переход к последовательности состояний, реализующих тело варианта. Эти состояния строятся по обычным правилам, описанным выше.

Кроме того, если **DECISION** не содержит варианта **ELSE**, добавляется служебное dREAL-состояние с **ELSE-**выражением. Если этого не сделать, процесс попадёт в тупиковую ситуацию, если выражение **условие** не удовлетворяет ни одному **варианту**.

Процедуры

Те процедуры SDL, которые описывают атомарные действия, и которые можно транслировать в оператор EXE, переводятся в процедуры dREAL. Все остальные процедуры раскрываются в месте своего вызова [5].

3.3. Система моделирования и верификации, использующая SPIN

Трансляция в язык Promela выполняется следующим образом. Сначала dREAL.ex-спецификация преобразуется во внутреннее представление, близкое к дереву синтаксического разбора, а затем проходом по дереву генерируется выходной текст на языке Promela. Транслятор написан на языке Perl.

Опишем изменения в трансляции по сравнению с [5]. Прежде всего, новая версия транслятора ориентирована на использование новой версии SPIN (6.0+), которая поддерживается графическим интерфейсом iSpin. Также добавлена поддержка макросов-процедур, которые транслируются в inline-макроопределения языка Promela.

Рассмотрим пример трансляции процедур. Процедура на языке dREAL.ex увеличения значения переменной

```
PROCEDURE inc(x,y)
x := x + y;
END PROCEDURE.
```

транслируется в макроопределение на языке Promela

```
inline inc(x, y){ d_step{ x = x + y; } } .
```

В процедурах могут использоваться локальные переменные, которые обнуляются в конце dstep-части inline-функции в Promela, чтобы не увеличивать пространство состояний.

Также процедуры могут вызывать другие процедуры.

Конструкция **d_step{ ... }** в языке Promela означает «детерминированный шаг» и используется для объединения нескольких шагов спецификации в одно «атомарное» действие, когда значения переменных на промежуточных шагах (внутри **d_step**) не представляют интереса для верификации. Как и в языке SDL, рекурсия (явная или неявная) недопустима.

Вызов вышеприведённой процедуры на языке dREAL.ex:

```
TRANSITION start
   EXE  v1:=0;
        CALL inc(v1, 5);
        ASSERT(v1==5);
   FROM NOW TO NOW
   JUMP done.
```

переводится в следующий фрагмент на языке Promela:

```
local byte v1;
L_start :
atomic{
  v1 = 0;
  /* call inline */ inc(v1, 5);
  assert((v1==5));
  goto L_done;
} .
```

4. СПЕЦИФИКАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ СЕТЬЮ КАСС-ТЕРМИНАЛОВ

Эта система состоит из терминалов (процесс terminal на рис. 4) и главной машины (процесс main_machine на рис. 4), к которой терминалы обращаются за информацией о наличии и цене билетов до нужной пассажиру станции. Система работает следующим образом. В начале работы терминалы инициализируются и каждому присваивается порядковый номер. Если к ещё не инициализированному терминалу есть очередь (процесс queue на рис. 4), то после его инициализации начинается обслуживание первого пассажира (процесс passenger на рис. 4) из очереди. Если же очереди нет, то после инициализации терминал переходит в режим ожидания пассажира.

Пассажир подходит к нужному терминалу. Если терминал свободен, то начинается обслуживание этого пассажира. Если же терминал занят или ещё не инициализирован, то этот пассажир встает в очередь. Как только терминал освободится или инициализируется, пассажир нажимает на кнопку начала работы. Терминал сообщает о готовности и ожидает указания номера станции. Получив номер станции, терминал запрашивает у главной машины наличие билетов и стоимость проезда до этой станции. Если билетов нет или все оставшиеся билеты забронированы другими пассажирами, то главная машина сообщает об этом терминалу, который передает это сообщение пассажиру, и он уходит из очереди. Если же свободные билеты есть, то один билет бронируется на время сеанса, и главная машина сообщает терминалу стоимость билета, которую тот показывает на индикаторе. Возможно два случая. Если у пассажира не хватает выделенных на покупку билета средств, то он прекращает сеанс по своей инициативе, нажав кнопку «Cancel», а бронь билета снимается. Если средств хватает, то пассажир оплачивает проезд и нажимает кнопку «Done». В этом случае, если сумма проходит проверку, то терминал печатает билет до нужной пассажиру станции, а иначе терминал показывает на индикаторе, сколько осталось заплатить. В любом случае в конце сеанса пассажир завершает свою работу, нажав кнопку «Done», а терминал переходит в режим ожидания нового пассажира. Если в очереди к данному терминалу есть хотя бы один пассажир, то первый из них начинает свой сеанс.

Существуют особенности, связанные с одновременной работой нескольких пассажиров. Если они одновременно запрашивают последний билет до одной и той же станции, то в результате недетерминированного выбора происходит бронирование билета на время сеанса связи для одного пассажира, а другие пассажиры завершат свою работу из-за отсутствия билетов.

Выполнимая спецификация этой системы на языке SDL представлена на Рис. 4, 5.1, 5.2, 5.3, 6.1, 6.2, 7.1, 7.2, 8, контекст спецификации на языке SDL представлен в Приложении 1, спецификация системы на языке dREAL представлена в Приложении 2 в графическом (рис. 9) и текстовом виде с комментариями.

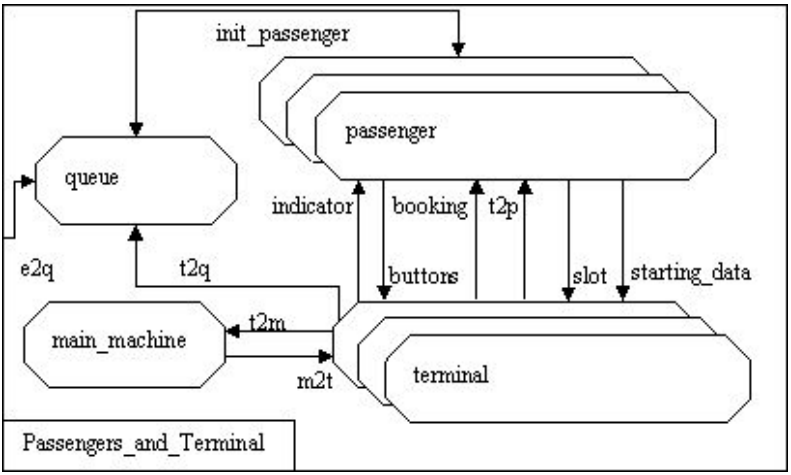

Рис. 4. Спецификация системы управления сетью касс-терминалов на верхнем уровне, представленная на языке SDL

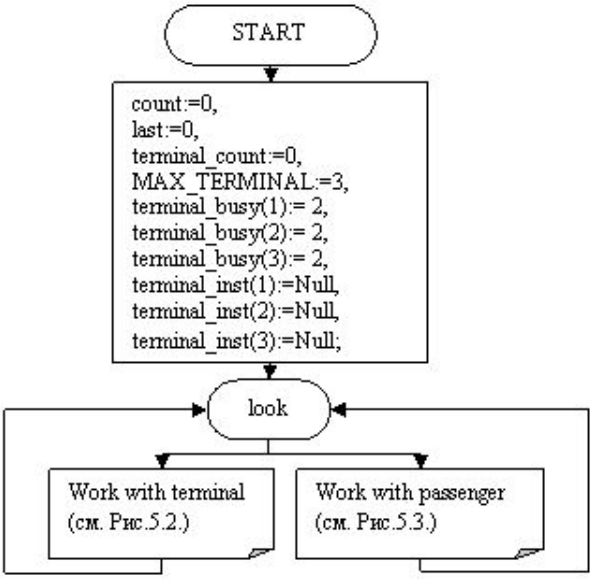

Рис. 5.1. Спецификация процесса queue

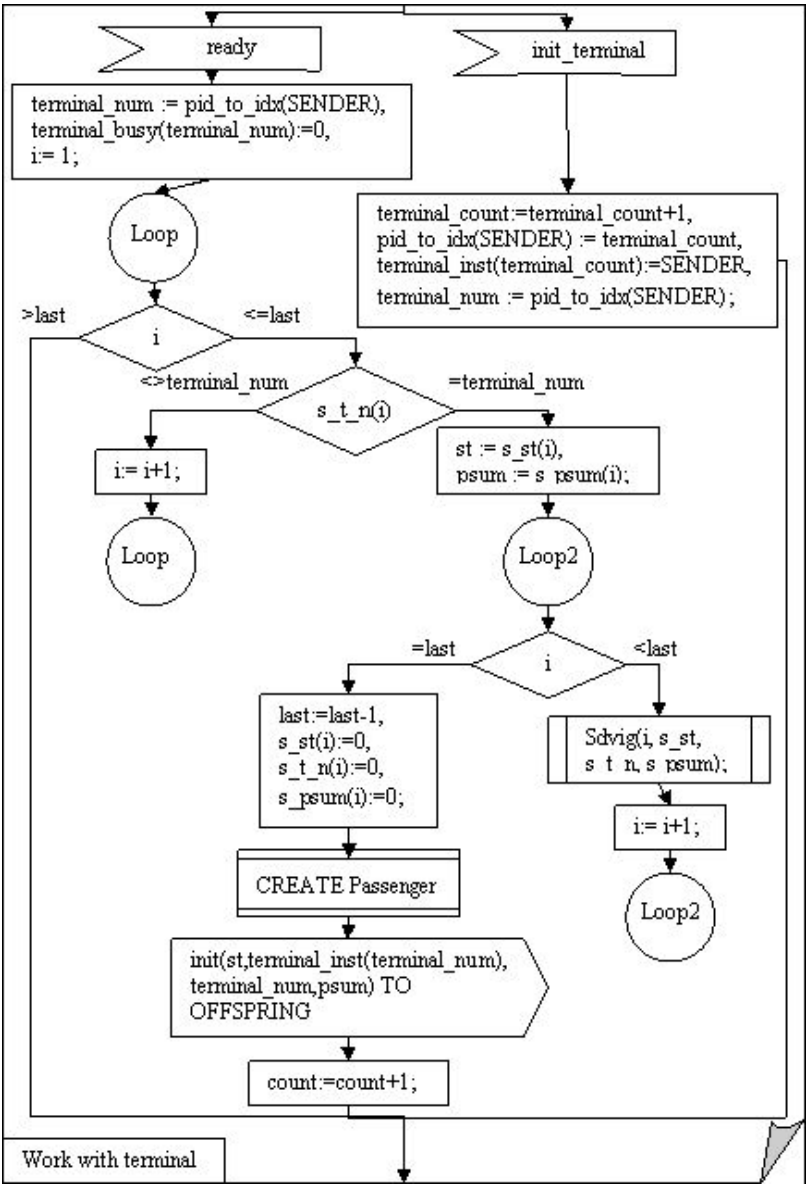

Рис. 5.2. Спецификация фрагмента процесса queue

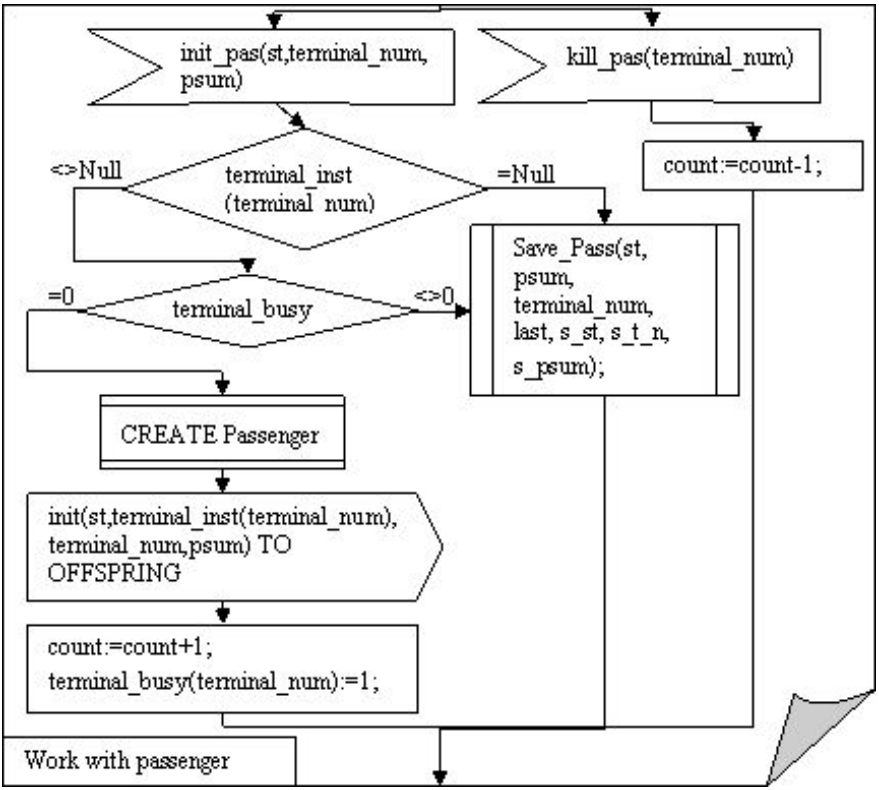

Рис. 5.3. Спецификация фрагмента процесса queue

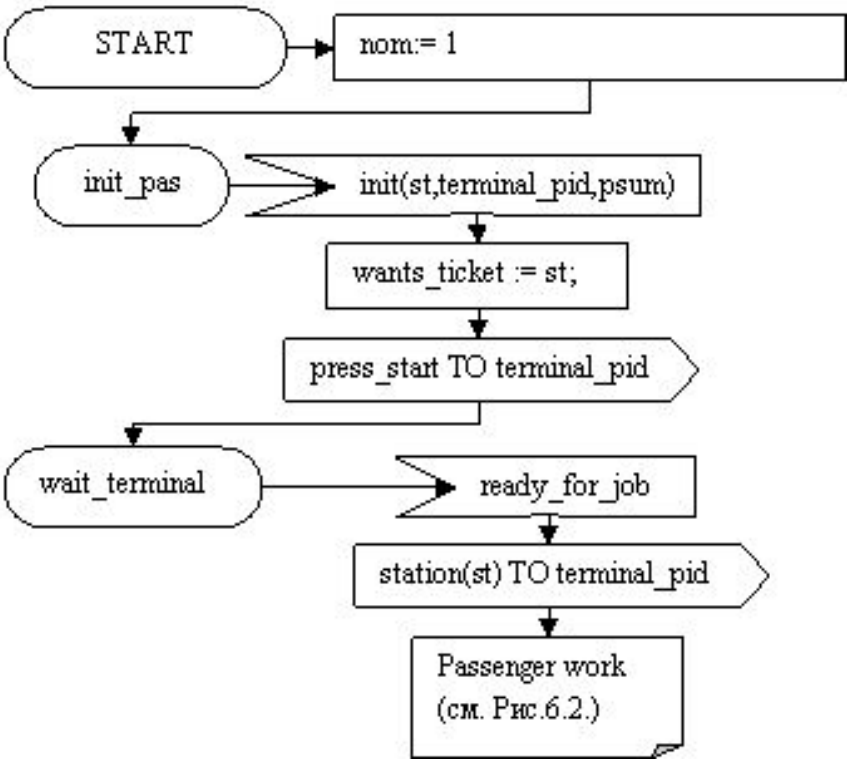

Рис. 6.1. Спецификация процесса passenger

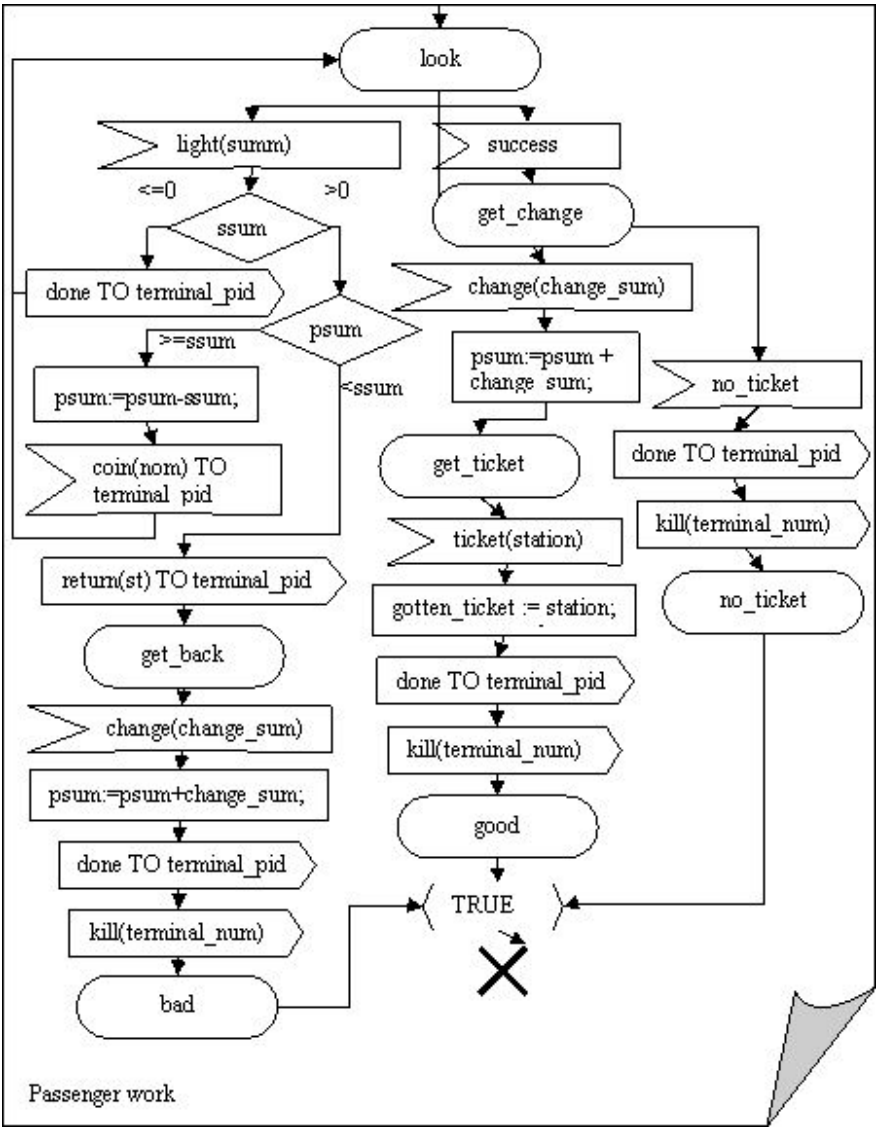

Рис. 6.2. Спецификация фрагмента процесса passenger

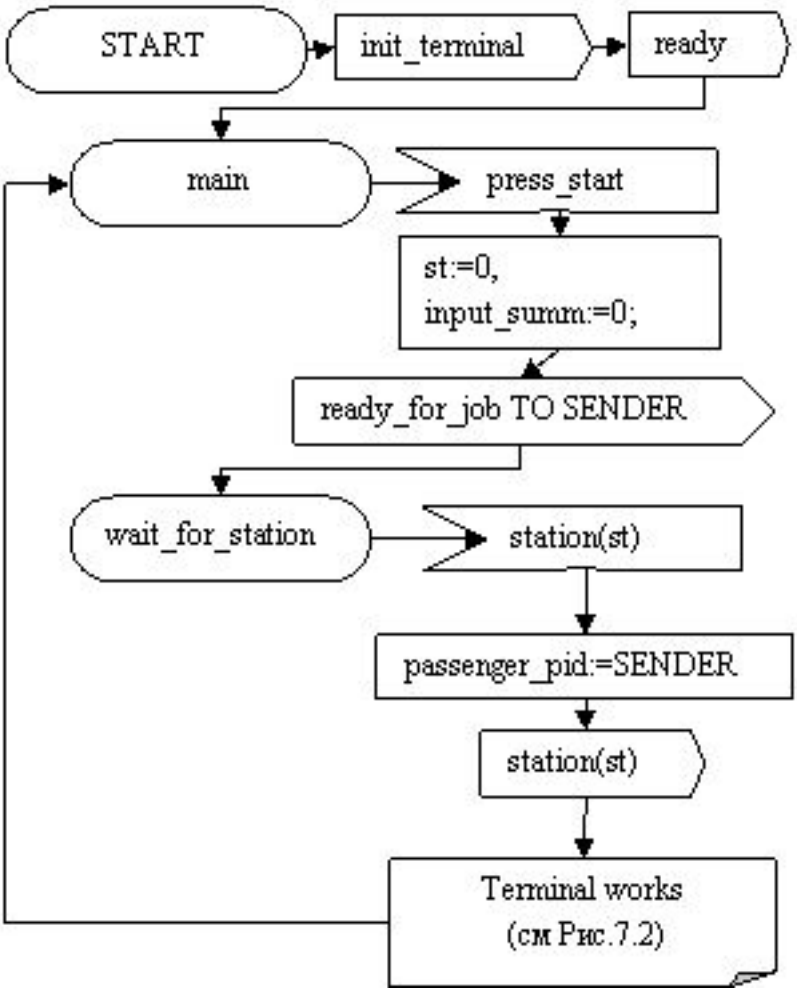

Рис. 7.1. Спецификация процесса terminal

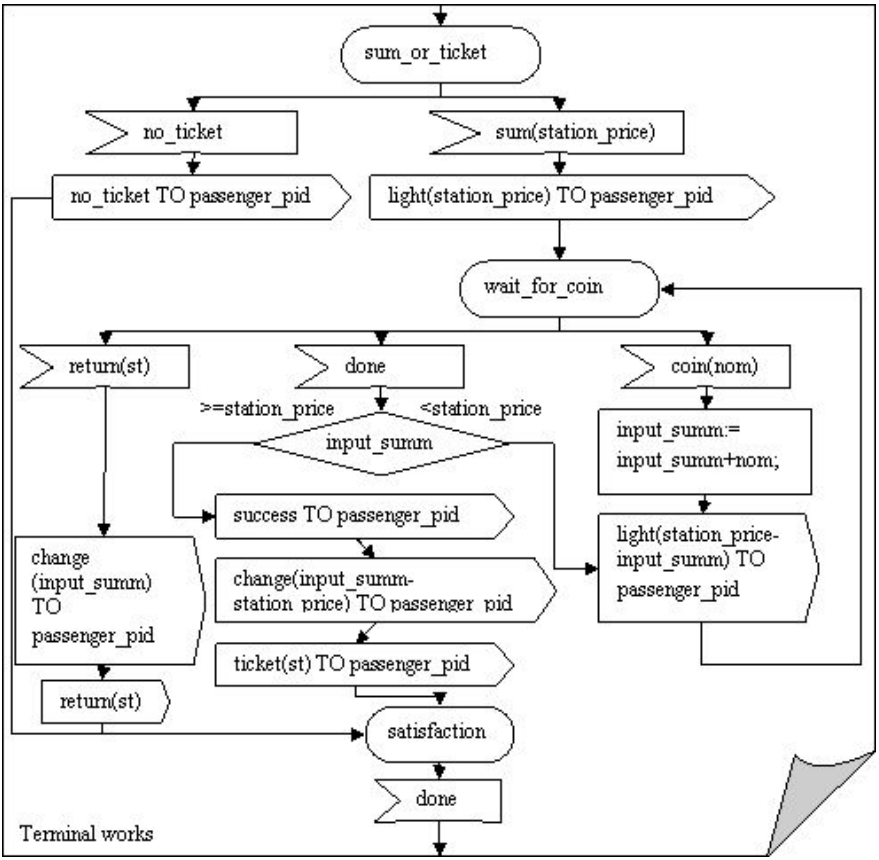

Рис. 7.2. Спецификация фрагмента процесса terminal

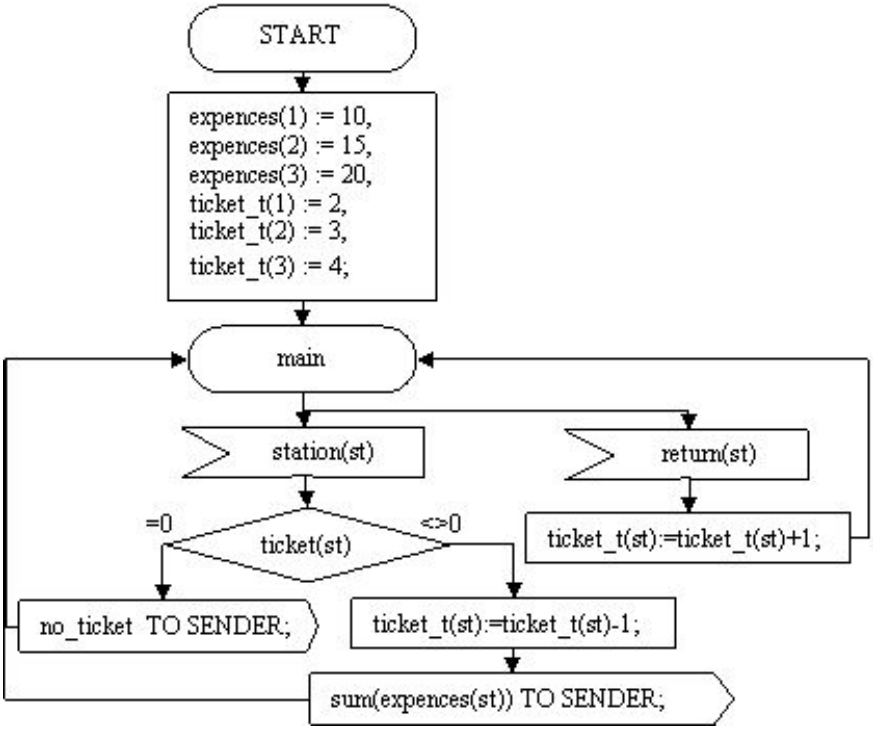

Рис. 8. Спецификация процесса main-machine

5. ЭКСПЕРИМЕНТЫ

5.1 Эксперимент с системой автоматического моделирования

В первом эксперименте полученная dREAL-спецификация и соответствующий запрос пользователя были поданы на вход системы автоматического моделирования. Эта dREAL-спецификация была успешно проверена с помощью следующих запросов:

Переменные использующиеся в запросах:

`wants_ticket[pid]` − массив индексируемый PId процессов, в кото-

ром хранятся номера станций, запрашиваемых пассажирами. Каждому PId соответствует номер запрашиваемой станции.

gotten_ticket[pid] – массив индексируемый PId процессов, в котором хранятся номера станций, билеты до которых получили пассажиры. Каждому PId соответствует номер станции на билете.

enough_money[pid] – массив индексируемый PId процессов, в котором хранится информация хватает денег пассажиру для покупки или нет.

reserve_ticket[pid] – массив индексируемый PId процессов, в котором хранятся номера станций, билет до которых был забронирован пассажиром. Каждому PId соответствует номер запрашиваемой станции.

passenger[pid] – экземпляр процесса `Passenger`

passenger[pid].psum – переменная в которой хранится сумма, которой располагает пассажир для покупки билета. У каждого экземпляра процесса Passenger эта переменная своя.

passenger[pid].summ – переменная в которой стоимость билета до станции, которую запрашивает пассажир. У каждого экземпляра процесса `Passenger` эта переменная своя.

passenger[pid].good – состояние `good` экземпляра процесса `Passenger.` Процесс попадает в него, если все закончилось хорошо, билет получен.

passenger[pid].bad – состояние `bad` экземпляра процесса `Passenger.` Процесс попадает в него, если не хватило денег на покупку, билет не был получен.

passenger[pid].no_ticket – состояние `no_ticket` экземпляра процесса `Passenger.` Процесс попадает в него, если пассажиру не хватило билетов до запрашиваемой станции, билет не был получен.

Эксперимент проводился с 5 терминалами и 10 пассажирами.

Система инициализирована следующими данными.

Количество станций: 3.

Количество билетов и стоимость проезда до станции 1: 2 билета, 10 монет.

Количество билетов и стоимость проезда до станции 2: 3 билета, 15 монет.

Количество билетов и стоимость проезда до станции 3: 4 билета, 20 монет.

Пассажиры инициализированы следующими данными.

e2q.init_pas(1,1, 15) – пассажир 1 едет до 1 станции, подходит к 1 терминалу и имеет 15 монет.

e2q.init_pas(1,2, 20) – пассажир 2 едет до 1 станции, подходит к 2 терминалу

и имеет 20 монет.

e2q.init_pas(1,3, 5) – пассажир 3 едет до 1 станции, подходит к 3 терминалу и имеет 5 монет.

e2q.init_pas(1,1, 10) – пассажир 4 едет до 1 станции, подходит к 1 терминалу и имеет 10 монет.

e2q.init_pas(2,1, 15) – пассажир 5 едет до 2 станции, подходит к 1 терминалу и имеет 15 монет.

e2q.init_pas(3,2, 20) – пассажир 6 едет до 3 станции, подходит к 2 терминалу и имеет 20 монет.

e2q.init_pas(2,3, 15) – пассажир 7 едет до 2 станции, подходит к 3 терминалу и имеет 15 монет.

e2q.init_pas(2,4, 15) – пассажир 8 едет до 2 станции, подходит к 4 терминалу и имеет 15 монет.

e2q.init_pas(3,3, 25) – пассажир 9 едет до 3 станции, подходит к 3 терминалу и имеет 25 монет.

e2q.init_pas(1,5, 15) – пассажир 10 едет до 1 станции, подходит к 5 терминалу и имеет 10 монет.

В процессе эксперимента были выполнены запросы, при которых SDL-спецификация содержит ошибки. Эти запросы означают, что в какой-либо конфигурации для произвольного экземпляра процесса Passenger с идентификатором pid выполняются следующие условия:

1. станция на полученном билете не совпадает с требуемой станцией:
   WHEN ((wants_ticket[pid] != gotten_ticket[pid]) && (passenger[pid] AT passenger[pid].good)) PAUSE .
2. если в момент обращения к терминалу достаточно билетов до требуемой станции, и пассажиру хватает денег, но пассажир не получит билет (т.е. не попадет в состояние good):
   WHEN (reserve_ticket[pid]==1 && enough_money[pid]==1 && (passenger[pid] AT passenger[pid].bad || passenger[pid] AT passenger[pid].no_ticket))PAUSE.
3. если в момент обращения к терминалу нет билетов до требуемой станции, но процесс Passenger попадет в состояние good (т.е. пассажир получит билет) или bad(т.е. пассажир обнаружит недостаток средств):
   WHEN (reserve_ticket[pid]==0 &&(passenger[pid] AT passenger[pid].good || passenger[pid] AT passenger[pid].bad)) PAUSE .
4. если в момент обращения к терминалу есть билеты до требуемой станции, и у пассажира не хватает денег на билет, но процесс Passenger

попадет в состояние good(т.е. пассажир получит билет) или попадет в состояние no_ticket (т.е. нет билетов до требуемой станции):

```
WHEN (reserve_ticket[pid]==1 &&
enough_money[pid]==0 && (passenger[pid] AT
passenger[pid].good || passenger[pid] AT
passenger[pid].no_ticket))PAUSE.
```

Результаты этого эксперимента состоят в следующем. Один из пассажиров 1, 2, 10 не получит билета из-за того, что билетов до 1 станции не хватит. Пассажир 3 не получит билета из-за нехватки денег. Пассажир 4 не получит билет из-за того, что билеты до 1 станции кончились. Остальные пассажиры получат билеты.

5.2. Эксперимент с системой SPIN

Второй эксперимент состоял в трансляции dREAL-спецификации системы управления сетью касс-терминалов в язык PROMELA и верификации ее с помощью системы SPIN.

Из-за того что в системе SPIN невозможно использовать в формулах локальные переменные, такие локальные переменные отмечены в SDL-спецификации как REVEALED, что делает их глобальными массивами при переводе в dREAL. Таким образом, локальный массив процесса, не имеющего экземпляров, просто становится глобальным массивом, а скалярная переменная процесса с экземплярами становится глобальным массивом, индексированным идентификаторами процессов.

Определим вначале следующие функции и предикаты.

`passenger[pid].wants_ticket` – функция, определяющая номер станции, до которой пассажиру нужен билет (переменная wants_ticket экземпляра процесса passenger с PId'ом **`pid`**);

`passenger[pid].gotten_ticket` – функция, определяющая номер станции, до которой пассажир получил билет;

`passenger[pid].`– функция, определяющая, что пассажир получил;

```
reserved_ticket(pid)  : PRED
(passenger[pid].summ > 0);
```

(пассажир получил сигнал/сообщение о цене билета, то есть он зарезервировал билет);

```
enough_tickets(pid)  : PRED
   (main_machine.ticket[passenger[pid].wants_ticket] >
   0);
```

(до нужной пассажиру станции есть билеты);

```
   enough_money(pid)   : PRED
(passenger[pid].psum >=
   main_machine.expences[passenger[pid].wants_ticket]);
```

(у пассажира достаточно денег на билет до нужной ему станции) ;

```
got_good_ticket(pid)   : PRED
   (passenger[pid].gotten_ticket == passenger
   [pid].wants_ticket);
```

(пассажир получил билет до нужной ему станции).

Проверялись следующие три свойства для 1–5 терминалов и 2–-10 пассажиров.

5.2.1. Свойство prop1

Если пассажир зарезервировал билет до нужной ему станции, и у него достаточно денег, то он получит билет до этой станции.

Тогда свойство **prop1** на языке SPL представляется в виде

```
( ALL PROCESS passenger WITH PID pid
( (reserved_ticket(pid) && enough_money(pid)
 =>
(EACH ET got_good_ticket(pid)))) ).
END;
.
```

Это свойство представляется на языке dREAL.log в виде

```
 ( ALL PROCESS passenger WITH PID pid
 ( (reserved_ticket( wants_ticket[pid] ) &&
enough_money(pid)
  =>
(EACH ET ( got_good_ticket(pid))))) ).
reserved_ticket(pid)   : PRED
   ( passenger[pid].summ > 0);
enough_money(pid)   : PRED
   (psum[ pid ] >= expences[  wants_ticket[pid] ]);
got_some_ticket(pid)   : PRED
   (gotten_ticket[pid] > 0);
got_good_ticket(pid)   : PRED
   (gotten_ticket[pid] == wants_ticket[pid]);
.
```

Следующая LTL формула представляет свойство prop1:

```
enough_tickets && enough_money -> <> (got_good_ticket)
```

.

В силу специфики Spin для верификации используется отрицание проверяемой формулы, а порождаемый верификатор находит контр-пример.

5.2.2. Свойство prop2

Если до нужной пассажиру станции билетов нет, то он не получит никакого билета.

Тогда свойство **prop2**на языке SPLпредставляется в виде

```
 ( ALL PROCESS passenger WITH PID pid
( (not_enough_tickets(pid)
        =>
(EACH AT ( NOT got_some_ticket(pid))))) ).
где
not_enough_tickets(pid)  : PRED
(main_machine.ticket[  passenger[pid].wants_ticket  ]  =
0);
/* до нужной пассажиру станции нет билетов */
got_some_ticket(pid)  : PRED
(passenger[pid].gotten_ticket > 0 )
/* пассажир получил какой-то билет */
```

Это свойство представляется на языке dREAL.log в виде

```
( ALL PROCESS passenger WITH PID pid
( (not_enough_tickets(pid)
        =>
(EACH AT ( NOT got_some_ticket(pid))))) ).
not_enough_tickets(pid)  : PRED
(ticket[ wants_ticket[pid] ] = 0);
got_some_ticket(pid)  : PRED
(gotten_ticket[pid] > 0 )
```

.Следующая LTL формула представляет свойство prop2:

```
not_enough_tickets -> [] ! got_some_ticket
```

5.2.3. Свойство prop3

Если у пассажира выделено недостаточно денег до нужной ему станции, то он не получит никакого билета.

Тогда свойство **prop3**на языке SPLпредставляется в виде

```
    ( ALL PROCESS passenger WITH PID pid
   ( (not_enough_money( pid )
           =>
```

```
   (EACH AT ( NOT got_some_ticket(pid))))) ).
   где
   not_enough_money(pid)   : PRED
   (passenger[pid].psum  <  main_machine.expences[  pas-
senger[pid].wants_ticket ]);
   /* пассажир выделил недостаточно денег на билет до
нужной ему станции */
   got_some_ticket(pid)   : PRED
   (passenger[pid].gotten_ticket > 0 )
```

Это свойство представляется на языке dREAL.log в виде

```
    ( ALL PROCESS passenger WITH PID pid
   ( (not_enough_money( wants_ticket[pid] )
           =>
   (EACH AT ( NOT got_some_ticket(pid))))) ).
   где
   not_enough_money(station)   : PRED
   (psum[pid] < expences[station]);
   got_some_ticket(pid)   : PRED
   (gotten_ticket[pid] > 0 )
```

.

Следующая LTL формула представляет свойство prop3:

```
   not_enough_money -> [] ! got_some_ticket
```

.

В силу особенности SPIN для доказательства этих свойств утверждались их отрицания, которые опровергались системой SPIN. Время работы верификатора — в пределах одной-двух секунд. В этих экспериментах рассматривалась следующая инициализация экземпляров процессов Passenger:

```
/* Add read-from-channels lines here */
/*  C_Env2queue_ch  :  S_init_pas    (aSENDER,  STATION,
TERMINAL, PURSE, TARGET_PID=Null);  */
C_Env2queue_ch ! S_init_pas(_pid, 1, 1, 101, Null);
C_Env2queue_ch  !  S_init_pas(_pid,  2,  1,  101,  Null);
/* to the same terminal to a different station */

#if _MAX_TERMINAL > 1
C_Env2queue_ch ! S_init_pas(_pid, 3, 2, 101, Null);
C_Env2queue_ch ! S_init_pas(_pid, 2, 1, 101, Null);
#endif
#if _MAX_TERMINAL > 2
```

```
C_Env2queue_ch ! S_init_pas(_pid, 2, 3, 101, Null); /*
To higher terminals */
C_Env2queue_ch ! S_init_pas(_pid, 1, 2, 101, Null);
#endif
#if _MAX_TERMINAL > 3
C_Env2queue_ch ! S_init_pas(_pid, 1, 4, 101, Null);
C_Env2queue_ch ! S_init_pas(_pid, 2, 1, 101, Null);
#endif
#if _MAX_TERMINAL > 4
C_Env2queue_ch ! S_init_pas(_pid, 2, 5, 101, Null);
C_Env2queue_ch ! S_init_pas(_pid, 1, 1, 101, Null);
#endif
```

.

6. ЗАКЛЮЧЕНИЕ

Язык Dynamic-REAL является комбинированным языком спецификаций как распределенных систем, так и их свойств. Преимущества языка dREAL обусловлены простым синтаксисом, допускающим графическое представление выполнимых спецификаций, полной компактной операционной семантикой и выразительным языком представления свойств. Программный комплекс SRDSV2 является мощной системой, которая включает транслятор из SDL в dREAL, системы симуляции и автоматического моделирования dREAL-спецификаций, а также использует известную систему верификации методом проверки моделей SPIN. Входной язык этого транслятора включает все базовые конструкции языка SDL (кроме абстрактных типов данных), которые образуют язык SDL-88 и широко применяются на практике [12]. Система автоматического моделирования играет важную роль, так как может успешно применяться и в тех случаях, когда система верификации SPIN неприменима из-за громоздких и сложных моделей SDL-спецификаций. Преимущества нашего подхода иллюстрируются автоматическим моделированием и верификацией динамической системы управления сетью касс-терминалов.

Предполагается расширить комплекс SRDSV транслятором dREAL-спецификаций в сети Петри высокого уровня и системой верификации методом проверки моделей относительно свойств, представленных в логике ветвящегося времени CTL. Также предполагается расширить язык запросов системы автоматического моделирования с целью обнаружения тупиков и зацикливания.

СПИСОК ЛИТЕРАТУРЫ

1. Карабегов А.В., Тер-Микаэлян Т.М. Введение в язык SDL. — М.: Радио и связь, 1993.
2. Карпов Ю.Г. Model checking. Верификация параллельных и распределенных программных систем. — Санкт-Петербург: БХВ-Петербург, 2010.
3. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. — М.: МЦНМО, 2002.
4. Непомнящий В.А., Бодин Е.В, Веретнов С.О. Язык спецификаций распределенных систем Dynamic-REAL // Препринт ИСИ СО РАН № 147, 2007.
   http://www.iis.nsk.su/preprints/pdf/147.pdf
5. Непомнящий В.А., Бодин Е.В., Веретнов С.О. Моделирование и верификация распределенных систем, представленных на языке SDL, с помощью языка Dynamic-REAL // Препринт ИСИ СО РАН № 156, 2010.
   http://www.iis.nsk.su/preprints/pdf/156.pdf
6. Непомнящий В.А., Шилов Н.В. Real92: Комбинированный язык спецификаций для систем и свойств взаимодействующих процессов реального времени // Программирование. — 1993. — № 6. — С. 64–80.
7. Непомнящий В.А., Шилов Н.В., Бодин Е.В. Спецификация и верификация распределенных систем средствами языка Elementary-REAL // Программирование. — 1999. — № 4. — С. 54–67.
8. Непомнящий В.А., Шилов Н.В., Бодин Е.В. REAL: язык для спецификации и верификации систем реального времени // Системная информатика.— Новосибирск: Наука, 2000. — Вып. 7. — С. 174–224.
9. Becker Ph., Christmann D., Gotzhein R. Model-Driven Development of Time-Critical Protocols with SDL-MDD. Proc. SDL 2009. Lect. Notes in Comp. Sci. — 2009. — Vol. 5719. — P. 34–52.
10. Bosnacki D., Dams D., Holenderski L., Sidorova N. Model checking SDL with Spin. Proc. TACAS/ETAPS 2000. Lect. Notes Comput. Sci. — 2000. — Vol. 1785. — P. 363–377.
11. Bozga M., Graf S., Ober Il., Ober Iu., Sifakis J. The IF toolset. Proc. SFM-RT 2004. Lect. Notes in Comp. Sci. — 2004. — Vol. 3185. — P. 237–267.
12. Grammes R., Gotzhein R. SDL Profiles – Formal Semantics and Tool Support. Proc. FASE 2007. Lect. Notes in Comp. Sci. — 2007. — Vol. 4422. — P. 200–214.
13. Holzmann, G.J.: The SPIN model checker. Primer and Reference Manual.-Addison-Wesley, 2004.
14. Nepomniaschy V.A., Shilov N.V., Bodin E.V., Kozura V.E. Basic-REAL: integrated approach for design, specification and verification of distributed systems. Proc. IFM 2002. Lect. Notes Comput. Sci. — 2002. — Vol. 2335. — P. 69–88.

15. Showk A., Szczesny D., Traboulsi Sh., Badr I., Gonzalez E., Bilgic A. Modeling LTE Protocol for Mobile Terminals Using a Formal Description Technique. Proc. SDL 2009. Lect. Notes in Comp. Sci. — 2009. — Vol. 5719. — P. 222–238.
16. Sidorova N., Steffen M. Verifing Large SDL-Specifications Using Model Checking. Proc. SDL 2001. Lect. Notes in Comp. Sci. — 2001. — Vol. 2078. — P. 403–420.
17. Specification and Description Language (SDL). — CCITT, Recommendation Z.100, 1988.

ПРИЛОЖЕНИЕ 1.
КОНТЕКСТ СПЕЦИФИКАЦИИ «СЕТЬ КАСС» НА ЯЗЫКЕ SDL

```
SYSTEM All;
  BLOCK Passengers_and_Terminals;
      SIGNAL
          station     (integer),  /*сигнал с номером станции*/
          coin        (integer),  /*сумма, опущенная в монетоприемник*/
          done,                   /*сигнал завершения действия*/
          sum         (integer), /*сигнал с требуемой стоимостью
    билета*/
          kill        (integer),/*сигнал о завершении работы с терминалом N*/
          init_pas    (integer, integer, integer),/*сигнал о приходе пассажира от окружения*/
          init        (integer, PId, integer, integer),/*сигнал инициализации пассажира для работы с конкретным терминалом*/
          light       (integer),/*сигнал с информацией о стоимости билета*/
          ticket          (integer),/*сигнал о выдаче билета*/
          ready_for_job,          /*сигнал готовности терминала к работе*/
          no_ticket,          /*сигнал об отсутствии билетов*/
          passenger_id    (PId),      /*сигнал с идентификатором пассажира*/
          return          (integer),/*сигнал об отмене операции*/
          ready,                  /*сигнал о включении терминала*/
          press_start;            /*сигнал о начале сеанса связи пассажира с терминалом*/
```

```
/*канал от окружения к очереди, по которому приходит информация о новых пассажирах*/
        SIGNALROUTE e2q
            FROM ENV TO queue WITH init_pas;
/*канал инициализации и деинициализации пассажиров*/
        SIGNALROUTE init_and_killing_passenger
            FROM queue TO passenger WITH init;
            FROM passenger TO queue WITH kill;
/*канал для отображения информации о наличии и стоимости билетов*/
        SIGNALROUTE indicator
            FROM terminal TO passenger WITH light,
no_ticket,ready_for_job;
/*канал для монетоприемника*/
        SIGNALROUTE slot
            FROM passenger TO terminal WITH coin;
/*канал для передачи информации о начале сеанса связи пассажира с терминалом*/
        SIGNALROUTE starting_data
            FROM passenger TO terminal WITH press_start;
/*канал для ввода требуемой информации пассажиром*/
        SIGNALROUTE buttons
            FROM passenger TO terminal WITH station,return,done;
/*канал для выдачи билетов*/
        SIGNALROUTE booking
            FROM terminal TO passenger WITH ticket;
/*канал для связи терминала с главной машиной*/
        SIGNALROUTE t2m
            FROM terminal TO main_machine WITH station, return;
/*канал для связи главной машины с терминалом*/
        SIGNALROUTE m2t
            FROM main_machine TO terminal WITH sum, no_ticket;
/*канал для инициализации терминала */
        SIGNALROUTE t2q
            FROM terminal TO queue WITH ready;
        NEWTYPE arr_int
            array (integer, integer)
        ENDNEWTYPE arr_int;
        PROCESS queue               REFERENCED;/*очередь*/
        PROCESS main_machine        REFERENCED;/*главная машина*/
```

```
PROCESS passenger(0,10)          REFERENCED;/*пассажиры*/
PROCESS terminal(5,5)            REFERENCED;/*терминалы*/

PROCEDURE Save_Pass REFERENCED;/*процедура сохранения пассажира в очереди*/
PROCEDURE Sdvig REFERENCED;/*процедура сдвига очереди на 1*/

ENDBLOCK Passengers_and_Terminals;
ENDSYSTEM All;
```

ПРИЛОЖЕНИЕ 2.
СПЕЦИФИКАЦИЯ «СЕТЬ КАСС-ТЕРМИНАЛОВ» НА ЯЗЫКЕ DREAL-EX

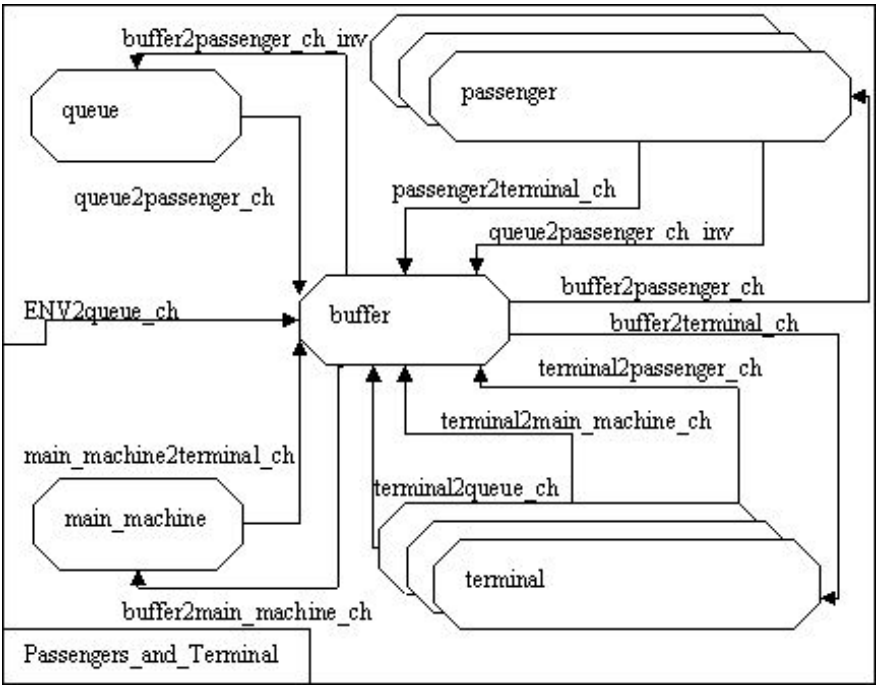

Рис. 9. Общая спецификация управления сетью касс-терминалов, представленная на языке dREAL

```
/* Sdl to Real Translater*/
All: BLOCK
 Passengers_and_Terminals: BLOCK
  TYPE int_arr__revealed IS
   integer ARRAY OF integer.
  TYPE integer__revealed IS
   PId ARRAY OF integer.
  TYPE arr_int IS
   integer ARRAY OF integer.
  PR VAR expences OF
int_arr__revealed.
  PR VAR ticket_t OF
int_arr__revealed.
  PR VAR wants_ticket OF inte-
ger__revealed.
  PR VAR gotten_ticket OF inte-
ger__revealed.
  INN UNB QUEUE CHN termi-
nal2queue_ch
    FOR ready
   WITH PAR p1 OF PId;
   FOR init_terminal
   WITH PAR p1 OF PId.
  INN UNB QUEUE CHN
main_machine2terminal_ch
   FOR sum
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR no_ticket
   WITH PAR p1 OF PId.
  INN UNB QUEUE CHN buff-
er2main_machine_ch
   FOR station
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR return
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId.
  INN UNB QUEUE CHN termi-
nal2main_machine_ch
   FOR station
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR return
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId.
  INN UNB QUEUE CHN buff-
er2terminal_ch
   FOR sum
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR no_ticket
   WITH PAR p1 OF PId;
   FOR press_start
   WITH PAR p1 OF PId;
   FOR coin
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR done
   WITH PAR p1 OF PId;
   FOR return
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR station
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId.
  INN UNB QUEUE CHN passen-
ger2terminal_ch
   FOR station
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR return
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR done
   WITH PAR p1 OF PId;
   FOR coin
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR press_start
   WITH PAR p1 OF PId.
  INN UNB QUEUE CHN termi-
nal2passenger_ch
   FOR change
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR light
   WITH PAR p1 OF integer,
   WITH PAR p2 OF PId;
   FOR no_ticket
   WITH PAR p1 OF PId;
   FOR ready_for_job
   WITH PAR p1 OF PId;
   FOR success
   WITH PAR p1 OF PId;
   FOR ticket
```

```
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId.
      INN UNB QUEUE CHN passen-
ger2queue_ch
        FOR kill_pas
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId.
      INN UNB QUEUE CHN buff-
er2passenger_ch
        FOR change
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId;
        FOR light
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId;
        FOR no_ticket
        WITH PAR p1 OF PId;
        FOR ready_for_job
        WITH PAR p1 OF PId;
        FOR success
        WITH PAR p1 OF PId;
        FOR init
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId,
        WITH PAR p3 OF integer,
        WITH PAR p4 OF integer,
        WITH PAR p5 OF PId;
        FOR ticket
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId.
      INN UNB QUEUE CHN
queue2passenger_ch
        FOR init
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId,
        WITH PAR p3 OF integer,
        WITH PAR p4 OF integer,
        WITH PAR p5 OF PId.
      INN UNB QUEUE CHN buff-
er2queue_ch
        FOR ready
        WITH PAR p1 OF PId;
        FOR init_terminal
        WITH PAR p1 OF PId;
        FOR init_pas
        WITH PAR p1 OF integer,
        WITH PAR p2 OF integer,
        WITH PAR p3 OF integer,
        WITH PAR p4 OF PId;
        FOR kill_pas
        WITH PAR p1 OF integer,
        WITH PAR p2 OF PId.
      INP UNB QUEUE CHN
ENV2queue_ch
        FOR init_pas
        WITH PAR p1 OF integer,
        WITH PAR p2 OF integer,
        WITH PAR p3 OF integer,
        WITH PAR p4 OF PId.

        FROM terminal CHN termi-
nal2queue_ch TO buffer.
        FROM main_machine CHN
main_machine2terminal_ch TO buffer.
        FROM buffer CHN buff-
er2main_machine_ch TO main_machine.
        FROM terminal CHN termi-
nal2main_machine_ch TO buffer.
        FROM buffer CHN buff-
er2terminal_ch TO terminal.
        FROM passenger CHN passen-
ger2terminal_ch TO buffer.
        FROM terminal CHN termi-
nal2passenger_ch TO buffer.
        FROM passenger CHN passen-
ger2queue_ch TO buffer.
        FROM buffer CHN buff-
er2passenger_ch TO passenger.
        FROM queue CHN
queue2passenger_ch TO buffer.
        FROM buffer CHN buffer2queue_ch
TO queue.
      FROM ENV CHN ENV2queue_ch
TO buffer.

      buffer: PROCESS
        PR VAR TARGET_PID OF PId.

        TRANSITION read_signal
        READ ready(TARGET_PID) FROM
terminal2queue_ch
        FROM NOW TO INF
        JUMP write_ready_queue_ch.

        TRANSITION read_signal
        READ init_terminal(TARGET_PID)
FROM terminal2queue_ch
        FROM NOW TO INF
```

```
        JUMP write_init_terminal_queue_ch.

        TRANSITION read_signal
         READ
sum(p1_integer,TARGET_PID)        FROM
main_machine2terminal_ch
         FROM NOW TO INF
        JUMP write_sum_terminal_ch.

        TRANSITION read_signal
         READ        no_ticket(TARGET_PID)
FROM main_machine2terminal_ch
         FROM NOW TO INF
        JUMP write_no_ticket_terminal_ch.

        TRANSITION
write_station_main_machine_ch
         WHEN (TARGET_PID <> Null)
         WRITE  station(p1_integer,SENDER)
INTO                                  buff-
er2main_machine_ch[TARGET_PID]
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_station_main_machine_ch
         WHEN (TARGET_PID = Null)
         WRITE  station(p1_integer,SENDER)
INTO buffer2main_machine_ch
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_return_main_machine_ch
         WHEN (TARGET_PID <> Null)
         WRITE   return(p1_integer,SENDER)
INTO                                  buff-
er2main_machine_ch[TARGET_PID]
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_return_main_machine_ch
         WHEN (TARGET_PID = Null)
         WRITE   return(p1_integer,SENDER)
INTO buffer2main_machine_ch
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION read_signal
         READ                              sta-
tion(p1_integer,TARGET_PID)        FROM
terminal2main_machine_ch
         FROM NOW TO INF
        JUMP
write_station_main_machine_ch.

        TRANSITION read_signal
         READ                              re-
turn(p1_integer,TARGET_PID)        FROM
terminal2main_machine_ch
         FROM NOW TO INF
        JUMP
write_return_main_machine_ch.

        TRANSITION
write_sum_terminal_ch
         WHEN (TARGET_PID <> Null)
         WRITE      sum(p1_integer,SENDER)
INTO buffer2terminal_ch[TARGET_PID]
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_sum_terminal_ch
         WHEN (TARGET_PID = Null)
         WRITE      sum(p1_integer,SENDER)
INTO buffer2terminal_ch
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_no_ticket_terminal_ch
         WHEN (TARGET_PID <> Null)
         WRITE   no_ticket(SENDER)   INTO
buffer2terminal_ch[TARGET_PID]
         FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_no_ticket_terminal_ch
         WHEN (TARGET_PID = Null)
         WRITE   no_ticket(SENDER)   INTO
buffer2terminal_ch
         FROM NOW TO NOW
        JUMP read_signal.
```

```
        TRANSITION
write_press_start_terminal_ch
        WHEN (TARGET_PID <> Null)
        WRITE press_start(SENDER) INTO
buffer2terminal_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_press_start_terminal_ch
        WHEN (TARGET_PID = Null)
        WRITE press_start(SENDER) INTO
buffer2terminal_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_coin_terminal_ch
        WHEN (TARGET_PID <> Null)
        WRITE     coin(p1_integer,SENDER)
INTO buffer2terminal_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_coin_terminal_ch
        WHEN (TARGET_PID = Null)
        WRITE     coin(p1_integer,SENDER)
INTO buffer2terminal_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_done_terminal_ch
        WHEN (TARGET_PID <> Null)
        WRITE done(SENDER) INTO buff-
er2terminal_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_done_terminal_ch
        WHEN (TARGET_PID = Null)
        WRITE done(SENDER) INTO buff-
er2terminal_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_return_terminal_ch
        WHEN (TARGET_PID <> Null)
        WRITE   return(p1_integer,SENDER)
INTO buffer2terminal_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_return_terminal_ch
        WHEN (TARGET_PID = Null)
        WRITE   return(p1_integer,SENDER)
INTO buffer2terminal_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_station_terminal_ch
        WHEN (TARGET_PID <> Null)
        WRITE station(p1_integer,SENDER)
INTO buffer2terminal_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_station_terminal_ch
        WHEN (TARGET_PID = Null)
        WRITE station(p1_integer,SENDER)
INTO buffer2terminal_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION read_signal
        READ                          sta-
tion(p1_integer,TARGET_PID)      FROM
passenger2terminal_ch
        FROM NOW TO INF
        JUMP write_station_terminal_ch.

        TRANSITION read_signal
        READ                          re-
turn(p1_integer,TARGET_PID)      FROM
passenger2terminal_ch
        FROM NOW TO INF
        JUMP write_return_terminal_ch.

        TRANSITION read_signal
        READ done(TARGET_PID) FROM
passenger2terminal_ch
```

```
        FROM NOW TO INF
        JUMP write_done_terminal_ch.

        TRANSITION read_signal
        READ
coin(p1_integer,TARGET_PID)      FROM
passenger2terminal_ch
        FROM NOW TO INF
        JUMP write_coin_terminal_ch.

        TRANSITION read_signal
        READ      press_start(TARGET_PID)
FROM passenger2terminal_ch
        FROM NOW TO INF
        JUMP write_press_start_terminal_ch.

        TRANSITION read_signal
        READ
change(p1_integer,TARGET_PID)     FROM
terminal2passenger_ch
        FROM NOW TO INF
        JUMP write_change_passenger_ch.

        TRANSITION read_signal
        READ
light(p1_integer,TARGET_PID)      FROM
terminal2passenger_ch
        FROM NOW TO INF
        JUMP write_light_passenger_ch.

        TRANSITION read_signal
        READ      no_ticket(TARGET_PID)
FROM terminal2passenger_ch
        FROM NOW TO INF
        JUMP write_no_ticket_passenger_ch.

        TRANSITION read_signal
        READ ready_for_job(TARGET_PID)
FROM terminal2passenger_ch
        FROM NOW TO INF
        JUMP
write_ready_for_job_passenger_ch.

        TRANSITION read_signal
        READ      success(TARGET_PID)
FROM terminal2passenger_ch
        FROM NOW TO INF
        JUMP write_success_passenger_ch.

        TRANSITION read_signal
        READ                    tick-
et(p1_integer,TARGET_PID) FROM termi-
nal2passenger_ch
        FROM NOW TO INF
        JUMP write_ticket_passenger_ch.

        TRANSITION read_signal
        READ
kill_pas(p1_integer,TARGET_PID)   FROM
passenger2queue_ch
        FROM NOW TO INF
        JUMP write_kill_pas_queue_ch.

        TRANSITION
write_change_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE change(p1_integer,SENDER)
INTO buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_change_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE change(p1_integer,SENDER)
INTO buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_light_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE     light(p1_integer,SENDER)
INTO buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_light_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE     light(p1_integer,SENDER)
INTO buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_no_ticket_passenger_ch
        WHEN (TARGET_PID <> Null)
```

```
        WRITE no_ticket(SENDER) INTO
buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_no_ticket_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE no_ticket(SENDER) INTO
buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_ready_for_job_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE ready_for_job(SENDER)
INTO buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_ready_for_job_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE ready_for_job(SENDER)
INTO buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_success_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE success(SENDER) INTO
buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_success_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE success(SENDER) INTO
buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE
init(p1_integer,p2_PId,p3_integer,p4_integer
,SENDER) INTO buff-
er2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE
init(p1_integer,p2_PId,p3_integer,p4_integer
,SENDER) INTO buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_ticket_passenger_ch
        WHEN (TARGET_PID <> Null)
        WRITE ticket(p1_integer,SENDER)
INTO buffer2passenger_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_ticket_passenger_ch
        WHEN (TARGET_PID = Null)
        WRITE ticket(p1_integer,SENDER)
INTO buffer2passenger_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION read_signal
        READ
init(p1_integer,p2_PId,p3_integer,p4_integer
,TARGET_PID) FROM queue2passenger_ch
        FROM NOW TO INF
        JUMP write_init_passenger_ch.

        TRANSITION write_ready_queue_ch
        WHEN (TARGET_PID <> Null)
        WRITE ready(SENDER) INTO buff-
er2queue_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION write_ready_queue_ch
        WHEN (TARGET_PID = Null)
```

```
        WRITE ready(SENDER) INTO buff-
er2queue_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_terminal_queue_ch
        WHEN (TARGET_PID <> Null)
        WRITE  init_terminal(SENDER)  IN-
TO buffer2queue_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_terminal_queue_ch
        WHEN (TARGET_PID = Null)
        WRITE  init_terminal(SENDER)  IN-
TO buffer2queue_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_pas_queue_ch
        WHEN (TARGET_PID <> Null)
        WRITE
init_pas(p1_integer,p2_integer,p3_integer,S
ENDER)              INTO              buff-
er2queue_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_init_pas_queue_ch
        WHEN (TARGET_PID = Null)
        WRITE
init_pas(p1_integer,p2_integer,p3_integer,S
ENDER) INTO buffer2queue_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_kill_pas_queue_ch
        WHEN (TARGET_PID <> Null)
        WRITE
kill_pas(p1_integer,SENDER)  INTO  buff-
er2queue_ch[TARGET_PID]
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION
write_kill_pas_queue_ch
        WHEN (TARGET_PID = Null)
        WRITE
kill_pas(p1_integer,SENDER)  INTO  buff-
er2queue_ch
        FROM NOW TO NOW
        JUMP read_signal.

        TRANSITION read_signal
        READ
init_pas(p1_integer,p2_integer,p3_integer,T
ARGET_PID) FROM ENV2queue_ch
        FROM NOW TO INF
        JUMP write_init_pas_queue_ch.
       END; /* buffer */

       queue: PROCESS

        PROCEDURE
Sdvig(Sdvig_s_st_i,Sdvig_s_t_n_i,Sdvig_s_
psum_i,Sdvig_s_st_i1,Sdvig_s_t_n_i1,Sdvig
_s_psum_i1);
        Sdvig_s_st_i:=Sdvig_s_st_i1;
        Sdvig_s_t_n_i:=Sdvig_s_t_n_i1;
        Sdvig_s_psum_i:=Sdvig_s_psum_i1;
       END PROCEDURE


        PROCEDURE
Save_Pass(Save_Pass_st,Save_Pass_psum,S
ave_Pass_terminal_num,Save_Pass_last,Sav
e_Pass_s_st_last,Save_Pass_s_t_n_last,Save
_Pass_s_psum_last);
        Save_Pass_last:=Save_Pass_last+1;
        Save_Pass_s_st_last:=Save_Pass_st;

Save_Pass_s_t_n_last:=Save_Pass_terminal_
num;

Save_Pass_s_psum_last:=Save_Pass_psum;
        END PROCEDURE

        TYPE arr_pid IS
        PId ARRAY OF integer.
        TYPE arr IS
        integer ARRAY OF PId.
        PR VAR SENDER_PID OF PId.
        PR VAR i OF integer.
```

```
        PR VAR last OF integer.
        PR VAR terminal_count OF integer.
        PR VAR terminal_num OF integer.
        PR VAR MAX_TERMINAL OF inte-
ger.
        PR VAR MAX_PASSENGER OF in-
teger.
        PR VAR st OF integer.
        PR VAR psum OF integer.
        PR VAR count OF integer.
        PR VAR pid_to_idx OF arr_pid.
        PR VAR s_st OF arr_int.
        PR VAR s_t_n OF arr_int.
        PR VAR s_psum OF arr_int.
        PR VAR terminal_busy OF arr_int.
        PR VAR terminal_inst OF arr.

        TRANSITION Start__rl
        EXE count:=0;
        last:=0;
        terminal_count:=0;
        MAX_TERMINAL:=3;
        terminal_busy[1]:=2;
        terminal_busy[2]:=2;
        terminal_busy[3]:=2;
        terminal_inst[1]:=Null;
        terminal_inst[2]:=Null;
        terminal_inst[3]:=Null;
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__rl
        READ    init_terminal(SENDER_PID)
FROM buffer2queue_ch
        FROM NOW TO INF
        JUMP look__X1.

        TRANSITION look__X1
        EXE                              termi-
nal_count:=terminal_count+1;

pid_to_idx[SENDER_PID]:=terminal_count;
        termi-
nal_inst[terminal_count]:=SENDER_PID;
        termi-
nal_num:=pid_to_idx[SENDER_PID];
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__rl
        READ  ready(SENDER_PID)  FROM
buffer2queue_ch
        FROM NOW TO INF
        JUMP look__X2.

        TRANSITION look__X2
        EXE i:=1;
        termi-
nal_num:=pid_to_idx[SENDER_PID];
        terminal_busy[terminal_num]:=0;
        FROM NOW TO NOW
        JUMP Loop__rl.

        TRANSITION Loop__rl
        EXE SKIP
        FROM NOW TO NOW
        JUMP begin_decision__1.

        TRANSITION begin_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__1.

        TRANSITION look__1
        WHEN (i<=last)EXE SKIP
        FROM NOW TO NOW
        JUMP begin_decision__2.

        TRANSITION begin_decision__2
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__2.

        TRANSITION look__2
        WHEN   (s_t_n[i]=terminal_num)EXE
st:=s_st[i];
        psum:=s_psum[i];
        FROM NOW TO NOW
        JUMP Loop2__rl.

        TRANSITION Loop2__rl
        EXE SKIP
        FROM NOW TO NOW
        JUMP begin_decision__3.

        TRANSITION begin_decision__3
        EXE SKIP
        FROM NOW TO NOW
```

```
        JUMP look__3.

        TRANSITION look__3
        WHEN (i<last)EXE CALL
Sdvig(s_st[i],s_t_n[i],s_psum[i],s_st[i+1],s_t
_n[i+1],s_psum[i+1]);
        i:=i+1;
        FROM NOW TO NOW
        JUMP Loop2__rl.

        TRANSITION look__3
        WHEN (NOT (i<last))EXE last:=last-
1;
        s_st[i]:=0;
        s_t_n[i]:=0;
        s_psum[i]:=0;
        FROM NOW TO NOW
        JUMP end_decision__3.

        TRANSITION end_decision__3
        CREATE PROCESS passenger
        FROM NOW TO NOW
        JUMP look__3__X3.

        TRANSITION look__3__X3
        EXE termi-
nal_busy[terminal_num]:=1;
        FROM NOW TO NOW
        JUMP look__3__X4.

        TRANSITION look__3__X4
        WRITE
init(st,terminal_inst[terminal_num],terminal_
num,psum,OFFSPRING) INTO
queue2passenger_ch
        FROM NOW TO NOW
        JUMP look__3__X5.

        TRANSITION look__3__X5
        EXE count:=count+1;
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__2
        WHEN (NOT
(s_t_n[i]=terminal_num))EXE i:=i+1;
        FROM NOW TO NOW
        JUMP Loop__rl.

        TRANSITION end_decision__2
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION look__1
        WHEN (NOT (i<=last))EXE SKIP
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION end_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION look__rl
        READ
init_pas(st,terminal_num,psum,SENDER_PI
D) FROM buffer2queue_ch
        FROM NOW TO INF
        JUMP begin_decision__4.

        TRANSITION begin_decision__4
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__4.

        TRANSITION look__4
        WHEN (termi-
nal_inst[terminal_num]=Null)EXE CALL
Save_Pass(st,psum,terminal_num,last,s_st[la
st],s_t_n[last],s_psum[last]);
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__2
        WHEN (NOT (termi-
nal_inst[terminal_num]=Null))EXE SKIP
        FROM NOW TO NOW
        JUMP begin_decision__5.

        TRANSITION begin_decision__5
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__5.

        TRANSITION look__5
```

```
        WHEN (terminal_busy[terminal_num]=0)CREATE PROCESS passenger
        FROM NOW TO NOW
        JUMP look__5__X6.

        TRANSITION look__5__X6
        WRITE init(st,terminal_inst[terminal_num],terminal_num,psum,OFFSPRING) INTO queue2passenger_ch
        FROM NOW TO NOW
        JUMP look__5__X7.

        TRANSITION look__5__X7
        EXE count:=count+1;
        terminal_busy[terminal_num]:=1;
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__5
        WHEN (NOT (terminal_busy[terminal_num]=0))EXE CALL Save_Pass(st,psum,terminal_num,last,s_st[last],s_t_n[last],s_psum[last]);
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION end_decision__5
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_decision__4.

        TRANSITION end_decision__4
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION look__rl
        READ kill_pas(terminal_num,SENDER_PID) FROM buffer2queue_ch
        FROM NOW TO INF
        JUMP look__X8.

        TRANSITION look__X8
        EXE count:=count-1;
        FROM NOW TO NOW
        JUMP look__rl.

    END; /* queue */

    main_machine: PROCESS
     TYPE int_arr IS
      integer ARRAY OF integer.
     PR VAR SENDER_PID OF PId.
     PR VAR st OF integer.
     PR VAR summ OF integer.

     TRANSITION Start__rl
     EXE expences[1]:=10;
     expences[2]:=15;
     expences[3]:=20;
     ticket_t[1]:=2;
     ticket_t[2]:=3;
     ticket_t[3]:=4;
     FROM NOW TO NOW
     JUMP main__rl.

     TRANSITION main__rl
     READ station(st,SENDER_PID) FROM buffer2main_machine_ch
     FROM NOW TO INF
     JUMP begin_decision__1.

     TRANSITION begin_decision__1
     EXE SKIP
     FROM NOW TO NOW
     JUMP main__1.

     TRANSITION main__1
     WHEN (ticket_t[st]=0)EXE SKIP
     FROM NOW TO NOW
     JUMP main__1__X1.

     TRANSITION main__1__X1
     WRITE no_ticket(SENDER_PID) INTO main_machine2terminal_ch
     FROM NOW TO NOW
     JUMP main__rl.

     TRANSITION main__1
     WHEN (NOT (ticket_t[st]=0))EXE ticket_t[st]:=ticket_t[st]-1;
     FROM NOW TO NOW
     JUMP main__1__X2.

     TRANSITION main__1__X2
```

```
        WRITE
sum(expences[st],SENDER_PID)        INTO
main_machine2terminal_ch
        FROM NOW TO NOW
        JUMP main__rl.

        TRANSITION end_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION main__rl
        READ          return(st,SENDER_PID)
FROM buffer2main_machine_ch
        FROM NOW TO INF
        JUMP main__X3.

        TRANSITION main__X3
        EXE ticket_t[st]:=ticket_t[st]+1;
        FROM NOW TO NOW
        JUMP main__rl.
       END; /* main_machine */

       passenger: PROCESS(0,10)
        PR VAR SENDER_PID OF PId.
        PR VAR terminal_pid OF PId.
        PR VAR terminal_num OF integer.
        PR VAR psum OF integer.
        PR VAR summ OF integer.
        PR VAR input_summ OF integer.
        PR VAR nom OF integer.
        PR VAR st OF integer.
        PR VAR change_summ OF integer.
        PR VAR station OF integer.
        PR VAR is_terminal_ready OF inte-
ger.

        TRANSITION Start__rl
        EXE nom:=1;
        FROM NOW TO NOW
        JUMP init_pas__rl.

        TRANSITION init_pas__rl
        READ
init(st,terminal_pid,terminal_num,psum,SEN
DER_PID) FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP init_pas__X1.

        TRANSITION init_pas__X1
        EXE wants_ticket[SELF]:=st;
        FROM NOW TO NOW
        JUMP init_pas__X2.

        TRANSITION init_pas__X2
        WRITE  press_start(terminal_pid)  IN-
TO passenger2terminal_ch
        FROM NOW TO NOW
        JUMP wait_terminal__rl.

        TRANSITION wait_terminal__rl
        READ  ready_for_job(SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP wait_terminal__X1.

        TRANSITION wait_terminal__X1
        WRITE  station(st,terminal_pid)  INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__rl
        READ      light(summ,SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP begin_decision__1.

        TRANSITION begin_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__1.

        TRANSITION look__1
        WHEN (summ<=0)EXE SKIP
        FROM NOW TO NOW
        JUMP look__1__X1.

        TRANSITION look__1__X1
        WRITE    done(terminal_pid)    INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION look__1
        WHEN (NOT (summ<=0))EXE SKIP
        FROM NOW TO NOW
        JUMP begin_decision__2.
```

```
        TRANSITION begin_decision__2
        EXE SKIP
        FROM NOW TO NOW
        JUMP look__2.

        TRANSITION look__2
        WHEN (psum<summ)EXE SKIP
        FROM NOW TO NOW
        JUMP look__2__X2.

        TRANSITION look__2__X2
        WRITE return(st,terminal_pid) INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP get_back__rl.

        TRANSITION get_back__rl
        READ
change(change_summ,SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP get_back__X1.

        TRANSITION look__2
        WHEN (NOT (psum<summ))EXE
psum:=psum-summ;
        FROM NOW TO NOW
        JUMP look__2__X3.

        TRANSITION look__2__X3
        WRITE coin(summ,terminal_pid) IN-
TO passenger2terminal_ch
        FROM NOW TO NOW
        JUMP look__rl.

        TRANSITION end_decision__2
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_decision__1.

        TRANSITION end_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION look__rl
        READ        no_ticket(SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP look__X4.

        TRANSITION look__X4
        WRITE    done(terminal_pid)    INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP look__X5.

        TRANSITION look__X5
        WRITE    kill_pas(terminal_num,Null)
INTO passenger2queue_ch
        FROM NOW TO NOW
        JUMP no_ticket__rl.

        TRANSITION no_ticket__rl
        EXE SKIP
        FROM NOW TO NOW
        JUMP no_ticket__stop__rl.

        TRANSITION look__rl
        READ          success(SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP get_change__rl.

        TRANSITION get_change__rl
        READ
change(change_summ,SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP get_change__X1.

        TRANSITION get_back__X1
        EXE psum:=psum+change_summ;
        FROM NOW TO NOW
        JUMP get_back__X2.

        TRANSITION get_back__X2
        WRITE    done(terminal_pid)    INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP get_back__X3.

        TRANSITION get_back__X3
        WRITE    kill_pas(terminal_num,Null)
INTO passenger2queue_ch
        FROM NOW TO NOW
        JUMP bad__rl.
```

```
        TRANSITION bad__rl
        EXE SKIP
        FROM NOW TO NOW
        JUMP bad__stop__rl.

        TRANSITION get_change__X1
        EXE psum:=psum+change_summ;
        FROM NOW TO NOW
        JUMP get_ticket__rl.

        TRANSITION get_ticket__rl
        READ    ticket(station,SENDER_PID)
FROM buffer2passenger_ch
        FROM NOW TO INF
        JUMP get_ticket__X1.

        TRANSITION get_ticket__X1
        EXE gotten_ticket[SELF]:=station;
        FROM NOW TO NOW
        JUMP get_ticket__X2.

        TRANSITION get_ticket__X2
        WRITE    done(terminal_pid)    INTO
passenger2terminal_ch
        FROM NOW TO NOW
        JUMP get_ticket__X3.

        TRANSITION get_ticket__X3
        WRITE    kill_pas(terminal_num,Null)
INTO passenger2queue_ch
        FROM NOW TO NOW
        JUMP good__rl.

        TRANSITION good__rl
        EXE SKIP
        FROM NOW TO NOW
        JUMP good__stop__rl.

        TRANSITION no_ticket__stop__rl
        WHEN TRUE
        STOP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION good__stop__rl
        WHEN TRUE
        STOP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION bad__stop__rl
        WHEN TRUE
        STOP
        FROM NOW TO NOW
        JUMP end_of_process.
       END; /* passenger */

       terminal: PROCESS(5,5)
        PR VAR SENDER_PID OF PId.
        PR VAR passenger_pid OF PId.
        PR VAR nom OF integer.
        PR VAR st OF integer.
        PR VAR station_price OF integer.
        PR VAR input_summ OF integer.

        TRANSITION Start__rl
        WRITE    init_terminal(Null)    INTO
terminal2queue_ch
        FROM NOW TO NOW
        JUMP Start__X1.

        TRANSITION Start__X1
        WRITE    ready(Null)    INTO    termi-
nal2queue_ch
        FROM NOW TO NOW
        JUMP main__rl.

        TRANSITION main__rl
        READ    press_start(SENDER_PID)
FROM buffer2terminal_ch
        FROM NOW TO INF
        JUMP main__X1.

        TRANSITION main__X1
        EXE st:=0;
        input_summ:=0;
        FROM NOW TO NOW
        JUMP main__X2.

        TRANSITION main__X2
        WRITE
ready_for_job(SENDER_PID)    INTO    termi-
nal2passenger_ch
        FROM NOW TO NOW
        JUMP wait_for_station__rl.

        TRANSITION wait_for_station__rl
```

```
READ station(st,SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP wait_for_station__X1.

TRANSITION wait_for_station__X1
EXE passenger_pid:=SENDER_PID;
FROM NOW TO NOW
JUMP wait_for_station__X2.

TRANSITION wait_for_station__X2
WRITE station(st,Null) INTO terminal2main_machine_ch
FROM NOW TO NOW
JUMP sum_or_ticket__rl.

TRANSITION sum_or_ticket__rl
READ sum(station_price,SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP sum_or_ticket__X1.

TRANSITION sum_or_ticket__X1
WRITE light(station_price,passenger_pid) INTO terminal2passenger_ch
FROM NOW TO NOW
JUMP wait_for_coin__rl.

TRANSITION wait_for_coin__rl
READ coin(nom,SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP wait_for_coin__X1.

TRANSITION sum_or_ticket__rl
READ no_ticket(SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP sum_or_ticket__X2.

TRANSITION sum_or_ticket__X2
WRITE no_ticket(passenger_pid) INTO terminal2passenger_ch
FROM NOW TO NOW
JUMP satisfaction__rl.

TRANSITION satisfaction__rl
READ done(SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP satisfaction__X1.

TRANSITION wait_for_coin__X1
EXE input_summ:=input_summ+nom;
FROM NOW TO NOW
JUMP wait_for_coin__X2.

TRANSITION wait_for_coin__X2
WRITE light(station_price-input_summ,passenger_pid) INTO terminal2passenger_ch
FROM NOW TO NOW
JUMP wait_for_coin__rl.

TRANSITION wait_for_coin__rl
READ done(SENDER_PID) FROM buffer2terminal_ch
FROM NOW TO INF
JUMP begin_decision__1.

TRANSITION begin_decision__1
EXE SKIP
FROM NOW TO NOW
JUMP wait_for_coin__1.

TRANSITION wait_for_coin__1
WHEN (input_summ<station_price)EXE SKIP
FROM NOW TO NOW
JUMP wait_for_coin__1__X3.

TRANSITION wait_for_coin__1__X3
WRITE light(station_price-input_summ,passenger_pid) INTO terminal2passenger_ch
FROM NOW TO NOW
JUMP wait_for_coin__rl.

TRANSITION wait_for_coin__1
WHEN (NOT (input_summ<station_price))EXE SKIP
FROM NOW TO NOW
JUMP wait_for_coin__1__X4.

TRANSITION wait_for_coin__1__X4
```

```
        WRITE success(passenger_pid) INTO
terminal2passenger_ch
        FROM NOW TO NOW
        JUMP wait_for_coin__1__X5.

        TRANSITION wait_for_coin__1__X5
        WRITE change(input_summ-
station_price,passenger_pid) INTO termi-
nal2passenger_ch
        FROM NOW TO NOW
        JUMP wait_for_coin__1__X6.

        TRANSITION wait_for_coin__1__X6
        WRITE ticket(st,passenger_pid) INTO
terminal2passenger_ch
        FROM NOW TO NOW
        JUMP satisfaction__rl.

        TRANSITION end_decision__1
        EXE SKIP
        FROM NOW TO NOW
        JUMP end_of_process.

        TRANSITION wait_for_coin__rl
        READ return(st,SENDER_PID)
FROM buffer2terminal_ch
        FROM NOW TO INF
        JUMP wait_for_coin__X7.

        TRANSITION wait_for_coin__X7
        WRITE
change(input_summ,passenger_pid) INTO
terminal2passenger_ch
        FROM NOW TO NOW
        JUMP wait_for_coin__X8.

        TRANSITION wait_for_coin__X8
        WRITE return(st,Null) INTO termi-
nal2main_machine_ch
        FROM NOW TO NOW
        JUMP satisfaction__rl.

        TRANSITION satisfaction__X1
        WRITE ready(Null) INTO termi-
nal2queue_ch
        FROM NOW TO NOW
        JUMP main__rl.
       END; /* terminal */


     END; /* Passengers_and_Terminals */
      END; /* All */
```

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

ПРИМЕНЕНИЕ ЯЗЫКА DYNAMIC-REAL ДЛЯ АНАЛИЗА И ВЕРИФИКАЦИИ РАСПРЕДЕЛЕННЫХ СИСТЕМ, СПЕЦИФИЦИРОВАННЫХ НА ЯЗЫКЕ SDL

Препринт
161

Рукопись поступила в редакцию 12.12.2011
Редактор Т. М. Бульонкова
Рецензент Н.В. Шилов

Подписано в печать 28.12.2011
Формат бумаги 60 × 84 1/16 Объем 3.03 уч.-изд.л., 3.31 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42