МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.2

ПРИЛОЖЕНИЕ D К РЕКОМЕНДАЦИИ Z.100

РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЕЙ ЯЗЫКА SDL

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14 – 25 НОЯБРЯ 1988 ГОДА

МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.2

ПРИЛОЖЕНИЕ D К РЕКОМЕНДАЦИИ Z.100

РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЕЙ ЯЗЫКА SDL

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14 – 25 НОЯБРЯ 1988 ГОДА

ISBN 92-61-03764-X

СОДЕРЖАНИЕ КНИГИ МККТТ, ДЕЙСТВУЮЩЕЙ ПОСЛЕ IX ПЛЕНАРНОЙ АССАМБЛЕИ (1988 г.)

СИНЯЯ КНИГА

ВЫПУСК III.6 — Передача по линии нетелефонных сигналов. Передача сигналов звукового и телевизионного вещания. Рекомендации серий H и J (Исследовательская комиссия XV).

ВЫПУСК III.7 — Цифровая сеть с интеграцией служб (ЦСИС) — Общая структура и возможности служб. Рекомендации I.110—I.257 (Исследовательская комиссия XVIII).

ВЫПУСК III.8 — Цифровая сеть с интеграцией служб (ЦСИС) — Общесетевые аспекты и функции, стыки пользователь — сеть ЦСИС. Рекомендации I.310—I.470 (Исследовательская комиссия XVIII).

ВЫПУСК III.9 — Цифровая сеть с интеграцией служб (ЦСИС)—Межсетевые стыки и принципы технической эксплуатации. Рекомендации I.500—I.605 (Исследовательская комиссия XVIII).

Том IV

ВЫПУСК IV.1 — Общие принципы технической эксплуатации; техническая эксплуатация международных систем передачи и международных телефонных каналов. Рекомендации M.10—M.782 (Исследовательная комиссия IV).

ВЫПУСК IV.2 — Техническая эксплуатация международных телеграфных, фототелеграфных и арендованных каналов. Техническая эксплуатация международной телефонной сети общего пользования. Техническая эксплуатация морских спутниковых систем и систем передачи данных. Рекомендации M.800—M.1375 (Исследовательская комиссия IV).

ВЫПУСК IV.3 — Техническая эксплуатация международных каналов звукового и телевизионного вещания. Рекомендации серии N (Исследовательская комиссия IV).

ВЫПУСК IV.4 — Требования к измерительному оборудованию. Рекомендации серии O (Исследовательская комиссия IV).

Том V — Качество телефонной передачи. Рекомендации серии P (Исследовательская комиссия XII).

Том VI

ВЫПУСК VI.1 — Общие Рекомендации по телефонной коммутации и сигнализации. Функции и информационные потоки для служб в ЦСИС. Дополнения. Рекомендации Q.1—Q.118 bis (Исследовательская комиссия XI).

ВЫПУСК VI.2 — Требования к системам сигнализации № 4 и № 5. Рекомендации Q.120—Q.180 (Исследовательская комиссия XI).

ВЫПУСК VI.3 — Требования к системе сигнализации № 6. Рекомендации Q.251—Q.300 (Исследовательская комиссия XI).

ВЫПУСК VI.4 — Требования к системам сигнализации R1 и R2. Рекомендации Q.310—Q.490 (Исследовательская комиссия XI).

ВЫПУСК VI.5 — Цифровые местные, транзитные, комбинированные и международные станции в интегральных цифровых сетях и смешанных аналого-цифровых сетях. Дополнения. Рекомендации Q.500—Q.554 (Исследовательская комиссия XI).

ВЫПУСК VI.6 — Взаимодействие систем сигнализации. Рекомендации Q.601—Q.699 (Исследовательская комиссия XI).

ВЫПУСК VI.7 — Требования к системе сигнализации № 7. Рекомендации Q.700—Q.716 (Исследовательская комиссия XI).

ВЫПУСК VI.8 — Требования к системе сигнализации № 7. Рекомендации Q.721—Q.766 (Исследовательская комиссия XI).

ВЫПУСК VI.9 — Требования к системе сигнализации № 7. Рекомендации Q.771—Q.795 (Исследовательская комиссия XI).

ВЫПУСК VI.10 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), уровень звена данных. Рекомендации Q.920 и Q.921 (Исследовательская комиссия XI).

ВЫПУСК VI.11 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), сетевой уровень, управление пользователь-сеть. Рекомендации Q.930—Q.940 (Исследовательская комиссия XI).

ВЫПУСК VI.12 — Сухопутная подвижная сеть общего пользования. Взаимодействие с ЦСИС и коммутируемой телефонной сетью общего пользования. Рекомендации Q.1000—Q.1032 (Исследовательская комиссия XI).

ВЫПУСК VI.13 — Сухопутная подвижная сеть общего пользования. Подсистема подвижного применения и стыки. Рекомендации Q.1051—Q.1063 (Исследовательская комиссия XI).

ВЫПУСК VI.14 — Взаимодействие со спутниковыми подвижными системами. Рекомендации Q.1100—Q.1152 (Исследовательская комиссия XI).

Том VII

ВЫПУСК VII.1 — Телеграфная передача. Рекомендации серии R. Оконечное оборудование телеграфных служб. Рекомендации серии S (Исследовательская комиссия IX).

ВЫПУСК VII.2 — Телеграфная коммутация. Рекомендации серии U (Исследовательская комиссия IX).

ВЫПУСК VII.3 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.0—T.63 (Исследовательская комиссия VIII).

ВЫПУСК VII.4 — Процедуры испытания на соответствие Рекомендациям по службе телетекс. Рекомендация T.64 (Исследовательская комиссия VIII).

ВЫПУСК VII.5 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.65—T.101, T.150—T.390 (Исследовательская комиссия VIII).

ВЫПУСК VII.6 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.400—T.418 (Исследовательская комиссия VIII).

ВЫПУСК VII.7 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.431—T.564 (Исследовательская комиссия VIII).

Том VIII

ВЫПУСК VIII.1 — Передача данных по телефонной сети. Рекомендации серии V (Исследовательская комиссия XVII).

ВЫПУСК VIII.2 — Сети передачи данных: службы и возможности, стыки. Рекомендации X.1—X.32 (Исследовательская комиссия VII).

ВЫПУСК VIII.3 — Сети передачи данных: передача, сигнализация и коммутация, сетевые аспекты, техническая эксплуатация и административные положения. Рекомендации X.40—X.181 (Исследовательская комиссия VII).

ВЫПУСК VIII.4 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Модель и система обозначений, определение служб. Рекомендации X.200—X.219 (Исследовательская комиссия VII).

ВЫПУСК VIII.5 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Требования к протоколам, аттестационные испытания. Рекомендации X.220—X.290 (Исследовательская комиссия VII).

ВЫПУСК VIII.6 — Сети передачи данных: взаимодействие между сетями, подвижные системы передачи данных, межсетевое управление. Рекомендации X.300—X.370 (Исследовательская комиссия VII).

ВЫПУСК VIII.7 — Сети передачи данных: системы обработки сообщений. Рекомендации X.400—X.420 (Исследовательская комиссия VII).

ВЫПУСК VIII.8 — Сети передачи данных: справочная служба. Рекомендации X.500—X.521 (Исследовательская комиссия VII).

Том IX — Защита от мешающих влияний. Рекомендации серии K (Исследовательская комиссия V). Конструкция, прокладка и защита кабелей и других элементов линейных сооружений. Рекомендации серии L (Исследовательская комиссия VI).

Том X

ВЫПУСК X.1 — Язык функциональных спецификаций и описания (SDL). Критерии применения формальных методов описания (FDT). Рекомендация Z.100 и приложения A, B, C и E, Рекомендация Z.110 (Исследовательская комиссия X).

ВЫПУСК X.2 — Приложение D к Рекомендации Z.100: руководство для пользователей языка SDL (Исследовательская комиссия X).

ВЫПУСК X.3 — Приложение F.1 к Рекомендации Z.100: формальное определение языка SDL. Введение (Исследовательская комиссия X).

ВЫПУСК X.4 — Приложение F.2 к Рекомендации Z.100: формальное определение языка SDL. Статическая семантика (Исследовательская комиссия X).

ВЫПУСК X.5 — Приложение F.3 к Рекомендации Z.100: формальное определение языка SDL. Динамическая семантика (Исследовательская комиссия X).

ВЫПУСК X.6 — Язык МККТТ высокого уровня (CHILL). Рекомендация Z.200 (Исследовательская комиссия X).

ВЫПУСК X.7 — Язык человек-машина (MML). Рекомендации Z.301—Z.341 (Исследовательская комиссия X).

СОДЕРЖАНИЕ ВЫПУСКА X.2 СИНЕЙ КНИГИ

Приложение D к Рекомендации Z.100

Руководство Пользователя языка SDL

ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

1 Вопросы, порученные каждой Исследовательской комиссии на исследовательский период 1989—1992 годов, содержатся в документе № 1 для данной Исследовательской комиссии.

2 В данном выпуске для краткости термин "Администрация" используется для обозначения как администрации связи, так и признанной частной эксплуатационной организации.

ВЫПУСК X.2

Приложение D к Рекомендации Z.100

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ЯЗЫКА SDL

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

ПРИЛОЖЕНИЕ D

(к Рекомендации Z.100)

Руководство Пользователя языка SDL

СОДЕРЖАНИЕ

D.1 *Предисловие*

Язык спецификаций и описаний МККТТ, известный под именем SDL (Specification and Description Language), первоначально был описан в Рекомендациях Z.101 – Z.103 в 1976 году (Оранжевая книга, Том VI.4), позже развит в Рекомендациях Z.101 – Z.104 в 1980 году (Желтая книга) и затем развит и реорганизован в Рекомендациях Z.100 – Z.104 в 1984 году (Красная книга). В исследовательский период 1985–1988 годов язык подвергся дальнейшему развитию и гармонизации, имевшиеся Рекомендации были слиты в одну и было предложено математическое определение.

Для облегчения применения SDL в телекоммуникационных системах широкого диапазона необходимо иметь руководство пользователя. Назначением руководства является облегчение пользователям понимания Рекомендации по SDL и применения SDL в различных областях.

SDL широко применяется МККТТ и его коллективными членами, а диапазон применения SDL непрерывно расширяется. Настоящее руководство пользователя должно помочь всем тем, кто намеревается или приступает к использованию SDL, дополняя Рекомендации по SDL полезными советами и вспомогательными примерами. При этом учитывается, что Рекомендации и руководство пользователя будут в какой-то мере перекрывать друг друга: такое перекрытие можно считать целесообразным, так как обеспечивает независимость и доступность руководства. Однако основным документом является Рекомендация.

D.2 *Введение*

D.2.1 *Общий обзор SDL*

SDL может использоваться как для спецификации требуемого поведения системы, так и для описания ее фактического поведения. Целью разработки SDL является спецификация поведения коммутируемых телекоммуникационных систем, но SDL можно использовать для моделирования и других приложений. Фактически SDL очень удобен для описания систем, чье поведение можно с успехом моделировать с помощью расширенных автоматов с конечным числом состояний (§ D.2.1.1), и, особенно, систем, в которых внимание должно быть сосредоточено на аспектах взаимодействия.

Кроме того, SDL может служить основой метода документирования, преследующего цель полного представления спецификации или описания системы. В этом контексте смысл спецификации и описания связан с их использованием в жизненном цикле системы. Оба эти понятия определяют функциональные свойства системы абстрактным способом. Как правило, описание включает какие-то свойства, связанные с проектированием (например, обработка ошибок), и является более полным в отношении деталей функционирования. Оба понятия должны быть последовательным образом связаны с проектированием конкретной системы и в дальнейшем служить в качестве документации.

SDL может быть использован с различными степенями детализации для представления функциональных свойств систем, или функций, или устройств. Функциональные свойства состоят из некоторых структурных свойств (диаграмма взаимодействия блоков), а также из поведения. При этом под "поведением" понимается реакция на получаемые сигналы (вводы), то есть выполнение действий, например посылка сигналов (выводы), задавание вопросов (принятие решений) и выполнение работ.

Спецификация может быть весьма широкой и общей, если Администрация хочет исследовать возможности обновления системы с помощью новых средств, новых служб, новой технологии и т.д., одновременно предоставляя поставщику возможность предложить широкий диапазон решений в отношении проектирования. Такой тип спецификации часто не бывает детально разработанным. Другим крайним случаем является спецификация, в которой Администрация запрашивает расширение имеющегося коммутатора или его замену. Такой тип спецификации скорее всего будет обладать большей степенью детализации в силу необходимости детальной спецификации интерфейсов.

Спецификация и описание могут быть идентичными. В любом случае в новых разработках предпочтительнее выводить проектирование из спецификации, что обеспечит согласованность.

Обычно описания составляют поставщики в ответ на спецификацию (хотя они могут быть составлены поставщиком для рекламирования системы, которую он хочет продать). Как правило, описания содержат больше уровней детализации, чем спецификации, в силу необходимости привести детализованную характеристику поведения системы. В целях упрощения представление с помощью SDL в последующих разделах называется спецификацией.

Следует также отметить, что SDL предоставляет возможность описания системы с разной степенью формализованности.

Во-первых, можно составить описание системы, используя совместно конструкции SDL и естественные языки. В результате получается спецификация, с помощью которой читателю сообщается информация, касающаяся контекста описания, но машине эта информация будет недоступна. Только очень небольшое число проверок может быть выполнено автоматически.

Во-вторых, с конструкциями SDL могут быть связаны формальные предложения, состоящие из элементов заранее определенных типов и операторов над этими элементами. Свойства этих элементов не специфицируются; примером может служить предложение "Соединить A–B", где A и B типа "абонент", а "соединить" – это операция, допустимая для элементов такого типа. В результате получается спецификация, сообщающая информацию тем читателям, которым известны значения используемых операторов. Машина может понять такую спецификацию лишь до некоторой степени и выполнить некоторые проверки, но она не может ни выполнить полной проверки, ни "реализовать" такую систему, так как свойства операторов не специфицированы полностью.

В-третьих, кроме всего сказанного, можно предоставить описание всех свойств всех операторов. В последнем случае спецификация становится полностью формальной и машина может осуществить все проверки и концептуальную реализацию описанных систем.

В зависимости от преследуемых целей спецификация может быть приспособлена к нуждам пользователя путем применения этих трех уровней формализации. Конечно, чем более формализована спецификация, тем труднее человеку читать ее.

Ниже термин "спецификация" используется для представления как требуемого, так и фактического поведения.

D.2.1.1 *SDL опирается на модель Расширенной машины с конечным числом состояний*

В приложениях SDL специфицируемая система представляется некоторым числом взаимосвязанных абстрактных машин. Для полной спецификации требуется:

1) спецификация структуры системы в терминах машин и их взаимосвязь;

2) описание динамического поведения каждой машины в терминах ее взаимодействия с другими машинами и окружающей средой; и

3) описание операций над данными, связанными с взаимодействиями.

Динамическое поведение описывается с помощью моделей, определяющих механизмы функционирования абстрактных машин и взаимосвязь между ними. Абстрактные машины, используемые в SDL, являются расширениями детерминированных Машин с Конечным Числом Состояний (МКЧС). МКЧС обладает конечной внутренней памятью для хранения состояний; ее функционирование связано с наличием конечного набора дискретных вводов и выводов. Для каждой комбинации состояния и ввода память определяет вывод и следующее состояние. Считается, что переход из одного состояния в другое занимает нулевой промежуток времени.

Ограниченность МКЧС состоит в том, что вся требуемая для хранения информация должна быть представлена в виде явных состояний. Хотя большинство систем может быть представлено таким способом, это не всегда практически удобно. Может потребоваться хранение большого числа значений, которые существенны для дальнейшего поведения, но не помогают в значительной мере пониманию системы в целом. Такая информация не должна являться частью пространства явных состояний, так как это "засоряет" общее представление о системе. Для таких приложений МКЧС могут быть расширены за счет вспомогательной памяти и вспомогательных операций над этой памятью. Адресная информация и последовательные номера являются примерами информации, которую удобно хранить во вспомогательной памяти.

Рекомендация по SDL определяет две вспомогательные операции, которые могут быть включены в переходы Расширенных Машин с Конечным Числом Состояний (РМКЧС), а именно "решения" и "работы". "Решения" проверяют параметры, связанные с вводами, и информацию во вспомогательной памяти, если эта информация существенна для упорядочения функционирования главной машины. "Работы" выполняют такие функции, как подсчет, операции над вспомогательной памятью и манипулирование с вводимыми и выводимыми параметрами.

В SDL взаимодействия между машинами представляются в виде сигналов, то есть РМКЧС получает сигналы в качестве вводов и вырабатывает сигналы в качестве выводов. Сигналы состоят из уникальных идентификаторов сигналов и, возможно, набора параметров. SDL допускает возможность перехода, длящегося в течение ненулевого отрезка времени, и определяет дисциплину "первым в – первым из" в качестве концептуального механизма образования очереди сигналов, поступающих в машину за время совершения ею перехода. Сигналы рассматриваются по одному в каждый данный момент времени в порядке их поступления.

D.2.2 *Синтаксические формы SDL*

Язык SDL имеет две разные синтаксические формы, опирающиеся на одну и ту же семантическую модель. Одна называется SDL/GR (SDL Graphical Representation – Графическое представление SDL) и основывается на наборе стандартизованных графических символов. Другая называется SDL/PR (SDL textual Phrase Representation – Представление SDL текстовыми фразами) и основывается на программно-подобных предложениях. Обе эти формы представляют одни и те же концепции SDL.

Преимущество графического представления в том, что оно явно изображает структуру системы и позволяет человеку легко видеть управляющий поток. Представление в виде фраз гораздо больше подходит к использованию на машинах.

Как средство проектирования SDL должен иметь форму, которая помогает пользователю ясно и кратко выразить свои идеи. SDL/GR вполне отвечает этим требованиям и более соответствует традиционным методам изображения расширенных машин с конечным числом состояний.

Первоначальной формой SDL был SDL/GR. Он был разработан в период 1973 – 1976 годов и обнародован в версии 1976 года в Рекомендациях серии Z.100.

SDL/GR был составлен из различных графических языков, разработанных различными организациями для их внутренних нужд.

Представление SDL в виде фраз, SDL/PR, было разработано в исследовательский период 1977–1980 годов, но прежде чем его рекомендовать пользователям требовалось выполнить некоторые уточнения. Эти уточнения были осуществлены в следующий исследовательский период, и с 1984 года SDL/PR стал одним из рекомендуемых конкретных синтаксисов SDL.

Первоначально SDL/PR был задуман как простое средство ввода в машину документации по SDL, так как ввод SDL/GR гораздо сложнее. (Для этого требуется наличие графических внешних устройств.) Поэтому основное внимание было уделено возможности взаимно-однозначного отображения между PR и GR. Развитие графических терминалов (рост возможностей и понижение стоимости) привели к тому, что GR стал удобен для ввода в машину. Это не уменьшает значения PR для использования, так как некоторые пользователи считают его более соответствующим их требованиям, особенно те из них, которые работают с языками программирования.

Эта эволюция привела к ослаблению согласованности между GR и PR, однако и сейчас мы можем (легко) отобразить один из них на другой, но каждая из форм имеет свои особенности. На первый взгляд, PR во многом схож с языками программирования (см. рис. D-2.2.1).

```
-
-
STATE ожидание_снятия_трубки;
INPUT снять_трубку;
TASK 'активизировать_начисление_платы';
TASK 'соединить';
OUTPUT сбросить_таймер;
NEXTSTATE диалог;
-
-
```

РИСУНОК D-2.2.1

Пример SDL/PR

В действительности это зависит от того, что́ характеризует текст как язык программирования.

Если мы примем, что программа определяется как "информация, интерпретируемая машиной", то тогда не только PR, но и GR является "программой".

Однако существует некоторая разница между спецификацией на SDL и фактическими программами. Во-первых, для спецификации на SDL не имеет существенного значения возможность ее выполнения машиной (хотя это и не исключено); существенной является возможность передачи точной информации от одного человека к другому.

Рассматривая спецификацию на SDL как программу, придется счесть какую-то спецификацию "за плохую спецификацию на SDL" (в силу неполной формализованности текста), в то время как с точки зрения представления функциональных требований системы та же спецификация может считаться вполне корректным SDL.

Еще одно отличие спецификаций на SDL при их сравнении с обычным представлением программ заключается в "стиле" спецификаций.

Так как целью SDL является облегчение связи между людьми, то было уделено особое внимание возможности различного размещения изображения, с тем чтобы помочь читателю сосредоточиться на некоторых аспектах, рассматриваемых как более важные по сравнению с другими. Это, конечно, для программ не существенно, так как они рассчитаны на интерпретацию машинами. Машина не сосредоточивается на каких-то конкретных аспектах, а рассматривает их все на равных основаниях; точно так же машина не старается "понять" программу.

Некоторые программисты предпочитают PR в силу его схожести с программами; возможно, что для реализации требований они используют CHILL. Поэтому ощущается сильная потребность в установлении взаимно-однозначного соответствия между PR и CHILL; это позволит автоматически преобразовать требования, выраженные на PR, в код CHILL. Обратное также представляет интерес, так как позволило бы получить спецификацию на PR из программы на CHILL.

В § D.9 приведены примеры возможных путей отображения SDL на CHILL.

D.2.3 *Область применения SDL*

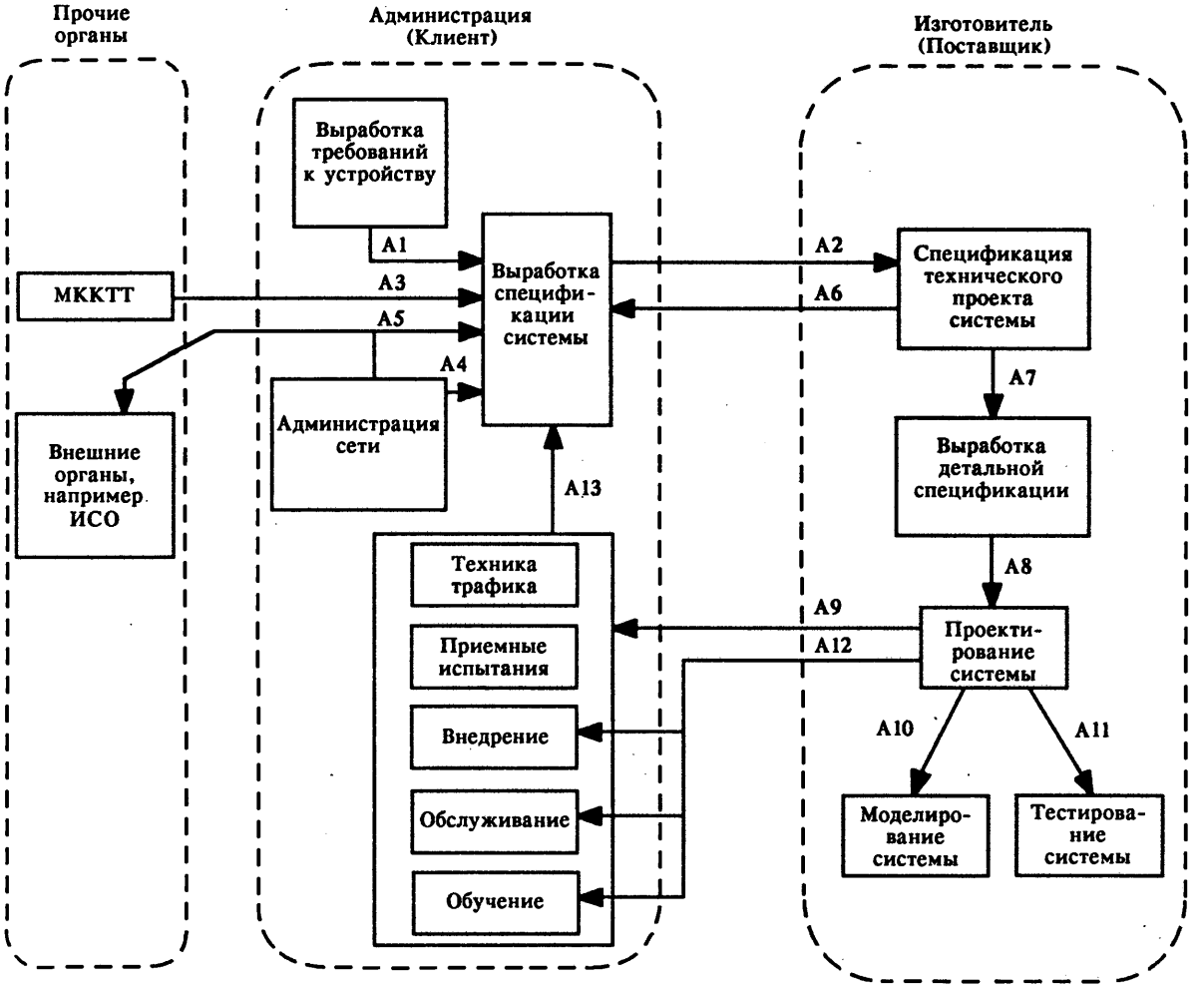
На рис. D-2.3.1 представлен диапазон возможных пользователей SDL в контексте приобретения и поставки коммутируемых телекоммуникационных систем.

Квадратами на этом рисунке изображены типичные функциональные подразделения, точные названия которых могут меняться в зависимости от организации, но деятельность которых типична для многих Администраций и изготовителей. Каждая из стрелок (линий потока) изображает комплект документов, передаваемых от одного функционального подразделения другому; SDL может использоваться как часть каждого из этих комплектов документов. Этот рисунок должен рассматриваться как чисто иллюстративный; он не является ни определяющим, ни исчерпывающим.

Область применения SDL состоит из тех приложений, которые эффективно моделируются сообщающимися Расширенными Машинами с Конечным Числом Состояний; например, функции телефонных и телексных систем, систем коммутации данных, систем сигнализации (например, Система Сигнализации № 7), взаимодействие систем сигнализации и протоколов данных, интерфейсы пользователей (MML).

В частности, для коммутаторов с управлением от записанной программы примерами функций, допускающих документирование с помощью SDL, являются: процесс вызова (например, обработка вызова, маршрутизация, сигнализация, замерения и т.д.), сопровождение программы и обработка ошибок (например, чрезвычайные случаи, автоматическое исправление ошибок, конфигурация системы, тестирование маршрутов и т.д.), управление системой (например, управление перегрузками) и интерфейс человек-машина. Примеры использования SDL приведены в § D.10.

Спецификация протоколов с помощью SDL рассматривается в Рекомендациях серии X МККТТ.



A1 Спецификация устройства или свойства, не зависящая от реализации и сети
A2 Не зависящая от реализации, но зависящая от сети спецификация устройства, включающая описание окружающей среды системы
A3 Рекомендации и руководства МККТТ
A4 Добавления к спецификации системы, отражающие требования администрации сети и функциональные требования
A5 Прочие Рекомендации, относящиеся к данному вопросу
A6 Описание предложения на реализацию
A7 Спецификация проекта
A8 Детальная спецификация проекта
A9 Полное описание системы
A10 Документация описания системы и окружающей среды, требующаяся для моделирования системы
A11 Документация описания системы и окружающей среды, требующаяся для тестирования системы
A12 Руководства по установке и обслуживанию
A13 Добавления к спецификации системы, представляемые специализированными функциональными подразделениями, входящими в Администрацию

Примечание 1. – На всех уровнях возможна итерация.

Примечание 2. – В некоторых случаях документация на SDL, изображенная здесь как внутренняя для одной организации, например A1, A7, A8, может быть поставлена другой организации.

РИСУНОК D-2.3.1

Общий сценарий использования SDL

D.3 *Основные концепции SDL*

D.3.1 *Система*

Как было отмечено выше, SDL моделирует системы. Таким образом, система – это то, что определяется спецификацией на SDL. Как таковая SDL-система может моделировать часть телефонной системы (или АТС) или полную сеть телефонных систем, или некоторую часть совокупности АТС (например, контролеры магистрали в обоих концах магистрали). Основным здесь является то, что с точки зрения SDL любая SDL-система содержит все, что пытаются определить с помощью спецификации. Особенности реализации, такие как дублирование систем или кратность эквивалентных узлов, не могут быть явно смоделированы на SDL. Точно так же окружающая среда находится за пределами спецификации и не может быть определена на SDL.

Интерфейс системы с окружающей средой осуществляется посредством каналов. Теоретически для осуществления интерфейса с окружающей средой достаточно иметь только один двусторонний канал. Однако на практике каналы обычно определяются для каждого логического интерфейса с окружающей средой.

Каждая система состоит из нескольких блоков, связанных между собой каналами. Каждый блок системы независим от любого другого блока. Каждый блок может содержать один или несколько процессов, описывающих поведение блока. Единственным средством связи между процессами, находящимися в двух различных блоках, является посылка сигналов, транспортируемых каналами. Критериями разбиения системы на блоки могут быть: представление системы в виде небольших, легко управляемых частей, установление соответствия с фактической структурой аппаратуры/программного обеспечения, следование естественному функциональному подразделению, минимизация взаимодействий и пр.

Для больших SDL-систем в SDL имеются конструкции, обеспечивающие спецификацию подструктур отдельных частей системы; благодаря этому, исходя из общего обзора системы, можно вводить все большее число деталей. В таких случаях мы говорим, что система представлена на различных уровнях детализации. Эти конструкции разъясняются в § D.4.

На первом уровне детализации SDL-спецификация системы описывает структуру системы; это описание содержит нижеследующие компоненты, которые будут разъяснены в последующих параграфах:

– Имя системы.

– Определения сигналов: спецификация типов сигналов, обмениваемых между блоками системы или между блоками и окружающей средой. Включают спецификацию типов значений, передаваемых сигналами (список сортов).

– Определения списков сигналов: спецификация идентификаторов, объединяющих несколько сигналов и/или другие списки сигналов. Такие идентификаторы могут использоваться для экономии места и обеспечения более ясной спецификации.

– Определения каналов: спецификация каналов, связывающих блоки системы друг с другом и с окружающей средой. Определение канала содержит спецификацию идентификаторов сигналов, транспортируемых по этому каналу.

– Определения данных: спецификация определенных пользователем новых типов, синонимичных типов и генераторов, доступных во всех блоках.

– Определения блоков: спецификация блоков, на которые разбита система.

– Определения макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

В соответствии с Рекомендацией по SDL существуют предварительно определенные типы данных, доступные в любой системе. Они не требуют определения и могут использоваться с помощью присвоенных им имен; а именно: INTEGER, REAL, CHARACTER, STRING, CHARSTRING, BOOLEAN, PID, TIME, DURATION. Предварительно определенные типы данных доступны на любом уровне определения системы: можно считать, что они неявно определены в библиотеке системы, доступной в любом месте спецификации.

Имена сортов, используемые в определениях сигналов на уровне системы, должны быть введены частичными определениями типа, доступными на уровне системы, то есть либо предварительно определенными типами данных, либо определенными на этом уровне пользователем новыми или синонимичными типами.

Дальнейшие объяснения использования типов данных изложены в § D.3.10 и § D.6.

Определение системы на SDL/PR содержит набор предложений, каждый из которых завершается знаком"; " (точкой с запятой). Определение структуры системы начинается предложением "SYSTEM имя; " и заканчивается предложением "ENDSYSTEM имя; ". Имя в замыкающем предложении не обязательно, но если оно приведено, то должно совпадать с именем, указанным после ключевого слова SYSTEM. Рекомендуется всегда использовать имя в замыкающем предложении, так как это повышает доступность документа.

Схема определения структуры системы на SDL/PR приведена на рис. D-3.1.1

```
SYSTEM . . . ;
    . . . определения сигналов . . .
    . . . список определений сигналов . . .
    . . . определения каналов . . .
    . . . определения данных . . .
    . . . определения блоков . . .
    . . . определения макрокоманд . . .
ENDSYSTEM . . . ;
```

РИСУНОК D-3.1.1

Схема определения структуры системы на SDL/PR

Чтобы получить более ясное и простое представление структуры системы и чтобы обеспечить спецификацию системы методом сверху вниз, в SDL предусмотрен общий механизм ссылок. На данном уровне механизм ссылок может быть применен к определению блоков. Это свойство языка позволяет пользователю при определении структуры системы специфицировать только имя блока; фактическое определение блока может быть приведено отдельно. (См. рис. D-3.1.2.)

```
SYSTEM s;
    . . .
    BLOCK b1 REFERENCED;
    BLOCK b2 REFERENCED;
    . . .
ENDSYSTEM s;
```

РИСУНОК D-3.1.2

Пример ссылок на определение блоков

Механизм ссылок особенно используется в SDL/GR, так как желательно помещать большинство диаграмм на отдельных листах, но очень часто не хватает места для вложенных графических спецификаций.

Примеры определения системы на SDL/GR приведены в § D.3.6.

D.3.2 *Блоки*

Внутри блока процессы могут связываться друг с другом с помощью либо сигналов, либо совместно используемых значений. Таким образом, блок обеспечивает не только удобный механизм группировки процессов, но и границу доступности данных. В силу этого при определении блоков следует проявлять осторожность, чтобы процессы, вошедшие в блок, образовывали функционально содержательную группу. В большинстве случаев целесообразно сначала разбить систему (или блок) на функциональные элементы, а затем определить процессы, попадающие в блок.

В блоке при необходимости можно (но не обязательно) определить пути связи между процессами или между процессами и окружающей средой блока (то есть границей блока). Такие пути связи называют маршрутами сигналов.

Для больших SDL-систем существует возможность описать подструктуру блока в терминах других блоков и каналов так, как будто блок является самой системой. Такой механизм описан в § D.4.

Определение структуры блока может включать следующие элементы:

– Имя блока.

– Определения сигналов: спецификация типов сигналов, обмениваемых внутри блока. Включает спецификацию типов значений, передаваемых сигналами (список сортов).

– Определения списка сигналов: спецификация идентификаторов, соответствующих спискам сигналов и/или другим идентификаторам списков сигналов. Такие идентификаторы, объединяющие несколько сигналов в одну группу, могут использоваться для экономии места и обеспечения более ясной спецификации.

– Определения маршрутов сигналов: спецификация путей связи, соединяющих процессы блока друг с другом и с окружающей средой блока. Определение маршрута сигнала содержит спецификацию идентификаторов сигналов, транспортируемых по этому маршруту.

– Соединения каналов с маршрутами: спецификация соединений каналов, внешних по отношению к блоку, с маршрутами сигналов, расположенных внутри блока.

– Определения процессов: спецификация типов процессов, описывающих поведение блока. Если блок не описан в терминах своей подструктуры, то в блоке должно находиться хотя бы одно определение типа процесса. Определение процессов обеспечено механизмом ссылок, подобным таковому для блоков и отмеченным в § D.3.1.

– Определения данных: спецификация определенных пользователем новых типов, синонимичных типов и генераторов, доступных во всех описанных процессах блока и/или подструктуре блока.

– Определения макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

Если для блока определена подструктура, то некоторые из вышеперечисленных элементов не являются обязательными. (Объяснения структурирования см. в § D.4.)

В блоке доступны следующие типы:

– предварительно определенные типы данных;

– типы данных, определенные пользователем в самом блоке;

– типы данных, определенные пользователем и доступные в родительском блоке (в случае разбиения блока).

В SDL/PR для отграничения определения блока используются ключевые слова BLOCK и ENDBLOCK. Примеры определения блока на SDL/GR приведены в § D.3.6.

D.3.3 *Каналы*

Каналы являются средством связи между различными блоками системы или между блоками и окружающей средой. Канал может соединить один блок с другим или один блок с окружающей средой либо только в одном направлении (односторонний канал), либо в обоих направлениях (двусторонний канал). Обычно канал является функциональным элементом, который может быть использован для обозначения конкретных путей связи. Фактически посредством разбиения каналов (описываемого в § D.4.5) возникает возможность дать формульную спецификацию поведения каждого канала.

Для каждого указанного направления или пути связи спецификация канала содержит список всех идентификаторов сигналов, которые могут быть переданы каналам в этом направлении. Этот список играет роль гаранта того, что каждый сигнал, посланный процессом на одном конце канала, может быть получен процессом в блоке на другом конце канала. Таким образом, спецификация канала становится частью спецификации интерфейса каждого блока. В больших проектах, в которые вовлечены многочисленные разработчики, ранние соглашения о сигналах в каналах и о спецификации этих сигналов снижают вероятность того, что два процесса не смогут связываться друг с другом, как это было задумано.

Определение канала включает следующие элементы:

– Имя канала.

– Один или два пути связи: путь связи специфицирует источник и приемник списка сигналов. В этом контексте могут использоваться идентификаторы блоков и ключевое слово "ENV" (означающее окружающую среду).

– Один или два списка сигналов: для каждого направления связи должен быть специфицирован список сигналов, транспортируемых в данном направлении. Список может включать идентификаторы сигналов, а также идентификаторы других списков.

– Необязательное определение (или ссылка на него) подструктуры канала; см. § D.4.5.

На SDL/PR определение канала заключено между ключевыми словами CHANNEL и ENDCHANNEL. Для выражения направления связи используются ключевые слова FROM и TO, а для выражения списка сигналов используется ключевое слово WITH. На рис. D-3.3.1 показан пример определения канала на SDL/PR.

На SDL/GR определение канала изображается линией, соединяющей две стороны, участвующие в связи. Имя канала должно быть ближе к линии, чем любой другой символ. Направления изображаются с помощью стрелок, а списки сигналов должны быть заключены в квадратные скобки, как это иллюстрируется примерами на рис. D-3.3.2. Стрелки нельзя помещать ни у одного из концов линии, чтобы не путать каналы с маршрутами сигналов (см. § D.3.5).

В двусторонних каналах каждый из двух списков сигналов должен быть ближайшим к соответствующей стрелке.

```
SYSTEM so_and_so;
    . . .
    SIGNALLIST s_list_id_1 = sig_b, sig_c, sig_d;
    . . .
    CHANNEL c1
        FROM block_a TO block_b WITH signal_1,signal_2,signal_3;
    ENDCHANNEL c1;

    CHANNEL chan_2
        FROM ENV TO block_c WITH ext_sig_1,ext_sig_2;
        EROM block_c TO ENV WITH int_sig_1;
    ENDCHANNEL chan_2;

    CHANNEL chan.name.3
        FROM bx TO by WITH sig_a,(s_list_id_1),sig_e;
    ENDCHANNEL chan.name.3;
    . . .
ENDSYSTEM so_and_so;
```

РИСУНОК D-3.3.1

Пример определения канала на SDL/PR

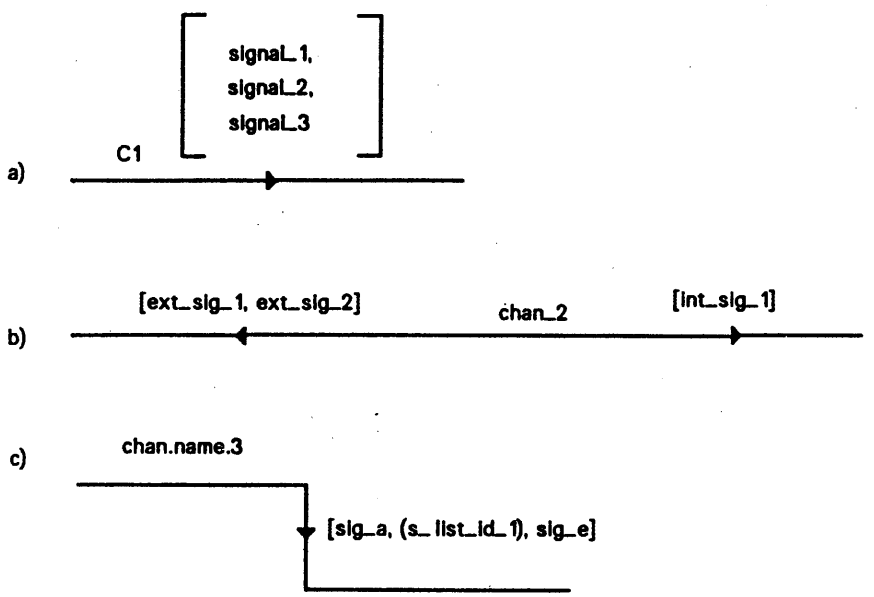


РИСУНОК D-3.3.2

Пример определения канала на SDL/GR

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.3

D.3.4 *Сигналы*

Сигналы могут быть определены на уровне системы, уровне блока или во внутренней части определения процесса. Сигналы, определенные на некотором уровне, могут использоваться на этом уровне, а также на более низких уровнях, однако, для упрощения каждого уровня рекомендуется определять сигналы как можно более локализованно. Сигналы, определенные внутри определения процесса, могут пересылаться либо между отдельными экземплярами процесса одного и того же типа (см. § D.3.8), либо между сервисами, входящими в данный процесс (§ D.5.3).

Определение сигнала включает следующие элементы:

– Имя сигнала.

– Список сортов (не обязательно): изображает список типов значений, передаваемых этим сигналом.

– Уточнение сигнала (не обязательно): поясняется в § D.4.7.

На SDL/PR определение сигнала специфицируется ключевым словом SIGNAL. Несколько сигналов (не подвергшихся уточнению) могут быть определены внутри конструкции, поставляющей имя сигнала и список сортов. Примеры определения сигналов на SDL/PR приведены на рис. D-3.4.1.

```
SIGNAL s1,s2,s3;
SIGNAL sig_a (sort1,sort2),
       sig_b (sort3,sort4);
```

РИСУНОК D-3.4.1

Примеры определения сигнала на SDL/PR

На SDL/GR определение сигнала специфицируется заключением линейных предложений в символ текста, как это изображено на рис. D-3.4.2.

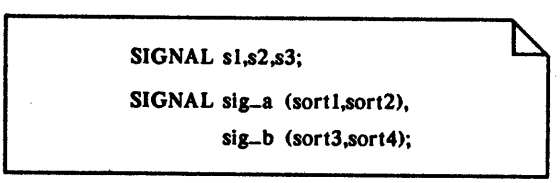


РИСУНОК D-3.4.2

Примеры определения сигнала на SDL/GR

D.3.5 *Маршруты сигналов*

Маршруты сигналов используются для определения путей связи подобно тому, как это делают каналы. Они могут использоваться на уровне блоков, а также на уровне процессов. Подобно каналам они могут быть односторонними или двусторонними, но в отличие от каналов их нельзя разбивать.

На уровне блока они представляют собой средства связи между процессами блока или между процессами и окружающей средой блока, то есть с каналами, ведущими к блоку или из него.

На уровне процесса маршруты сигналов могут использоваться при разложении процесса на сервисы (см. § D.5.3). В этом случае они соединяют сервисы друг с другом или с маршрутами сигнала данного процесса.

Если сигнал подается на маршрут сигнала, ведущий к границе блока, то сигнал поступает в канал, присоединенный к маршруту сигнала. Когда сигнал поступает к блоку по каналу, который присоединен к одному или нескольким маршрутам сигнала, то сигнал подается на маршрут сигнала, способный передать его.

На SDL/PR определение маршрута сигнала начинается ключевым словом SIGNALROUTE. Синтаксис направлений связи и списков сигналов совпадает с таковыми для каналов.

На SDL/GR единственная разница между маршрутом сигнала и каналом заключается в том, что для маршрутов сигналов стрелки должны помещаться в конце линий; вблизи от каждой стрелки должен находиться надлежащий список сигналов. В нижеследующих параграфах на рис.D-3.6.3 и рис.D-3.6.5 приведены примеры маршрутов сигналов.

D.3.6 *Диаграммы системы и блока*

На SDL/GR определение системы изображается с помощью совокупности диаграмм. Структура системы в терминах каналов и блоков изображается с помощью диаграммы системы.

Диаграмма системы состоит из:

– Символа кадра: символ, имеющий форму прямоугольника, охватывающего все прочие символы. Этот символ изображает границу системы: вне этого кадра находится окружающая среда системы.

– Заголовка системы: ключевое слово SYSTEM, за которым следует имя системы (заголовок помещается в левый верхний угол кадра).

– Необязательной нумерации страниц (помещаемой в правый верхний угол кадра).

– Символов текста: такой символ может охватывать линейные предложения. Обычно он используется для представления в диаграммах определений сигналов, списков сигналов и данных.

– Области взаимодействия блоков: содержит спецификацию блоков системы, каналов и списков сигналов, транспортируемых по каналам.

– Диаграмм макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

В диаграмме системы определением блока может быть один из двух нижеследующих элементов:

– Ссылка на блок: символ блока, содержащий только имя блока.

– Диаграмма блока: кадр, содержащий спецификацию структуры блока в терминах его процессов и их взаимодействий. Если блок разбит на подблоки, то внутри кадра блока должна быть помещена либо спецификация подструктуры, либо ссылка на нее (см. § D.4.3).

Форму символов, используемых в диаграммах систем и блока, можно найти в сводном описании SDL/GR. Размеры символов см. § D.7.1.4.

На рис. D-3.6.1 приводится пример диаграммы системы для системы "s". В этом примере система "s" разложена на два блока, B1 и B2, соединенных друг с другом и с окружающей средой каналами C1, C2, C3 и C4. В этом примере для блоков B1 и B2 приведены только ссылки. Дальнейшие разъяснения о каналах и символах списков сигналов приводятся в нижеследующих параграфах.

Тот же пример на SDL/PR приведен на рис. D-3.6.2.

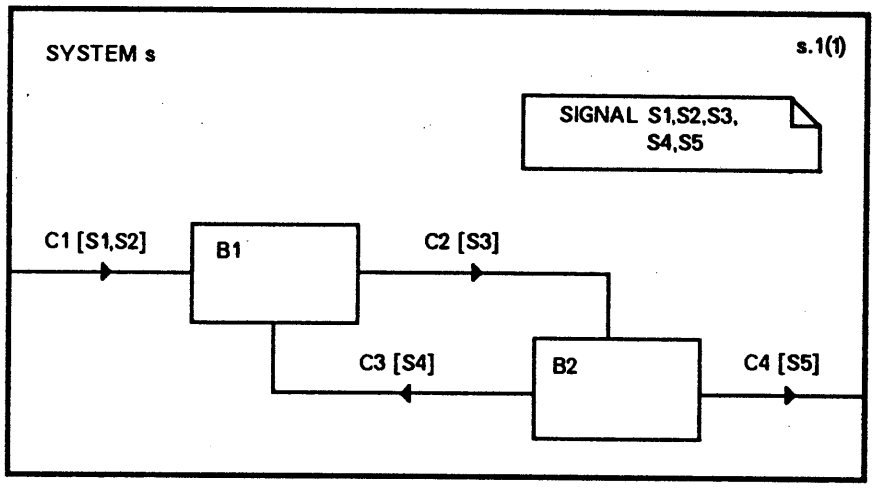


РИСУНОК D-3.6.1

Пример диаграммы системы

```
SYSTEM s;
    SIGNAL S1,S2,S3,S4,S5;
    CHANNEL C1 FROM ENV TO B1 WITH S1,S2;
    ENDCHANNEL C1;
    CHANNEL C2 FROM B1 TO B2 WITH S3;
    ENDCHANNEL C2;
    CHANNEL C3 FROM B2 TO B1 WITH S4;
    ENDCHANNEL C3;
    CHANNEL C4 FROM B2 TO ENV WITH S5;
    ENDCHANNEL C4;
    BLOCK B1 REFERENCED;
    BLOCK B2 REFERENCED;
ENDSYSTEM s;
```

РИСУНОК D-3.6.2

Пример определения структуры системы на SDL/PR

Структура блока в терминах процессов и маршрутов сигналов изображается на SDL/GR с помощью диаграммы блока.

Диаграмма блока состоит из:

– Символа кадра: символ, имеющий форму прямоугольника, охватывающего все прочие символы. Этот символ изображает границу блока: вне этого кадра находится окружающая среда блока.

– Заголовка блока: ключевое слово BLOCK, за которым следует имя блока (заголовок помещается в левый верхний угол кадра).

– Необязательной нумерации страниц (помещаемой в правый верхний угол кадра).

– Символов текста: такой символ может охватывать линейные предложения. Обычно он используется для представления определений сигналов, списков сигналов и данных.

– Области взаимодействия процессов: она содержит спецификацию процессов блока, а также, возможно, маршруты сигналов и списки сигналов, транспортируемых по этим маршрутам. В этой области возможно также изображать процессы, порождающие другие процессы; это свойство описывается в § D.3.8.1.

– Идентификаторов каналов: если в диаграмме изображаются маршруты сигналов, ведущие от окружающей среды блока или к ней, то вне кадра, но вблизи от соответствующих линий маршрутов сигналов должны быть приведены идентификаторы каналов, присоединенных к этим маршрутам сигналов.

– Диаграмм макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

В диаграмме блока спецификация процесса может быть представлена альтернативно:

– Ссылкой на процесс: символ процесса, содержащий имя процесса и, не обязательно, спецификацию числа экземпляров процессов. Эта спецификация числа экземпляров процессов состоит из пары целых чисел, разделенных запятой и заключенных в круглые скобки (см. § D.3.8).

– Диаграммой процесса: кадр, содержащий граф из соединенных символов; граф описывает поведение процесса в терминах состояний, вводов, выводов, действий и т.д. (см. § D.3.8). Если процесс разбит на сервисы, то диаграмма процесса содержит область взаимодействия сервисов (см. § D.5.3).

Если для данного блока указана его подструктура, то некоторые из вышеназванных элементов являются необязательными. (См. в § D.4 пояснения по структурированию.)

На рис. D-3.6.3 приведен пример диаграммы блока для блока "B1", указанного в примере на рис. D-3.6.1. Блок B1 описан в терминах двух процессов, P1 и P2, соединенных маршрутами сигналов R1, R2, R3, R4 и R5. В примере для процессов P1 и P2 указаны только ссылки. Вне кадра специфицированы также идентификаторы каналов C1, C2 и C3.

Тот же пример, но на SDL/PR, приведен на рис. D-3.6.4.

Как указывалось выше, вместо ссылок на определение блока сама диаграмма блока может быть включена в диаграмму системы. Например, на рис. D-3.6.5 диаграммы рисунков D-3.6.1 и D-3.6.3 влиты в единственную диаграмму системы.

Общим критерием разработки таких диаграмм должно быть стремление не создавать слишком сложных диаграмм, чтобы их было легко читать и можно было поместить на одну страницу.

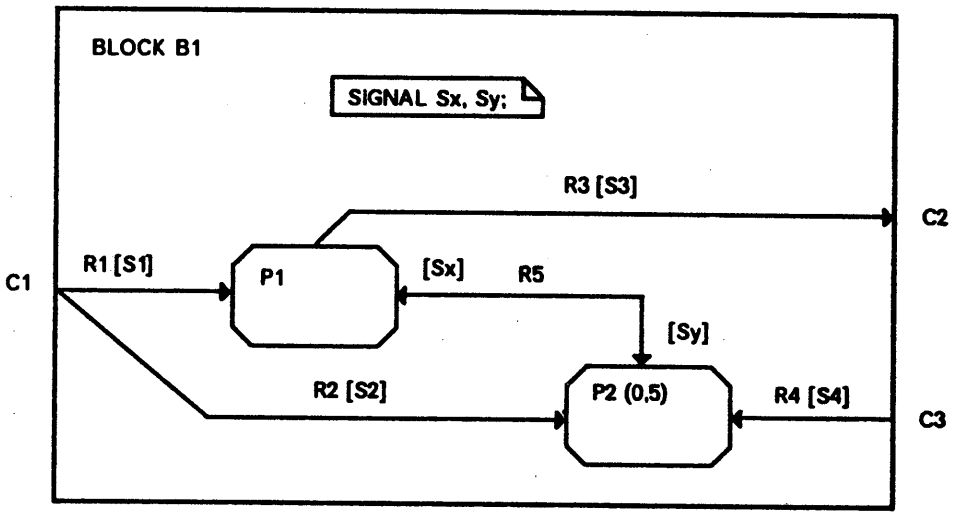


РИСУНОК D-3.6.3

Пример диаграммы блока

```
BLOCK B1;

    SIGNAL Sx,Sy;

    SIGNALROUTE R1 FROM ENV TO P1 WITH S1;
    SIGNALROUTE R2 FROM ENV TO P2 WITH S2;
    SIGNALROUTE R3 FROM P1 TO ENV WITH S3;
    SIGNALROUTE R4 FROM ENV TO P2 WITH S4;
    SIGNALROUTE R5 FROM P1 TO P2 WITH Sy;
                   FROM P2 TO P1 WITH Sx;

    CONNECT C1 AND R1,R2;
    CONNECT C2 AND R3;
    CONNECT C3 AND R4;

    PROCESS P1 REFERENCED;
    PROCESS P2 REFERENCED;

ENDBLOCK B1;
```

РИСУНОК D-3.6.4

Пример определения структуры блока на SDL/PR

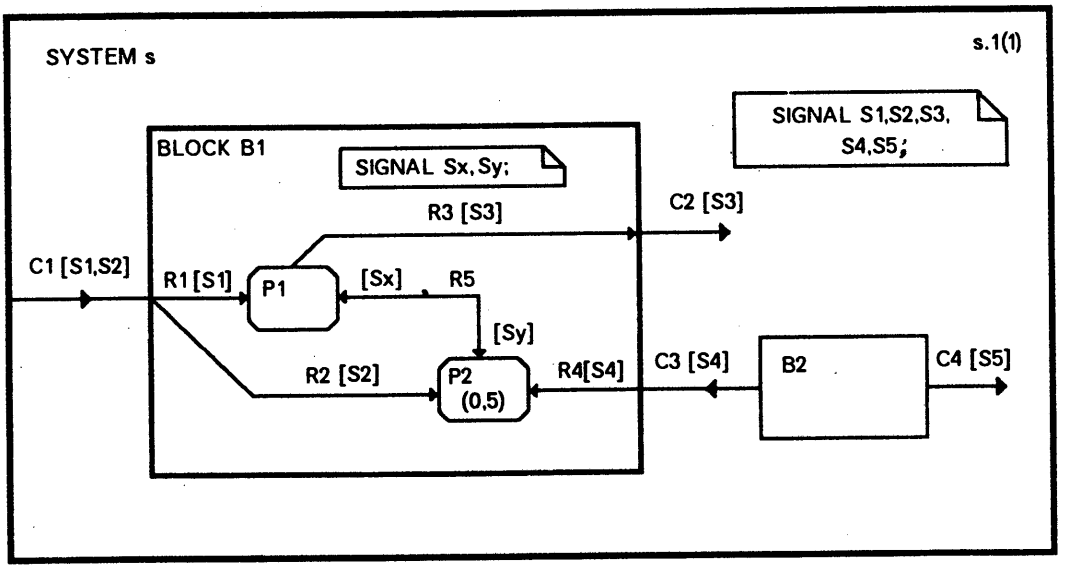


РИСУНОК D-3.6.5

Пример диаграммы системы, содержащей диаграмму блока

D.3.7 *Комментарии и расширение текста*

D.3.7.1 *Комментарии*

Для облегчения понимания читателем и разъяснения некоторых частей SDL-спецификации к ним могут быть добавлены комментарии. В SDL введены два типа комментариев, чтобы быть удобными как в SDL/PR, так и в SDL/GR.

Первый тип называется "замечание" и особенно используется в SDL/PR. Он отграничен знаками"/*" в начале и знаками "*/" в конце комментария.

В SDL/PR такой комментарий может быть помещен везде, где найдется свободное место. Комментарий не должен содержать специальной цепочки "*/".

В SDL/GR такой комментарий может быть помещен везде, где в линейных предложениях найдется свободное место.

На рис. D-3.7.1 и рис. D-3.7.2 приведены некоторые примеры этой формы комментария как для SDL/PR, так и для SDL/GR.

```
SYSTEM NSS;

/* Определение национальной
     коммутируемой системы.

     Имя системы: NSS */

/*****************************/

     . . .
     SIGNAL s1; /*Пояснения по использованию сигнала s1*/

     . . .
     BLOCK MAI REFERENCED; /*Сопровождение*/

     . . .
ENDSYSTEM NSS;
```

РИСУНОК D-3.7.1

Примеры первой формы комментария на SDL/PR

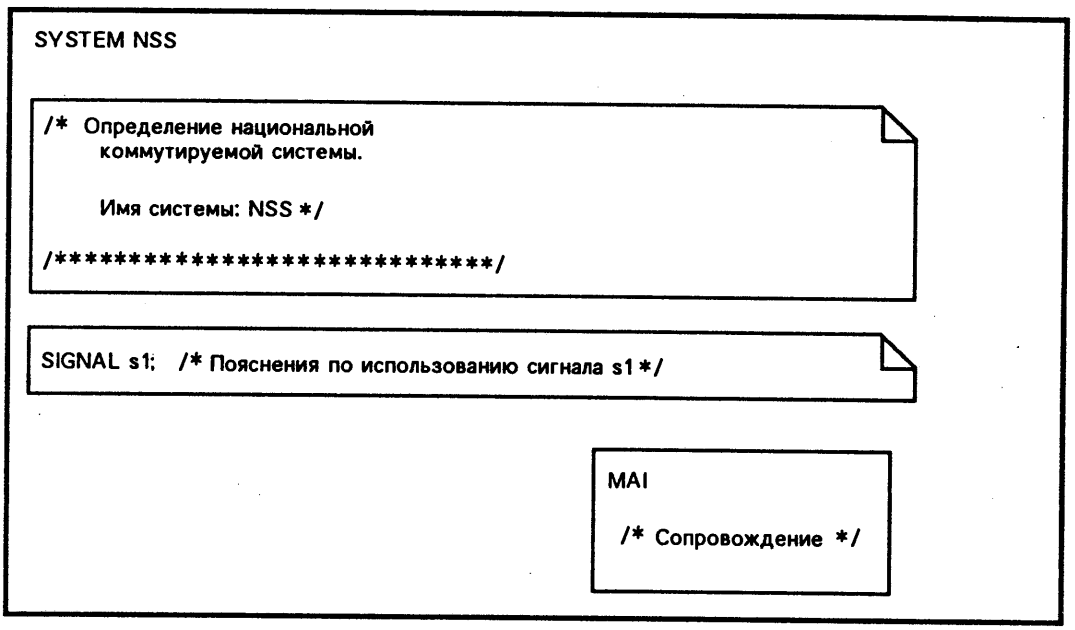


РИСУНОК D-3.7.2

Примеры первой формы комментария на SDL/GR

Вторая форма комментария допускает взаимно-однозначное отображение между SDL/PR и SDL/GR и более удобна в приложениях, в которых выполняется автоматическая трансляция.
В SDL/PR эта форма комментария состоит из ключевого слова COMMENT, вслед за которым идет цепочка знаков; комментарий может быть вставлен в качестве предложения (то есть после него должен стоять знак

";") всюду, где может стоять предложение "работа". Кроме того, он может быть вставлен перед знаком ";" (точка с запятой) в конце любого предложения.

В SDL/GR эта форма комментария изображается с помощью символа комментария, содержащего текст комментария. Символ комментария имеет форму прямоугольника, в котором отсутствует левая или правая сторона. Он должен быть достаточно вытянутым как по вертикали, так и по горизонтали, чтобы содержать весь текст. Он может быть присоединен к любому символу SDL/GR или линии потока. Для присоединения должна использоваться пунктирная линия. Если связь между текстом комментария и символом комментария не является двусмысленной, то символ комментария может быть изображен просто в виде квадратной скобки.

На рис. D-3.7.3 и рис. D-3.7.4 приведены некоторые примеры этой формы комментария как для SDL/PR, так и для SDL/GR.

```
SYSTEM s COMMENT 'комментарий, касающийся системы';
    CHANNEL C2
        FROM B1 TO B2
        WITH s3          COMMENT 'комментарий, касающийся канала';
    ENDCHANNEL C2;
    BLOCK B1
    COMMENT 'комментарий, касающийся блока';
        . . .
        PROCESS p1;
            . . .
            TASK 't1';
            TASK 't2';
            . . .
        ENDPROCESS p1;
    ENDBLOCK B1;
ENDSYSTEM s;
```

РИСУНОК D-3.7.3

Примеры второй формы комментария на SDL/PR

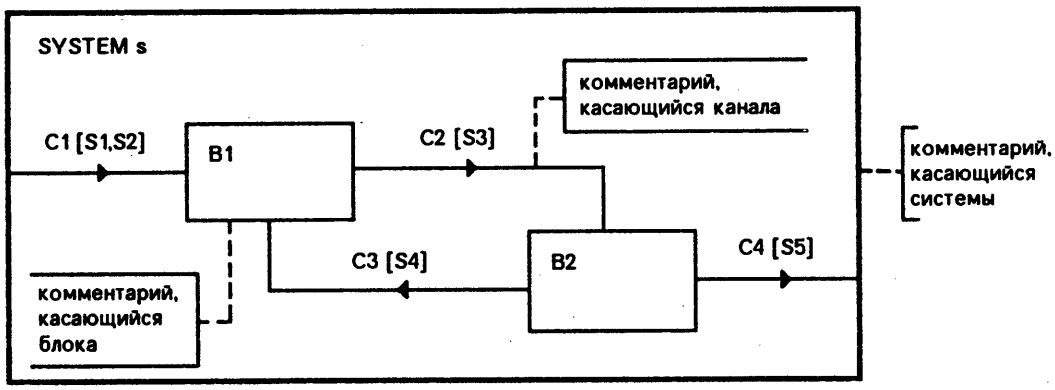


РИСУНОК D-3.7.4

Примеры второй формы комментария на SDL/GR

D.3.7.2 *Расширение текста*

Как правило, текст, связанный с символом, должен быть помещен внутри символа. Однако часто это или невыполнимо, или непрактично. В качестве альтернативы текст можно помещать внутрь символа расширения текста, присоединенного к соответствующему символу. Символ расширения текста подобен символу комментария; единственная разница заключается в том, что соединяющая линия является сплошной, а не пунктирной (см. рис. D-3.7.5).

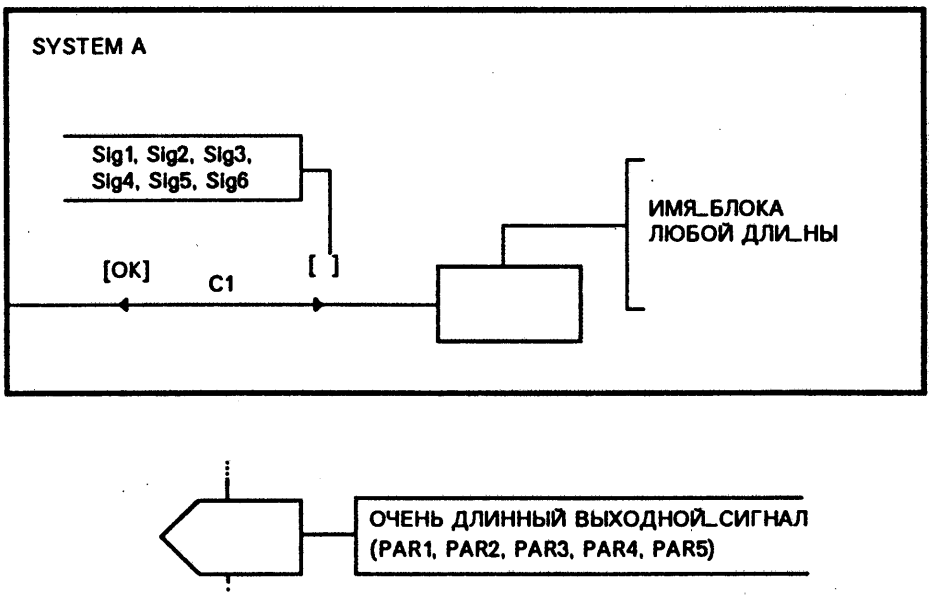


РИСУНОК D-3.7.5

Примеры использования символа расширения текста

Знак подчеркивания ("_") можно использовать в конце строки текста в качестве знака продолжения. Тогда остающиеся в строке знаки пробела не считаются входящими в текст. Более детальные пояснения по синтаксису имен можно найти в § D.3.13.

D.3.8 *Процессы*

Процесс является Расширенной Машиной с Конечным Числом Состояний, определяющей динамическое поведение системы. В основном процессы находятся в состояниях ожидания сигналов. После получения сигнала процесс отвечает выполнением конкретных действий, специфицированных для каждого типа сигнала, который процесс может получить. Процесс содержит много различных состояний, позволяющих процессу выполнять после получения сигнала различные действия. Эти состояния являются памятью о тех действиях, которые были выполнены ранее. После выполнения всех действий, связанных с получением конкретного сигнала, процесс входит в следующее состояние и начинает ждать поступления другого сигнала.

Процесс может либо существовать в момент создания системы, либо он может быть создан в результате запроса на создание со стороны другого процесса. Кроме того, процессы могут либо существовать вечно, либо они могут останавливаться, выполнив действие "стоп".

Определение процесса представляет собой спецификацию типа процесса; в одно и то же время могут быть созданы и существовать несколько процессов одного и того же типа; они могут действовать параллельно и независимо. Определение процесса состоит из следующих элементов (некоторые из них являются необязательными):

– Имя процесса.

– Пара целых чисел: первое число специфицирует число экземпляров процессов, создаваемых при создании системы; если оно отсутствует, то по умолчанию его значение равно 1. Второе целое специфицирует максимальное число экземпляров, которые могут одновременно существовать; если оно отсутствует, то по умолчанию максимальное число не ограничено.

– Формальные параметры: список идентификаторов переменных, связанных со своими сортами и используемых для передачи процессу информации в момент его создания. Для этих целей в запросе на создание процесса может быть указан список фактических параметров. Значения формальных параметров тех процессов, которые создаются в момент создания системы, не определены.

– Набор действительных входных сигналов: список идентификаторов сигналов, определяющий те сигналы, которые могут быть получены процессом.

– Определения сигналов: спецификация сигналов, которыми могут обмениваться экземпляры одного и того же процесса или сервисы в процессе (см. § D.5.3).

– Определения процедур: спецификации процедур, которые могут быть вызваны процессом. В этом контексте может использоваться ссылка на процедуру. (Процедуры разъясняются в § D.3.9.)

– Определение данных: спецификации определенных пользователем новых типов, синонимичных типов и генераторов, локальных для данного процесса.

– Определение переменных: объявление переменных процесса. Возможно объявление переменной как совместно используемой несколькими процессами одного и того же блока (переменная REVEALED) или как экспортируемой другим процессам, в том числе в других блоках (переменная EXPORTED). Для каждой объявляемой переменной должен быть специфицирован идентификатор ее сорта. Возможна (но не обязательно) спецификация начальных значений.

– Определения обозревания: объявление идентификаторов переменных, которые могут быть использованы для получения значений переменных, которыми владеют другие экземпляры процесса. Для каждого идентификатора переменной должен быть специфицирован сорт переменной.

– Определения импорта: спецификация идентификаторов тех переменных, которыми владеют другие процессы и которые хочет импортировать данный процесс. Для каждого идентификатора должен быть специфицирован сорт переменной.

– Определение таймера: разъясняется в § D.3.11.

– Определения макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

– Тело процесса: спецификация фактического поведения процесса в терминах состояний, вводов, выводов, работ и т.д. Если процесс разбит на подкомпоненты (сервисы), то вместо тела процесса определение процесса должно содержать раздел о разбиении на сервисы. Описание этого свойства приводится в § D.5.3.

Примеры и объяснения относительно данных, переменных, определений обозревания и определений импорта приводятся в § D.3.10.

Частичный пример определения процесса на SDL/PR приведен в рис. D-3.8.1 (ключевые слова языка написаны прописными буквами).

```
PROCESS p1 (2,100);
   FPAR var1,var2 sort1,
        var3 sort2, var4 sort3;      } Формальные параметры
   SIGNALSET s1,s2,s3;               } Набор действительных входных сигналов
   SIGNAL s4,s5;
   SIGNAL s6 (sort6,sort7);          } Определения сигналов
   PROCEDURE . . .                   } Определения процедур
   . . . определения типа данных . . . } Определения типов данных
   DCL . . .                         } Определения переменных
   . . . тело процесса . . .         } Тело процесса
ENDPROCESS p1;
```

РИСУНОК D-3.8.1

Частичный пример определения процесса на SDL/PR

Тело процесса представляет собой фактический граф машины с конечным числом состояний. На SDL/PR оно состоит из последовательности упорядоченных предложений, а на SDL/GR – это последовательность символов, соединенных ориентированными дугами (подобно блок-схемам). Спецификация тела процесса должна всегда начинаться с ключевого слова START, за которым следует набор действий (переход). Интерпретация экземпляра процесса начинается с момента создания процесса.

Возможными действиями, выполняемыми при переходе, являются:

- Работа: присвоение переменным (или неформальный текст).
- Экспорт: экспорт переменной.
- Установка: запрос на активизацию таймера.
- Сброс: сброс таймера.
- Вывод: посылка сигнала другому процессу.
- Запрос на создание: создание экземпляра процесса специфицированного типа.
- Принятие решения: выбор набора действий в зависимости от вопроса.
- Вызов процедуры: запрос на интерпретацию отдельного, замкнутого набора действий (аналогичного языкам программирования).
- Присоединение: спецификация "перескока" к другому набору действий.

Переход может закончиться одним из следующих действий:

- Следующее состояние: спецификация того состояния, которое примет экземпляр процесса.

- Стоп: немедленное прекращение экземпляра процесса.

После спецификации стартового действия и необязательного стартового перехода тело процесса включает определения всех возможных состояний данного процесса. Каждое определение состояния начинается спецификацией возможных стимулов, ожидаемых процессом в этом состоянии. Возможными стимулами являются:

- Вводы: сигналы, которые могут быть получены.

- Сохранения: сигналы, которые должны быть сохранены для будущей обработки.

- Разрешающие условия: объясняются в § D.3.8.5.

- Непрерывные сигналы: объясняются в § D.3.8.5.

Каждому стимулу, кроме сохранения, должен соответствовать специфицируемый переход. Такой переход представляет собой последовательность действий, которые выполнит процесс при наступлении этого стимула.

Если процесс выполняет действие "стоп" и если имеются ожидающие сигналы, которые были посланы процессу, но еще не получены им, то такие сигналы аннулируются.

На SDL/GR определение процесса изображается с помощью диаграммы процесса. Диаграмма процесса состоит из следующих элементов:

- Символ кадра: символ, имеющий форму прямоугольника, охватывающего все прочие символы. Если не существует маршрутов сигналов, присоединенных к символу кадра, то он может быть опущен.

- Заголовок процесса: ключевое слово PROCESS, за которым следует идентификатор процесса; далее могут следовать спецификация числа экземпляров процессов и за ней спецификация формальных параметров. Заголовок процесса помещается в верхний левый угол кадра.

- Необязательная нумерация страниц (помещаемая в правый верхний угол кадра).

- Символы текста: в диаграмме процесса символ текста может содержать определения сигналов, переменных, обозреваний, импортов, данных и таймера.

- Ссылки на процедуры: символ процедуры, содержащий имя процедуры, представляющее отдельно определенную процедуру данного процесса.

- Диаграммы процедур: по одной на каждую локальную процедуру, которая не задана ссылкой на нее.

- Область графа процесса: спецификация поведения процесса в терминах старта, состояний, вводов, выводов, работ, ... и ориентированных дуг. Если процесс разбит на сервисы, то область графа процес-

са содержит спецификацию сервисов или ссылки на них (см. § D.5.3). Символы GR, используемые в теле процесса, можно найти в свободном описании SDL/GR.

– Диаграммы макрокоманд: указания по использованию макрокоманд приведены в § D.5.1.

На рис. D-3.8.2 приведен пример определения процесса на SDL/GR; дальнейшие примеры и пояснения, касающиеся графов процессов, можно найти в последующих параграфах.

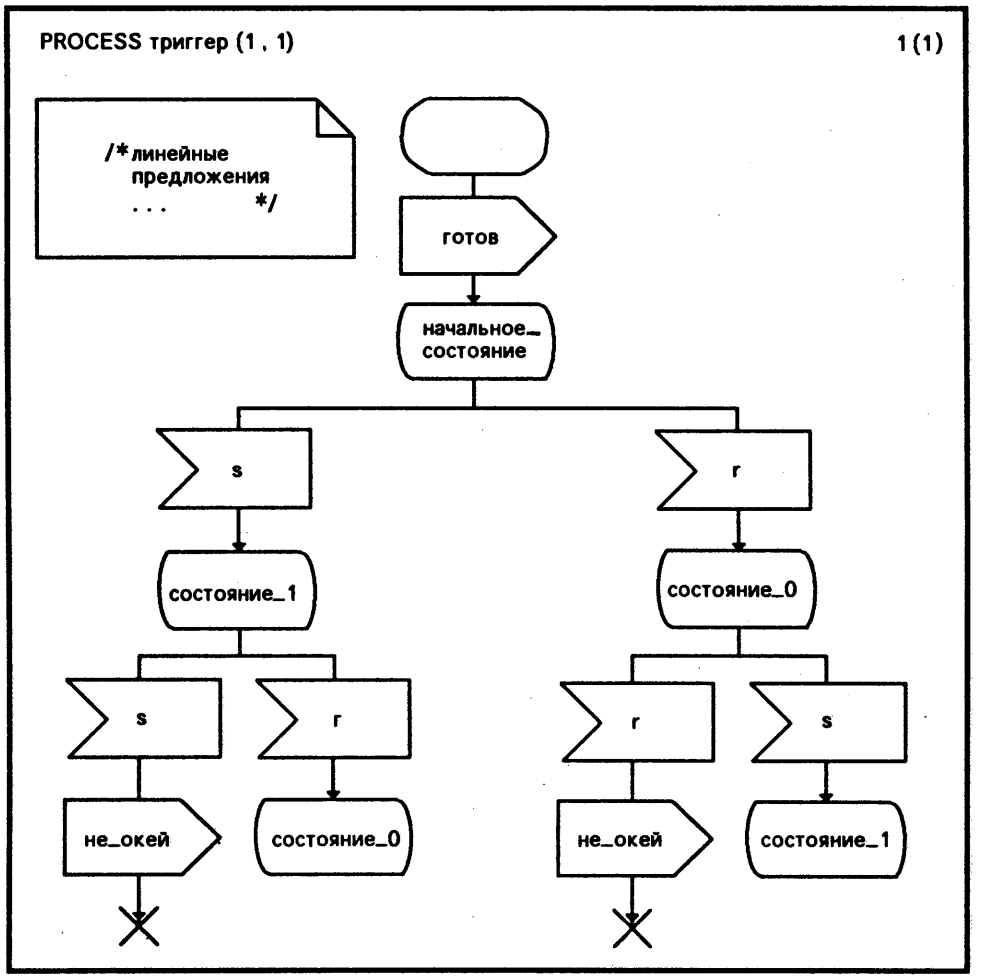


РИСУНОК D-3.8.2

Пример диаграммы процесса

Если граф процесса не помещается на одном листе, то диаграмма может размещаться на нескольких листах при условии, что:

– Должна быть обеспечена нумерация страниц, состоящая из номера страницы и общего числа страниц диаграммы, например 1 (9).

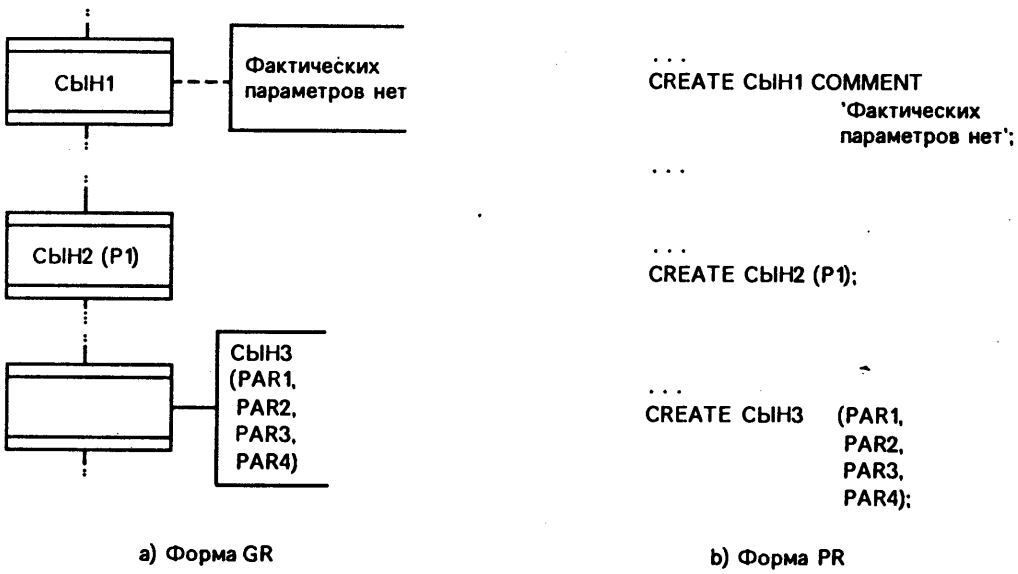
– Символ присоединения или символ следующего состояния может быть использован для соединения отдельных частей графа процесса.

Хорошим критерием деления графа процесса на части является возможность представления каждого определения состояния на отдельной странице. Если определение одного состояния не помещается на одной странице, то можно использовать символ присоединения, задающего связь с частью графа на другой странице.

D.3.8.1 *Создание процесса*

Как указывалось в предыдущем параграфе, процессы (то есть экземпляры процессов) могут быть созданы либо в результате явного запроса на создание, либо в момент создания системы.

Явный запрос на создание может выдать только другой процесс в том же блоке, в котором создается процесс; в запросе могут быть специфицированы фактические параметры, передающие информацию новому создаваемому процессу. Примеры создания процесса в формах PR и GR приведены на рис. D-3.8.3.



a) Форма GR

b) Форма PR

РИСУНОК D-3.8.3

Примеры запросов на создание на SDL/GR и SDL/PR

Если в момент создания системы создаются один или более процессов, то определения этих процессов не должны содержать обращения к формальным параметрам, так как они остаются неопределенными. Следовательно, любое определение процесса, содержащее формальные параметры, должно либо обеспечить, что к его формальным параметрам не будет обращения, прежде чем они получат значения, либо явно включить в спецификацию нулевое число экземпляров этого процесса в момент создания системы (см. рис. D-3.8.4).

```
PROCESS явно_созданный_процесс_1 (0,...);
  FPAR ...
  ...
ENDPROCESS явно_созданный_процесс_1;
```

РИСУНОК D-3.8.4

Определение процесса, содержащее спецификацию формальных параметров

При создании процесса значения экземпляров процесса могут быть определены с помощью следующих предварительно определенных выражений:

OFFSPRING: возвращает значение PId последнего созданного экземпляра.

SELF: возвращает значение PId самого́ экземпляра процесса.

PARENT: возвращает значение PId создающего экземпляра.

SENDER: возвращает значение PId процесса, пославшего последний воспринятый сигнал.

Если процесс создается в результате создания системы, то только выражение SELF возвращает значение PId, а значения выражений OFFSPRING и PARENT равны NULL.

Эти значения чрезвычайно важны, когда существует несколько экземпляров процессов одного и того же типа, так как они обеспечивают единственную возможность однозначно адресовать сигналы отдельным экземплярам. Действительно, как это подробнее объяснено в § D.3.8.6, если процесс посылает сигнал, то он должен специфицировать приемный экземпляр, пока приемный экземпляр однозначно не установлен.

Пользователь должен быть очень осторожен и обеспечить создаваемым экземплярам процесса возможность поддерживать друг с другом связь, если в этом возникает необходимость. Для обеспечения этого часто требуется наличие некоторого инициализирующего процесса. Такой процесс, создаваемый в момент создания системы, будет создавать другие процессы и, возможно, сообщать всем процессам те значения PId, в которых они будут нуждаться.

Другими важными замечаниями по поводу создания процессов являются следующие:

1) После того, как система создана, процесс может быть создан только другим процессом в том же блоке. Одним из способов, позволяющим создание процессов в другом блоке, является наличие в каждом блоке специального процесса, создающего процесс при получении сигнала из другого блока.

2) Раз возникнув, процессы начинают свою собственную жизнь, длительность которой ограничена ими самими. Умирают процессы, только выполнив действие "стоп" в течение перехода. Одним из способов моделирования системы, в которой допустима внешняя операция умерщвления, заключается в наличии специального сигнала умерщвления. Получив сигнал умерщвления, процесс выполняет действие "стоп".

Отношения между создающим и созданным процессами могут быть изображены в диаграмме блока с помощью символа линии создания, как это показано на рис. D-3.8.5.

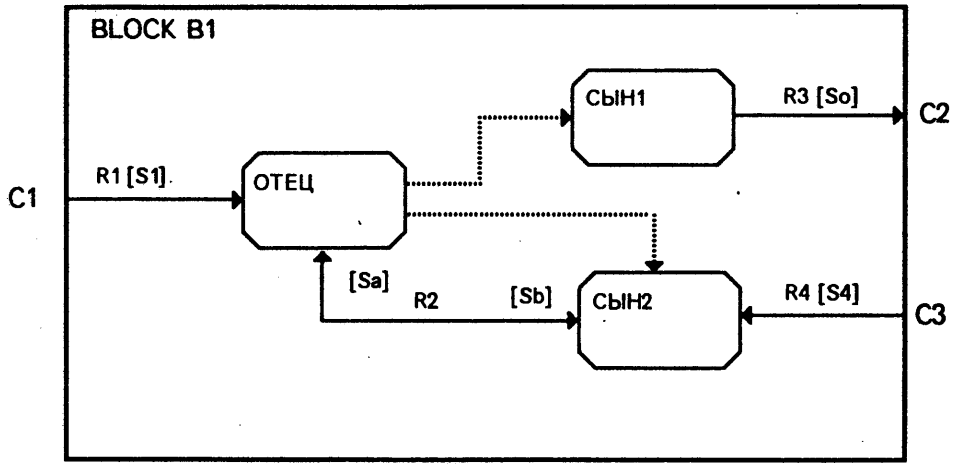


РИСУНОК D-3.8.5

Пример диаграммы блока, содержащей символ линии создания

D.3.8.2 *Состояния*

Состояние является таким моментом процесса, в который никакое действие не выполняется, но просматривается входная очередь: нет ли в ней поступивших сигналов. В зависимости от идентификатора сигнала, имеющегося во входном сигнале, прибытие сигнала побуждает процесс покинуть свое состояние и выполнить некоторую конкретную последовательность действий. Сигнал, который прибыл и вызвал переход, был "воспринят" и перестал существовать.

В SDL/PR состояние изображается ключевым словом STATE, за которым следует имя. Спецификация состояния заканчивается либо перед следующим предложением состояния, либо в конце процесса, либо посредством явно написанного ключевого слова ENDSTATE. В предложении состояния вместо имени состояния можно использовать звездочку; это является сокращенным обозначением того, что следующие за предложением состояния вводы, сохранения и соответствующие переходы должны интерпретироваться в любом состоянии.

Ключевое слово NEXTSTATE, за которым следует имя, используется для спецификации следующего состояния. В предложении, указывающем следующее состояние, вместо имени состояния может стоять черточка; это указывает на то, что следующее состояние совпадает с тем, из которого возник данный переход.

Рис. D-3.8.6 содержит частичный пример на SDL/PR.

```
SYSTEM s;
  . . .
  BLOCK b;
    . . .
    PROCESS p;
      . . .
      START;
        . . .
      STATE st1;
        . . .
      STATE st2;
        . . .
        NEXTSTATE st1;
      . . .
      ENDSTATE st2;
      STATE st3;
        . . .
    ENDPROCESS p;
  ENDBLOCK b;
ENDSYSTEM s;
```

РИСУНОК D-3.8.6

Частичный пример определений состояний на SDL/PR

На SDL/GR состояние изображается посредством символа состояния, содержащего имя состояния; к символу состояния присоединены символы ввода или символы сохранения. Для изображения следующего состояния используется такой же символ состояния с входящей стрелой.

Для удобства, чтобы упростить составление и облегчить понимание диаграмм, в SDL-диаграмме одно и то же состояние разрешается изображать по нескольку раз. Диаграмма, в которой это применимо, считается полностью эквивалентной диаграмме, полученной из данной слиянием всех вхождений одного и того же состояния. Примеры этого приведены на рис. D-3.8.7 и рис. D-3.8.8. На рис. D-3.8.7 b) символ состояния используется в качестве коннектора с основным состоянием, имеющим то же имя, когда это имя стоит в символе следующего состояния. На рис. D-3.8.8 состояние изображено с помощью нескольких символов, при каждом из которых имеется лишь подмножество вводов (или сохранений).

На рис. D-3.8.7 диаграммы a) и b) логически эквивалентны. Диаграмма a) содержит всего по одному вхождению каждого из состояний, в то время как в диаграмме b) используется кратное вхождение состояний. В диаграмме b) каждое из состояний имеет по одному основному вхождению, при котором указаны все относящиеся к этому состоянию символы ввода (или сохранения). Если это состояние может быть достигнуто из других точек диаграммы (в качестве завершающей точки перехода), то оно указывается как состояние, с которым не связаны ни вводы, ни сохранения. Если при символе следующего состояния дать комментарий, отсылающий к символу состояния, то это усилит ясность диаграммы, особенно если вхождения находятся на разных страницах.

В рис. D-3.8.8 используется кратное вхождение состояния для формирования полного комплекта вводов (и сохранений). При каждом из состояний указана лишь часть этого комплекта.

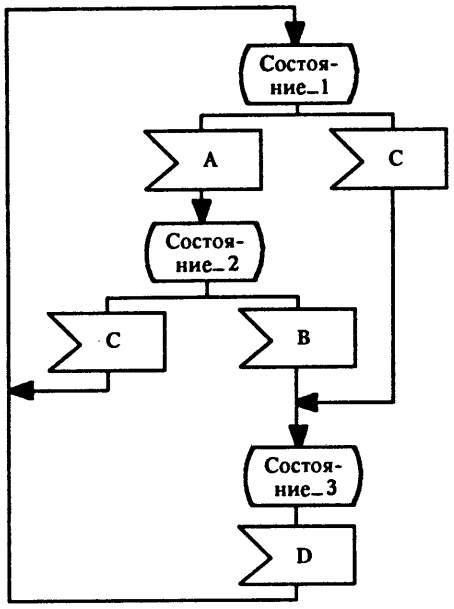


РИСУНОК D-3.8.7a

Однократное вхождение состояния

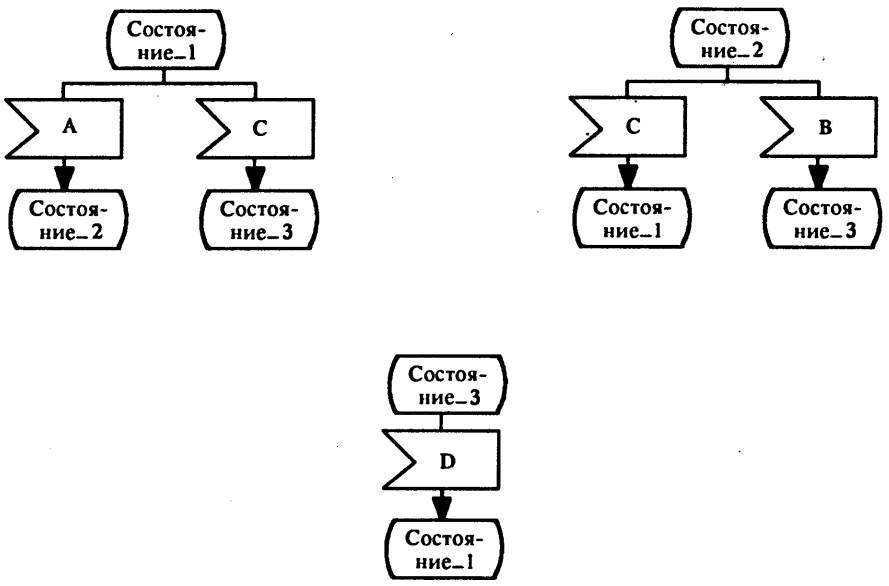


РИСУНОК D-3.8.7b

Диаграмма a) с основными состояниями и последующими состояниями, использованными в качестве коннекторов с состояниями

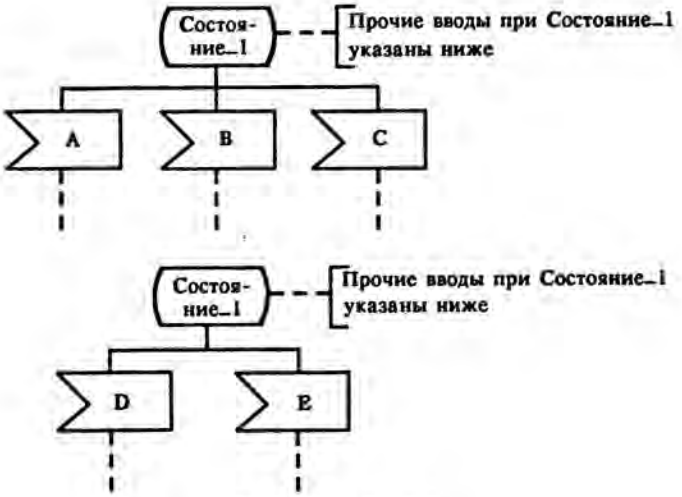


РИСУНОК D-3.8.8

Кратное вхождение состояния для случая, когда нельзя отчетливо указать все вводы при единственном символе состояния

Этот подход был с успехом использован в случаях, при которых с состояниями связано относительно большое число вводов и сохранений; он, однако, связан с опасностью, заключающейся в том, что, если читателям неизвестно о кратности вхождения состояний, они могут ошибочно истолковать диаграмму. Чтобы избежать этого недоразумения, состояния, при которых указана лишь часть вводов/сохранений, должны сопровождаться комментарием, отсылающим к другим состояниям, со связанными с ними вводами и сохранениями, как это изображено на рис. D-3.8.8.

Кратное вхождение может быть с успехом применено при желании сконцентрировать внимание читателя на некоторых аспектах (например, естественной последовательности обработки сигналов), отсрочивая прочие аспекты (например, обработка особых случаев) до следующих страниц.

В течение перехода процессу явно неизвестно, который входной сигнал вызвал переход. Это может быть выведено только из контекста (то есть этот переход мог возникнуть только в том случае, если был получен конкретный сигнал). На рис. D-3.8.9 работа T1 выполняется только тогда, когда получен I1. Однако работа T2 будет выполнена при получении как I2, так и I3. Если работе T2 важно знать, который из вводов был получен, то тогда лучше описать процесс так, как он изображен на рис. D-3.8.10.

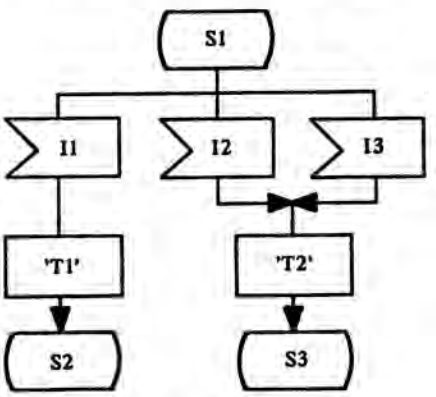


РИСУНОК D-3.8.9

Выполнение работы в зависимости от поступления двух из трех вводов, но независимо от того, который из этих двух воспринят

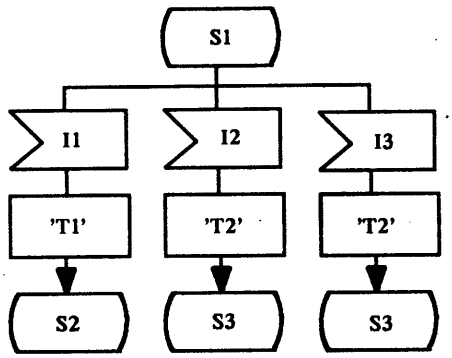


РИСУНОК D-3.8.10

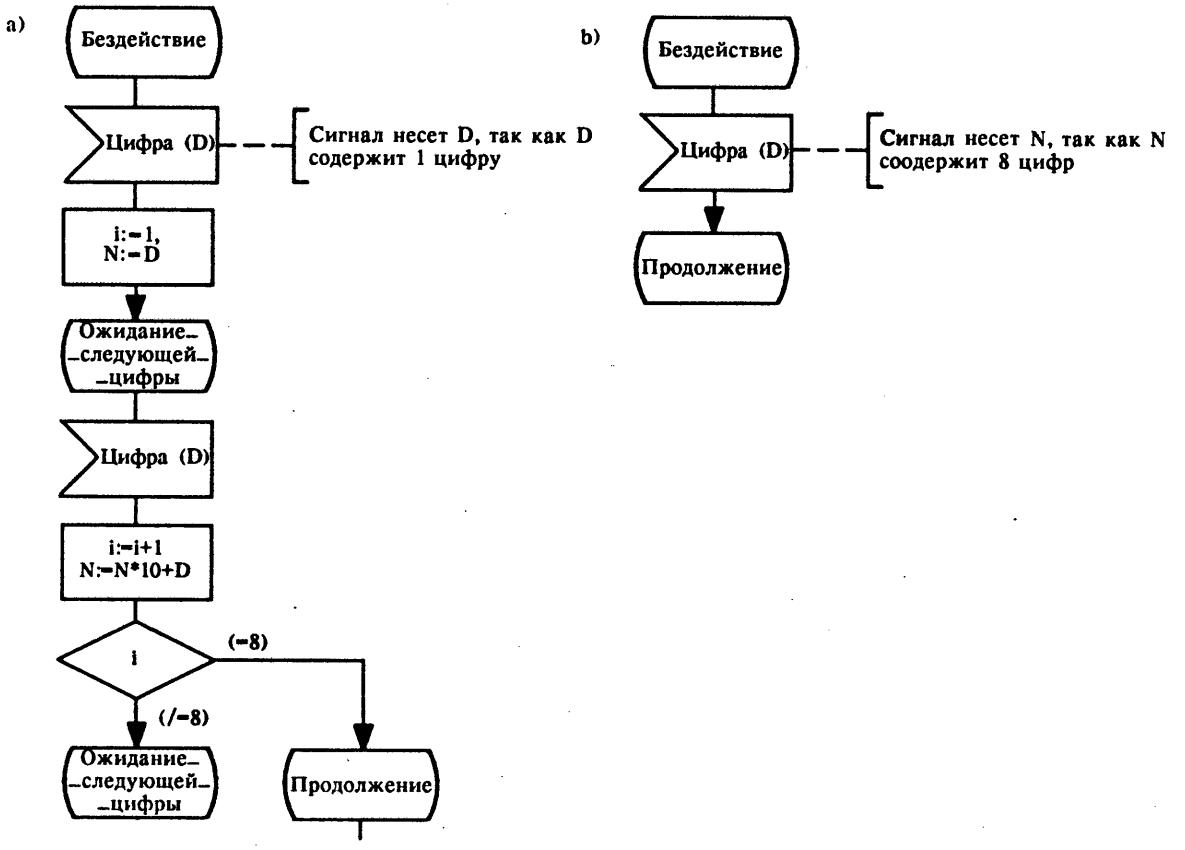
Выполнение работы в зависимости от полученного сигнала

D.3.8.2.1 *Выявление требуемых состояний*

Обычно разработчик диаграмм на SDL имеет некоторую свободу, вытекающую из подходов к определению состояний процесса. Он может выработать стратегию, дающую ему возможность выделения состояний процесса, причем эта стратегия может быть формальной или неформальной. Требуется строгий подход (формирующийся из практики), позволяющий разрабатывать диаграммы на SDL, которые не являются излишне сложными в силу выделения очень большого числа отдельных состояний и не используют присущие SDL достоинства в силу того, что искусственно понижают число состояний. Прежде чем разработчик начинает составление диаграммы, должны быть выполнены подготовительные шаги (рассмотренные в § D.3.2), к числу которых относятся, например:

- структурирование системы на функциональные блоки;
- представление функциональных блоков с помощью одного или более SDL-процессов на каждый блок;
- выбор входных и выходных сигналов;
- использование данных в процессе.

Все перечисленные факторы оказывают существенное влияние на выявление состояний каждого процесса. Влияние "выбора" входных сигналов на число состояний в SDL-диаграмме иллюстрируют два примера на рис. D-3.8.11.



Примечание. — Примеры a) и b) представляют одну и ту же функцию с разной степенью детализации. В результате этого меняется число состояний.

РИСУНОК D-3.8.11

Прием 8-разрядного телефонного номера

D.3.8.2.2 *Понижение числа состояний*

Использовав некоторую стратегию выявления состояний процесса, разработчик диаграммы на SDL может почувствовать, что получилось "слишком много состояний". Число состояний очень важно, так как часто сложность и размеры диаграммы непосредственно зависят от числа состояний. Могут существовать различные способы понижения числа состояний, но сама по себе сложность диаграммы не должна являться причиной ее преобразования, так как сложность диаграммы может быть просто отражением сложности, присущей данному процессу. Вообще набор состояний должен быть выбран так, чтобы обеспечить максимальную ясность во взаимодействии процесса с его окружающей средой. Как правило, такая ясность не достигается сокращением числа состояний. Число независимых последовательностей, осуществляемых процессом, оказывает мультипликативное влияние на число состояний. Поэтому желательно обрабатывать независимые последовательности в отдельных процессах, так как это понижает число состояний и повышает ясность.

Число состояний может быть понижено за счет выделения общих функций, слияния состояний, использования концепции процедур или концепции сервисов. Конкретная структура данных также может вызвать понижение числа состояний. Альтернативное представление может выиграть от использования макрокоманд.

D.3.8.2.3 *Выделение общих функций*

При планировании SDL-диаграмм может обнаружиться, что выделение какого-то конкретного, повторяющегося аспекта процесса потребует изображения повторяющихся состояний. На рис. D-3.8.12 приведена SDL-диаграмма процесса линейной сигнализации, на которой иллюстрируется следующее требование: чтобы сигнал линейной сигнализации мог считаться обнаруженным, он должен продолжаться в течение некоторого конкретного промежутка времени.

Для спецификации этого обстоятельства между состояниями Линейный_сигнал_не_обнаружен и Диалог требуется добавить промежуточное состояние. Допустим, что в полной диаграмме такая общая функция должна будет повторяться в каждом месте, где обнаруживается сигнал. Альтернативным решением будет определение отдельного процесса, ответственного за обнаружение сигнала линейной сигнализации, опирающееся на специфицированный отрезок времени распознавания сигнала. Наличие такого нового процесса позволит изобразить процесс, приведенный на рис. D-3.8.12, в форме, приведенной на рис. D-3.8.13. (В контексте приводимого примера эти два рисунка сделаны эквивалентными за счет введения нового сигнала Действительный_линейный_сигнал.)

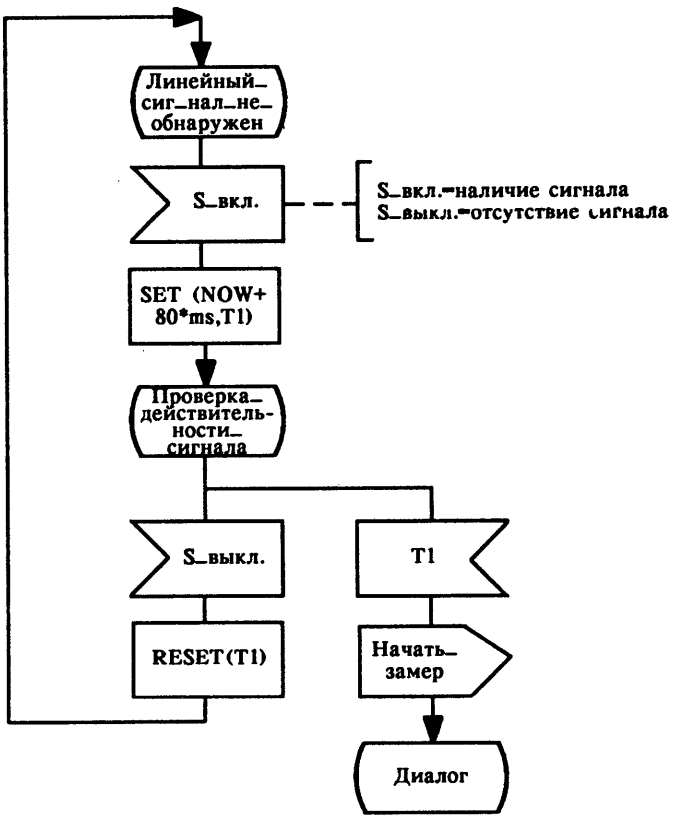
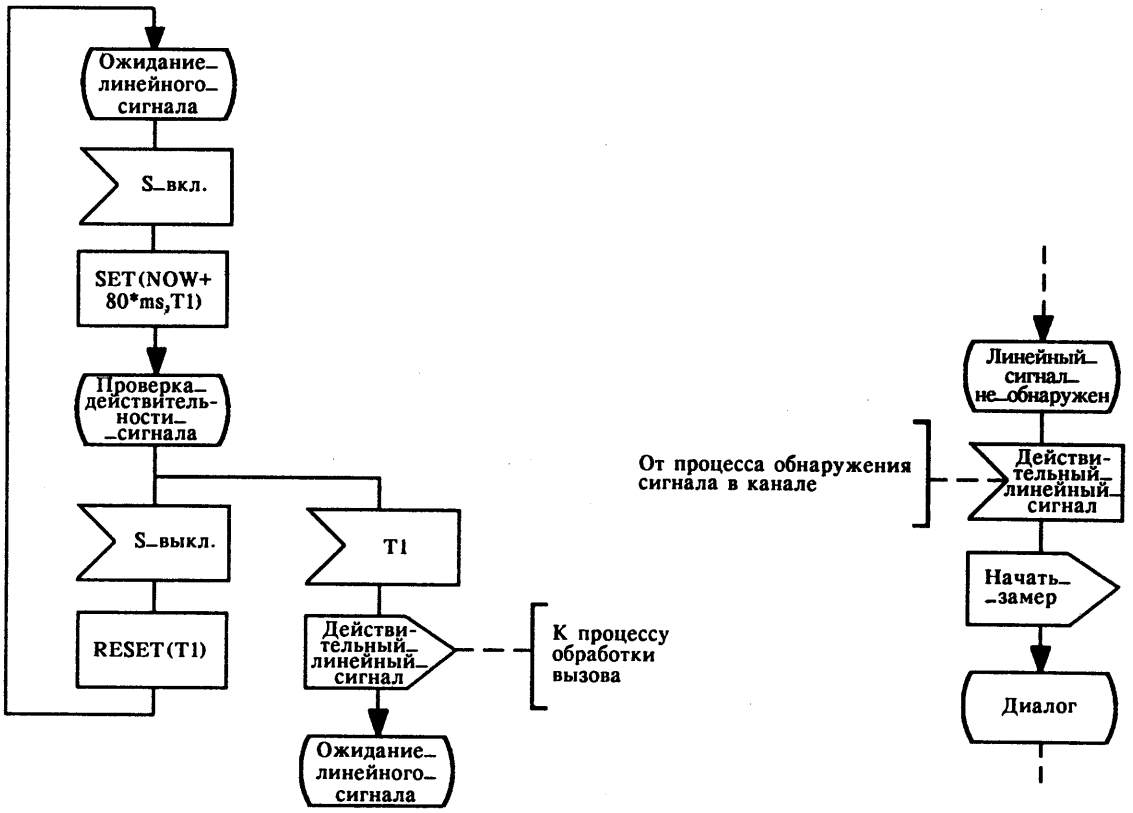


РИСУНОК D-3.8.12

Пример SDL-диаграммы, объединяющей обработку вызова и обнаружение линейного сигнала



a) Процесс обнаружения сигнала в канале

b) Процесс обработки вызова

РИСУНОК D-3.8.13

Пример выделения общей функции (обнаружение сигнала в канале), позволяющего исключить повторяющиеся состояния, такие как проверка_действительности_сигнала

Следует отметить небольшое различие между процессом, приведенным на рис. D-3.8.12, и двумя процессами, приведенными на рис. D-3.8.13.

Процесс на рис. D-3.8.12 начинает измерения немедленно по получении T1, в то время как процесс обработки вызова (процесс b) на рис. D-3.8.13) начинает измерения по получении сигнала Действительный_линейный_сигнал, а этот последний вырабатывается и посылается процессом a) на рис. D-3.8.13 после получения им сигнала T1. Из этого следует, что во втором случае измерения начинаются в момент времени, равный наступлению T1 + время выработки, посылки и приема сигнала. Кроме того, может оказаться, что за время, ушедшее на выработку и посылку Действительный_линейный_сигнал, появились и были поставлены в очередь другие сигналы.

Следует отметить еще одно обстоятельство. Такое выделение подфункции будет невозможным, если сигнал, который должен быть получен подфункцией, должен быть получен еще и самой основной функцией, так как сигнал всегда направлен только одному процессу. Если сигнал S_выкл. в приведенном примере должен быть воспринят еще и в другом состоянии, в котором его не надо было проверять на действительность в течение времени t1, то выделение подфункции проверки действительности сигнала другому процессу было бы невозможным.

Вообще говоря, решение с выделением процесса оказывается действенным только в тех случаях, когда характер обработки сигнала не зависит от состояний в основном процессе. В этих случаях пред- и пост-процессы могут осуществлять детальную последовательность действий, освобождая основной процесс от этих деталей. Это часто создает и полезную модуляцию, поскольку характеристики, например, систем сигналов могут быть отделены от основного процесса, более ориентированного на обслуживание.

Другой подход к решению этого вопроса заключается в использовании концепции сервисов, разъясняемой в § D.5.3.

Еще один подход к решению этого же вопроса заключается в использовании нотации MACRO, как это разъясняется в § D.3.3.1. При этом подходе мы достигаем требуемой компактности диаграмм, не внося при этом ни малейших изменений в их семантику. Более того, вызов MACRO можно осуществлять из различных состояний, если того требует логика процесса.

D.3.8.2.4 *Слияние состояний*

Если в некоей SDL-диаграмме будущее двух каких-то состояний совпадает, то независимо от предыстории этих состояний, они могут быть слиты в одно без малейшего влияния на логику диаграммы.

На рис. D-3.8.14 приведен фрагмент SDL-диаграммы, два состояния которой отличаются своей предысторией, но будущее которых "идентично". На рис. D-3.8.15 эти два состояния объединены в одно. Это довольно тривиальный пример, в котором уменьшение сложности невелико, но применение этого метода может дать существенные упрощения. Семантика SDL не дает возможности определить предысторию некоторого состояния с помощью принятия решения, следующего за данным состоянием (например, был ли в приводимом примере послан A6 или B4?), если только эта информация не была явно сохранена до вхождения в состояние. Следует заметить, что при названии данных во входном сигнале вызывается запоминание значения.

Состояния должны четко соответствовать возможным логическим ситуациям в процессе, и поэтому сливать логически различные ситуации в одно состояние не рекомендуется.

Нужно проявить осторожность, если диаграмма, в которой было осуществлено слияние, позже подвергается каким-то изменениям. Пользователь должен проверить, повлияют ли предполагаемые изменения на две (или более) исходные ветви таким же образом или нет.

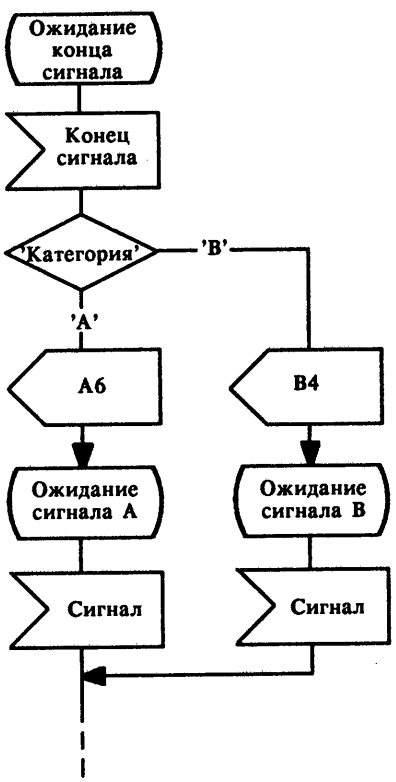


РИСУНОК D-3.8.14

Пример фрагмента SDL-диаграммы состояний до слияния

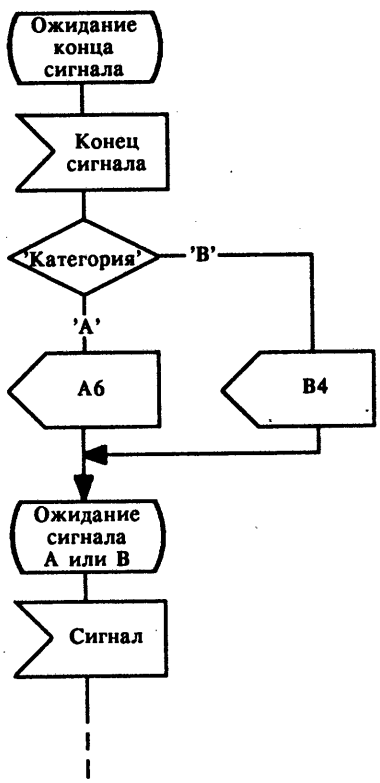


РИСУНОК D-3.8.15

Пример SDL-диаграммы со слитыми состояниями

D.3.8.3 *Вводы*

Параграф D.3.8.3 был написан с тем, чтобы разъяснить концепцию ввода и использование вводов в SDL-диаграммах, но без концепции сохранения. Концепция сохранения и совместное использование в SDL-диаграммах вводов и сохранений разъясняются в § D.3.8.4.

D.3.8.3.1 *Общие положения*

Символ ввода, присоединенный к состоянию, означает, что если к процессу, находящемуся в положении ожидания в данном состоянии, прибывает сигнал, имя которого стоит внутри символа ввода, то должен быть интерпретирован переход, следующий за символом ввода. После того, как сигнал вызвал интерпретацию перехода, он перестает существовать; говорят, что он был воспринят.

Сигнал может иметь связанные с ним значения. Например, сигнал, именуемый "цифра", служит не только для того, чтобы вызвать выполнение перехода в процессе, получившем этот сигнал, но и для того, чтобы вместе с собой доставить значение цифры (0–9), которое будет использовано процессом.

На SDL/PR предложение INPUT содержит список идентификаторов сигналов. Значения, содержащиеся в сигнале, именуются посредством идентификаторов переменных. Идентификаторы переменных должны принадлежать сорту, указанному в определении сигнала, и потому их расположение очень существенно. Эти идентификаторы переменных заключаются в круглые скобки и отделяются друг от друга запятыми (см. рис. D-3.8.16). Если одно или более значений сигнала аннулируются, то соответствующие переменные отсутствуют; это изображается двумя или более следующими друг за другом запятыми (см. рис. D-3.8.17).

```
...
SIGNAL sig1 (INTEGER,BOOLEAN,INTEGER);
...
PROCESS...
    ...
    DCL a INTEGER, b BOOLEAN, c INTEGER; /*объявления*/
    ...
    STATE st1;
        INPUT sig1(a,b,c); /*корректный ввод*/
        ...
    STATE st2;
        INPUT sig1(a,c,b,); /*некорректный ввод*/
        ...
ENDPROCESS...
```

РИСУНОК D-3.8.16

Предложения INPUT

```
INPUT a(var1,var2,,var4);
```

Примечание. – В этом примере третье значение сигнала аннулировано.

РИСУНОК D-3.8.17

Ввод сигнала a, в котором присутствуют только 3 из 4 значений, определенных для этого сигнала

На SDL/GR ввод изображается с помощью символа ввода, содержащего список идентификаторов сигналов и соответствующие идентификаторы переменных для транспортируемых значений. Чтобы эти значения стали доступны процессу, в символе ввода, в скобках, должны стоять их имена.

Примеры приема значений от вводов приведены на рис. D-3.8.18, D-3.8.19 и D-3.8.20.

Названные значения становятся доступными процессу приема при интерпретации ввода.

На рис. D-3.8.18 приведен прием сигнала Трубка_снята. С сигналом Трубка_снята связано число 9269 (Номер абонента). Этот сигнал может быть получен тремя способами, как это показано в пунктах a) – c) рисунка.

На рис. D.3.8.19 показано, как с помощью одного сигнала послать и получить несколько значений. Каждый элемент должен быть отделен от следующего запятой. На рис. D-3.8.20 c) мы показываем, как игнорировать нежелательные значения, оставляя пустое место в списке сортов.

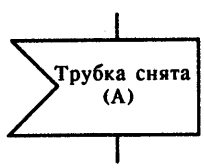
Обратите внимание на то, что в выходных сигналах на место E1, E2 или E3 можно написать выражения, но во входных сигналах для получения присылаемых значений надо использовать переменные.

В SDL нет нужды изображать символы ввода для представления сигналов, чье поступление вызовет нулевой переход (то есть переход, который не содержит никаких действий и ведет назад к тому же состоянию). По соглашению для любого сигнала, не указанного явно в символах ввода при некотором состоянии, существуют неявный символ ввода (при этом состоянии) и нулевой переход. В силу этого соглашения две диаграммы, приведенные на рис. D-3.8.21, логически эквивалентны; можно пользоваться любой из них.

a)

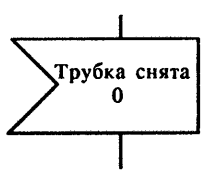
Примечание. — Значение 9269 сохраняется в переменной, имя которой "Номер абонента".

b)



Примечание. — Значение 9269 сохраняется в переменной, имя которой A.

c)

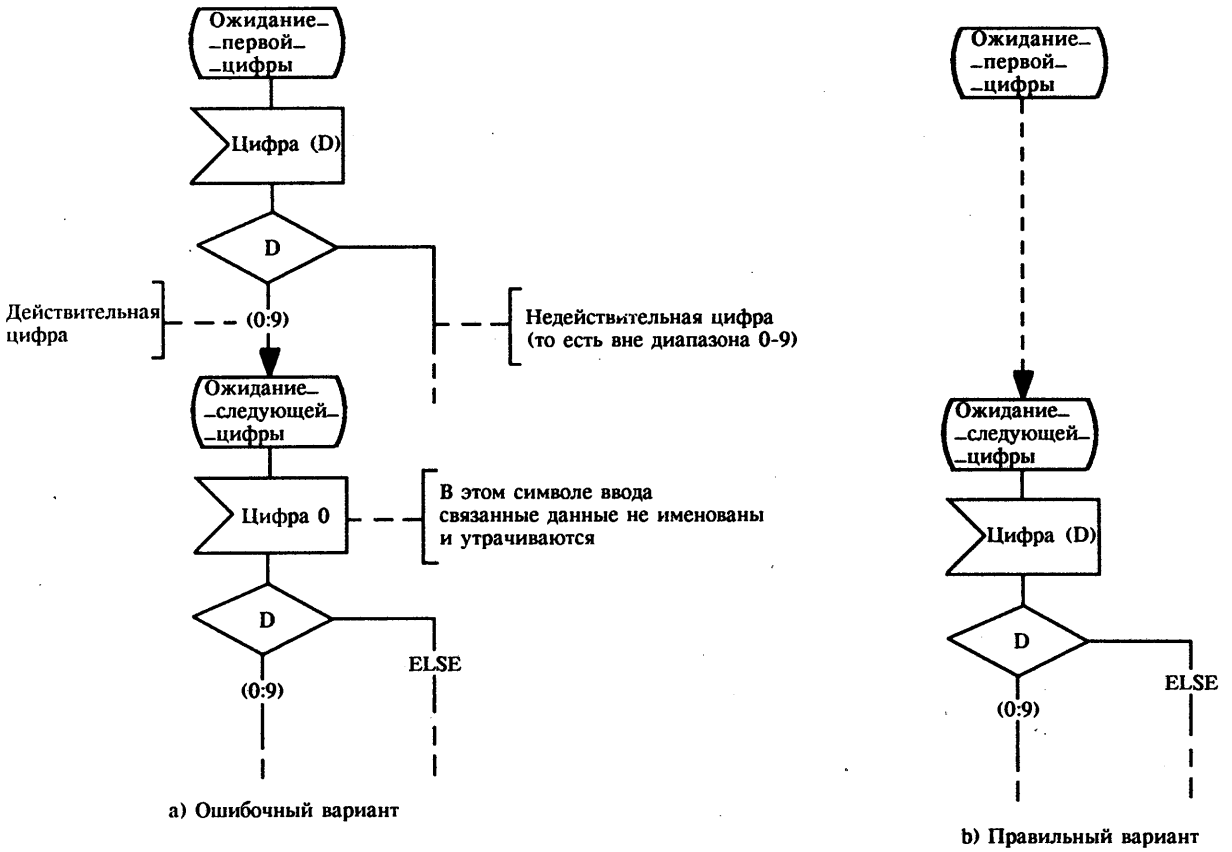


Примечание 1. — Здесь нет имени переменной, а потому значение 9269 утрачивается и не доступно получающему процессу.

Примечание 2. — Имя сигнала (Трубка_снята) должно соответствовать имени выходного сигнала, но имена данных не должны соответствовать.

РИСУНОК D-3.8.18

Примеры получения значений процессом



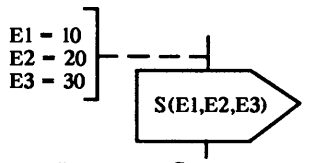
Примечание 1. — В варианте a) второе принятие решения использует значение D, полученное из первого сигнала.

Примечание 2. — В варианте b) значение D будет равно текущей цифре. Значения предшествующих цифр будут затерты.

РИСУНОК D-3.8.19

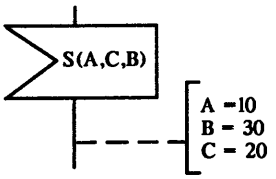
Фрагмент процесса получения цифр

a)



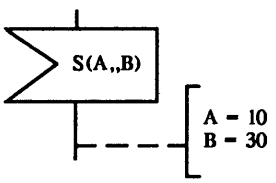
Примечание. — Выходной сигнал S содержит три переменные, обозначенные E1, E2 и E3. Эти элементы имеют три значения, например, равные в текущий момент 10, 20 и 30.

b)



Примечание. — В приемном процессе соответствующий входной сигнал S присваивает этим элементам имена A, C и B соответственно.

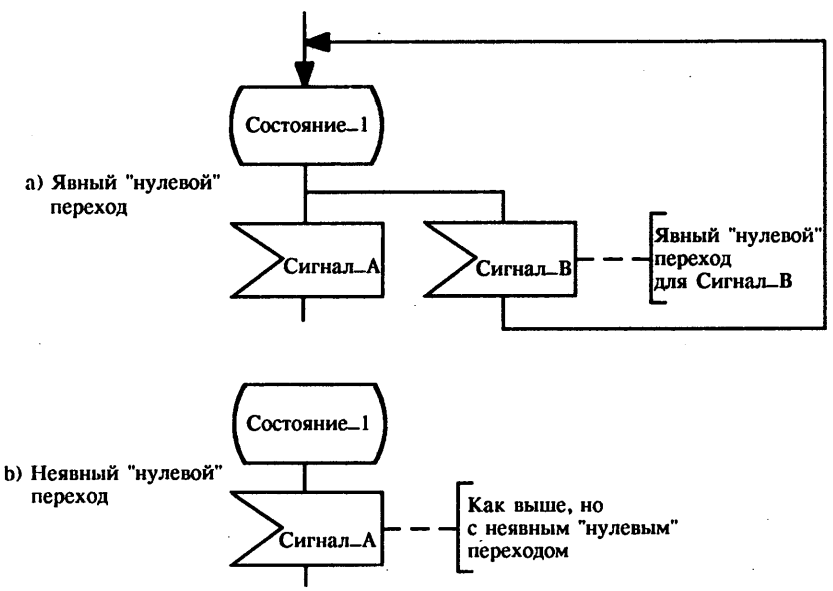
c)



Примечание. — Этот входной сигнал присваивает имена только двум переменным. Среднее значение утрачивается.

РИСУНОК D-3.8.20

Сигнал, с которым связано несколько значений

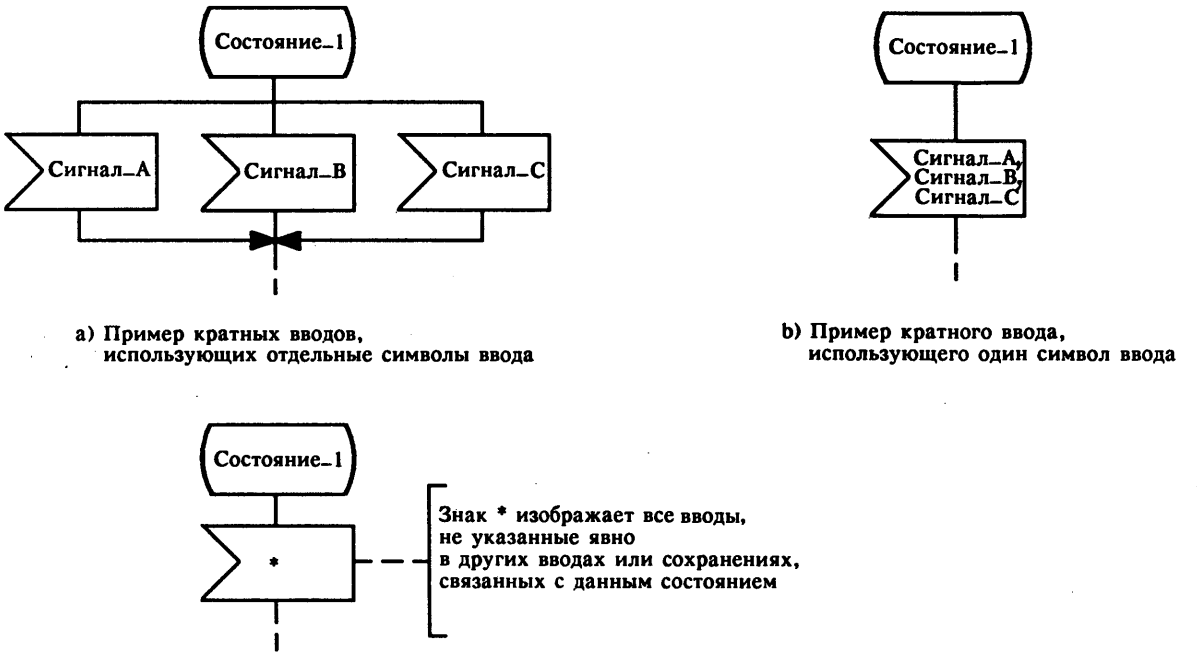


Примечание. — Если с сигналом Сигнал_B связаны данные, то в обоих случаях они утрачиваются. Если, однако, указать имя данных [например, сигнал_B(x)], для случая a) значение данных сохраняется. В этом случае эти два примера перестают быть идентичными.

РИСУНОК D-3.8.21

Явное и неявное изображение нулевого перехода

Если несколько вводов вызывают один и тот же переход, то идентификаторы всех связанных с этим сигналов могут быть помещены внутрь одного символа ввода. SDL предлагает сокращенную нотацию для изображения входа всех сигналов (действительных для данного процесса), явно не указанных для некоторого состояния. На рис. D-3.8.22 иллюстрируются эти возможности; все диаграммы этого рисунка логически эквивалентны. Если указываются идентификаторы всех сигналов, то они должны быть отделены друг от друга запятыми.



a) Пример кратных вводов, использующих отдельные символы ввода

b) Пример кратного ввода, использующего один символ ввода

c) Пример кратного ввода, использующего знак *

РИСУНОК D-3.8.22

Альтернативные изображения кратных вводов

D.3.8.3.2 *Неявный механизм образования очереди*

К моменту, когда процесс входит в новое состояние, уже несколько сигналов могут ждать восприятия. Это значит, что из сигналов должна быть каким-то способом образована очередь. Когда сигнал прибывает к своему приемному процессу, его помещают во входную очередь. В SDL для каждого процесса используется концептуальный механизм образования очереди "первым в – первым из": сигналы воспринимаются в порядке их прибытия к процессу. Когда процесс входит в какое-то состояние, ему поставляется один и только один сигнал из очереди. Из этого следует, что если очередь не пуста, то процесс воспринимает первый сигнал очереди. Если же очередь пуста, то процесс ждет, пока в очередь не прибудет сигнал, после чего он воспринимается процессом.

На рис. D-3.8.23 используется концептуальная очередь для того, чтобы объяснить функционирование SDL-процесса в случае, когда время перехода отлично от нуля. Следует отметить, что:

– Не используется концепция сохранения и поэтому сигналы воспринимаются в порядке их прибытия.

– Существенен порядок прибытия сигналов. Если бы за время перехода из Состояние_1 в Состояние_2 сигнал "C" прибыл до сигнала "B", то последовательностью состояний была бы 1, 2, 3, а не 1, 2, 4.

– Так как очередь не бывает пустой, когда процесс входит в каждое из состояний Состояние_2 и Состояние_4, то процесс не останавливается ни в одном из этих состояний.

– Не существует возможности присвоения сигналам приоритетов. Для межсервисных связей предусмотрен специальный механизм, так что сигналы, которыми обмениваются сервисы, обрабатываются раньше прочих сигналов (см. § D.5.3).

Если время перехода равно нулю, то всякий сигнал будет воспринят в момент его прибытия к процессу, при условии, что не используется операция сохранения (§ D.3.8.4).

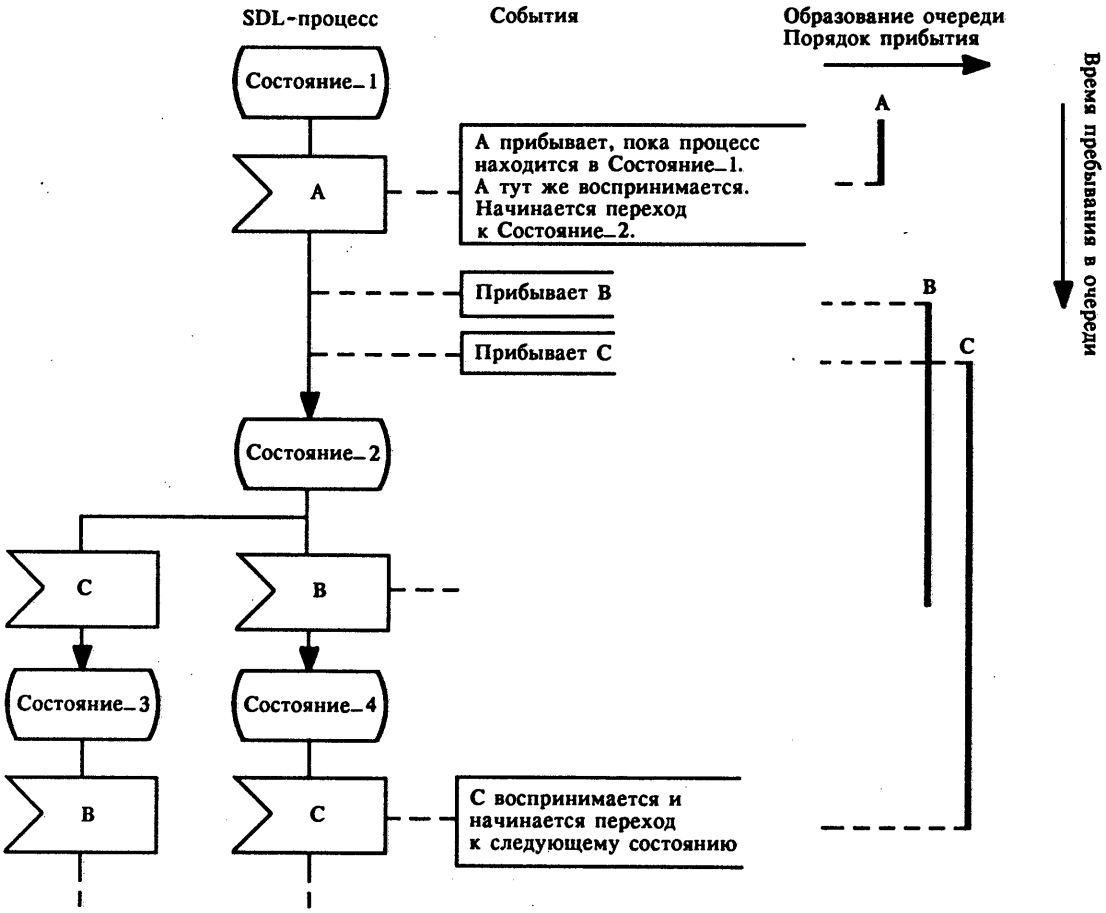


РИСУНОК D-3.8.23

Пример функционирования неявного механизма образования очереди

D.3.8.3.3 *Восприятие ошибочно прибывающих сигналов*

В каждом состоянии все возможные сигналы должны быть явно или неявно указаны. Нарушения (то есть прибытие сигналов, которые не должны были прибыть, но прибытие которых теоретически возможно) могут возникнуть почти во всех состояних. Обычно разработчик не описывает такие возможности, в результате чего такие сигналы, если они прибывают, оказываются аннулированными. Если же разработчик хочет включить такие нарушения в свои диаграммы, то тогда каждое состояние должно быть расширено экстравводом.

Другой способ – использование кратного вхождения состояния в сочетании с символом "все" (*). Например, если сигнал Трубка_А_повешена может быть получен во всех состояниях и если последующие действия идентичны, то может быть использован метод, приведенный на рис. D-3.8.24.

D.3.8.3.4 *Одновременное прибытие сигналов*

В §2 Рекомендации по SDL сказано, что сигналы могут прибыть к процессу одновременно и тогда их порядок будет произвольным.

Если пользователь разрабатывает процесс, который может получать одновременно прибывающие сигналы, он должен позаботиться о том, чтобы порядок прибытия сигналов не нарушил требующегося функционирования процесса.

SDL не определяет приоритета сигналов, то есть при одновременном прибытии сигналов один из них выбирается произвольно. Следует, однако, отметить, что сигналы межсервисных связей всегда обрабатываются первыми (см. § D.5.3).

Если при вхождении процесса в некоторое состояние доступными оказываются несколько сигналов, то только один из них поддается процессу и таким образом распознается в качестве ввода. Семантика SDL подразумевает, что все прочие сигналы фактически задерживаются.

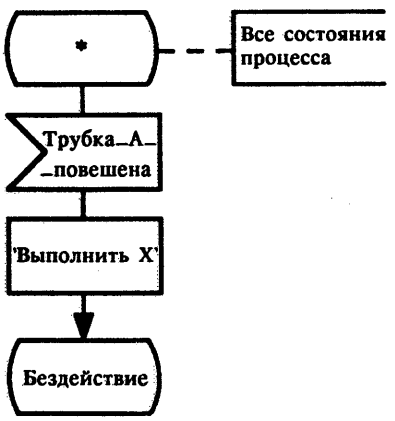


РИСУНОК D-3.8.24

Пример обработки сигналов, которая может иметь место в нескольких состояниях

D.3.8.3.5 *Идентификация отправителя*

Каждый сигнал несет с собой значение экземпляра процесса (PId) процесса-отправителя. После того, как сигнал воспринят, значение PId процесса-отправителя может быть получено с помощью выражения SENDER. Рис. D-3.8.25 показывает пример того, как можно это использовать.

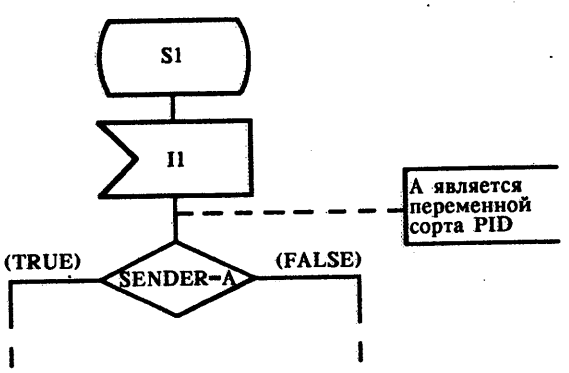


РИСУНОК D-3.8.25

Использование выражения SENDER

D.3.8.4 *Сохранения*

Концепция сохранения позволяет отсрочить восприятие сигнала до тех пор, пока не будут восприняты один или несколько других сигналов, прибывших позже. Как было отмечено в § D.3.8.3, если концепция сохранения не используется, то сигналы воспринимаются в порядке их прибытия.

Концепция сохранения может быть использована для упрощения процессов в тех случаях, когда относительный порядок прибытия некоторых сигналов несуществен, а фактический порядок прибытия сигналов не определен.

При каждом состоянии каждый сигнал описывается одним из следующих способов:

— он указывается в качестве ввода;

— он указывается в качестве сохранения;

— он трактуется как неявный ввод, ведущий к неявному нулевому переходу.

Функционирование неявного механизма образования очереди, описанного в § D.3.8.3, применимо и к концепции сохранения. По прибытии сигналы помещаются в очередь и когда процесс входит в некоторое состояние, то сигналы в очереди просматриваются по одному в каждый данный момент времени и в порядке их прибытия. Сигнал, описанный как явный или неявный ввод, воспринимается и выполняется соответствующий переход. Сигнал, описанный в сохранении, не воспринимается и остается в очереди, в том же относительном расположении, и рассматривается следующий в очереди сигнал. За сохранением не следует никакой переход.

На SDL/PR конструкция сохранения изображается с помощью ключевого слова SAVE, за которым следует список идентификаторов сигналов. Простой пример предложения SAVE приведен на рис. D-3.8.26.

```
...
STATE Состояние_31;
    SAVE s;
    INPUT r;
    NEXTSTATE Состояние_32;
STATE Состояние_32;
    INPUT s;
    ...
```

РИСУНОК D-3.8.26

Пример использования сохранения

На SDL/GR концепция сохранения изображается с помощью символа сохранения, содержащего идентификаторы сигналов.

Аналогично вводам может быть использовано "обозначение звездочкой", представляющее собой сохранение всех сигналов (действительных для данного процесса), не указанных явно при этом состоянии.

На рис. D-3.8.27 приведен пример SDL-процесса, содержащего символ сохранения. Надо подчеркнуть, что сигналы S и R воспринимаются в порядке R, S – обратном тому, в котором они были получены. Один символ сохранения может сохранять сигнал только до тех пор, пока процесс находится в том состоянии, при котором стоит символ и сохраняет его на время перехода процесса к следующему состоянию. В следующем состоянии сигнал будет воспринят посредством явного или неявного ввода (как это показано на рис. D-3.8.27), если только не возникает одна из следующих ситуаций: либо повторяется символ сохранения, содержащий имя сигнала, либо оказывается, что в неявной очереди имеется еще один сохраняемый сигнал, доступный для восприятия до данного (как это показано на рис. D-3.8.28).

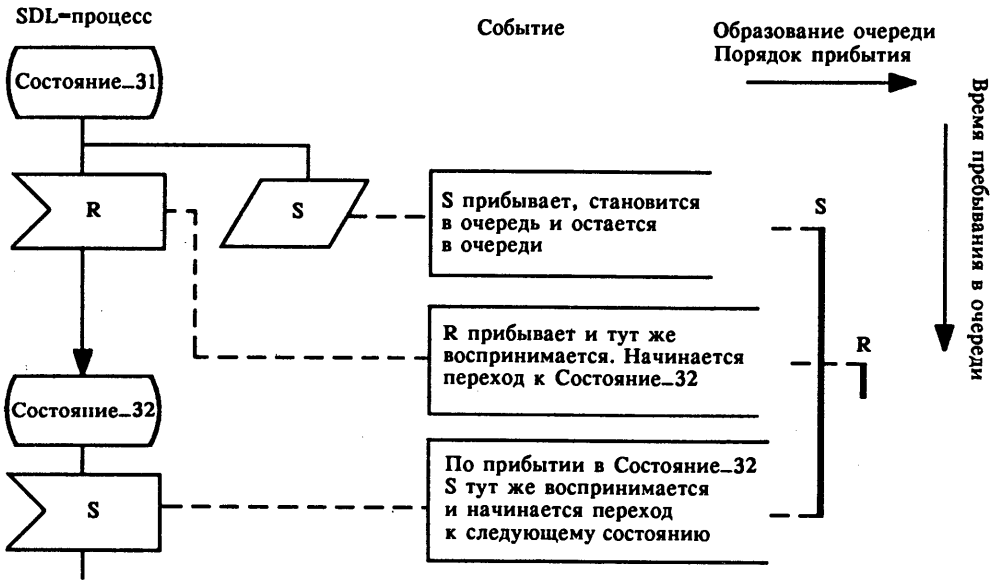


РИСУНОК D-3.8.27

Пример SDL-диаграммы, содержащей символ сохранения и иллюстрирующей неявный механизм образования очереди

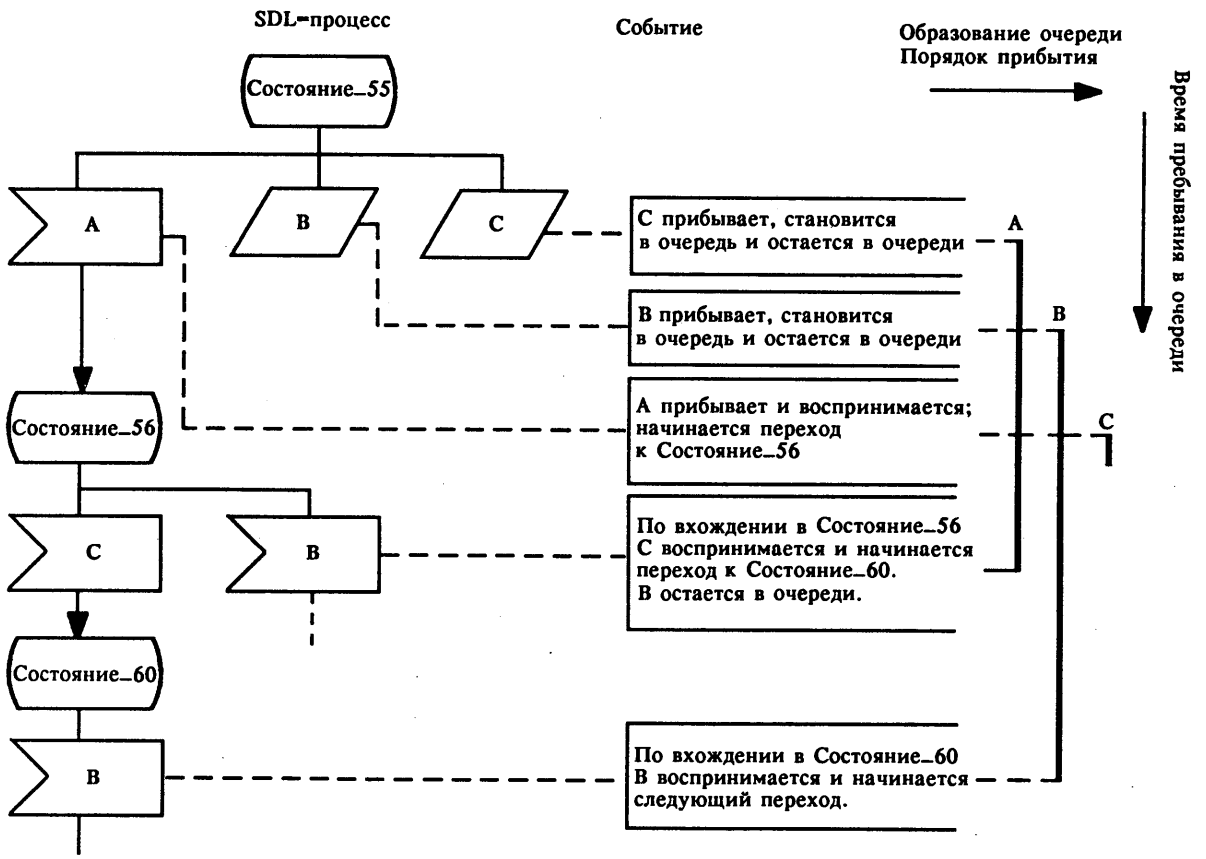


РИСУНОК D-3.8.28

Второй пример использования символа сохранения

Сохраненный сигнал становится доступным процессу только с помощью соответствующего символа ввода (явного или неявного). В частности, до того как будет воспринят сохраненный сигнал, никакие вопросы, касающиеся его, не могут быть заданы в принятии решения; в равной мере недоступны связанные с ним значения.

Если в некоем состоянии должно быть сохранено несколько сигналов, то можно либо дать отдельный символ сохранения для каждого сигнала, либо все они могут быть помещены внутри одного символа сохранения. Если сохранению подлежат несколько сигналов, то семантика символа сохранения подразумевает, что порядок их прибытия сохраняется.

Третий пример использования сохранения приведен на рис. D-3.8.29; функционирование концептуального механизма образования очереди приведено на рис. D-3.8.30.

Сохранение может быть использовано для упрощения диаграммы. Например, сохраняя сигнал, возможно отложить до следующего состояния получение сигнала и запись в память связанных с сигналом значений.

Хотя сохранение может быть использовано на любом уровне спецификации, на более низких уровнях может возникнуть необходимость в описании фактического механизма, реализующего концепцию сохранения.

Сохранением надо пользоваться с осторожностью. В противном случае очередь может стать слишком длинной и сигналы в ней будут сохраняться очень долго; в результате будет воспринят старый сигнал, когда ощущается потребность в свежем.

SDL не налагает никаких ограничений на длину очереди, что может вызвать трудности реализации.

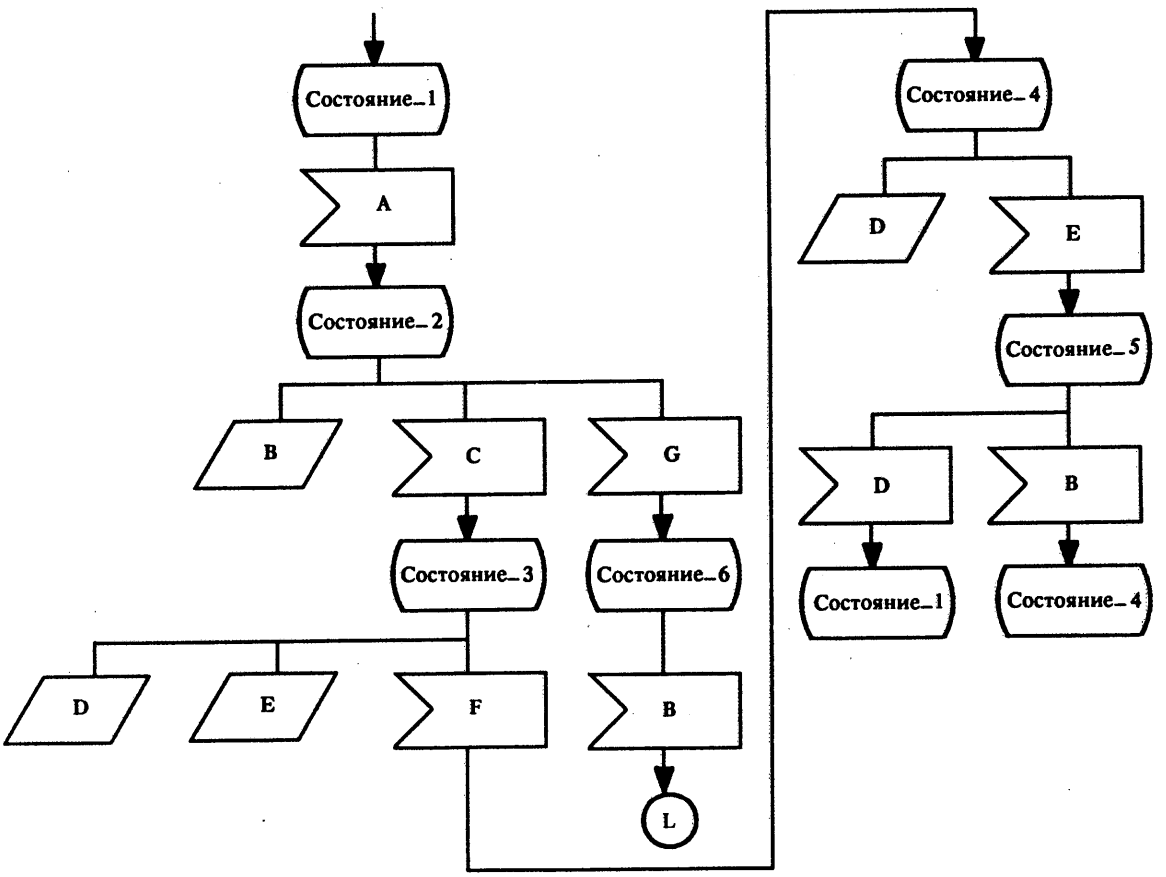


РИСУНОК D-3.8.29

Пример более сложной SDL - диаграммы

Текущее состояние	Событие	Образование очереди
Состояние_1	(Когда процесс входит в Состояние_1, в очереди уже находятся сигналы A, B, C, D, E). Первый сигнал в очереди, A, воспринят и начинается переход к Состояние_2.	Порядок прибытия → A B C D E F Время пребывания в очереди ↓
Состояние_2	Первый сигнал в очереди, B, находится в символе сохранения и остается в очереди.	
Состояние_2	Второй сигнал, C, воспринимается (явный ввод) и начинается переход к Состояние_3.	
Состояние_3	Первый сигнал в очереди, B, воспринят (неявный ввод).	
	Сигнал F прибывает и становится в очередь.	
Состояние_3	(При повторном входе в Состояние_3) первый сигнал в очереди, D, находится в символе сохранения и остается в очереди.	
Состояние_3	Второй сигнал, E, находится в символе сохранения и остается в очереди.	
Состояние_3	Третий сигнал, F, воспринят (явный ввод) и начинается переход к Состояние_4.	
Состояние_4	Первый сигнал в очереди, D, находится в символе сохранения и остается в очереди.	
Состояние_4	Второй сигнал, E, воспринят (явный ввод) и начинается переход к Состояние_5.	
Состояние_5	Первый (и единственный) сигнал в очереди, D, воспринят (явный ввод) и начинается переход к Состояние_1.	

РИСУНОК D-3.8.30

Функционирование механизма образования очереди

D.3.8.5 *Разрешающие условия и непрерывные сигналы*

Разрешающие условия обеспечивают возможность условного восприятия сигнала, опирающегося на специфицированное разрешающее условие. Если условие истинно, то сигнал воспринимается и переход интерпретируется. Если условие ложно, то сигнал сохраняется и процесс остается в том же состоянии до тех пор, пока либо не прибудет другой сигнал, либо условие не изменит значение с ложного на истинное. Это можно проиллюстрировать примером на рис. D-3.8.31. Когда P1 входит в состояние S1, вычисляется условие [то есть равняется ли IMPORT (X, P2) десяти]. Если условие истинно, то сигнал B может быть воспринят. В противном случае сигнал B сохраняется. В данном примере прибывает A и вызывает переход к состоянию S2. За время перехода значение X изменилось, стало равным 11, и P2 экспортировал его новое значение; теперь условие, подсоединенное к сигналу B (при состоянии S2) стало истинным. Так как B является первым сигналом в очереди, то выполняется следующий за ним переход и процесс завершается в состоянии S3.

Некоторыми важными свойствами разрешающего условия являются:

1) Разрешающее условие проверяется тогда, когда процесс входит в состояние и потом непрерывно опрашивается, пока процесс находится в этом состоянии. Так, если бы экспортируемое значение X изменялось от 9 к 11, а потом к 12, за время выполнения перехода восприятия A процесс остался бы в состоянии S2.

2) Разрешающие условия могут основываться на локальных переменных и/или любых языковых конструкциях, которые могут быть включены в выражения (например, IMPORT, VIEW, NOW).

3) В то время как при одном и том же состоянии можно использовать более одного разрешающего условия, нельзя использовать более одного разрешающего условия при одном и том же сигнале. Поэтому условие, показанное на рис. D-3.8.32, является недопустимым. Если при одном и том же сигнале требуется несколько условий, то их можно объединить в одно булевское выражение, как это показано на рис. D-3.8.33.

Разрешающие условия можно вычислять по нескольку раз и в любом порядке; поэтому пользователь должен быть осторожен, если выражения обладают взаимным побочным эффектом (например, комбинируя IMPORT и SENDER).

Следует отметить также, что сигнал, специфицированный в разрешающем условии, не может влиять на булевское значение условия, так как транспортируемые им значения не присваиваются до восприятия сигнала. Например, предложения:

```
INPUT x(A) PROVIDED (A=5); ...
INPUT y PROVIDED (SENDER=pid1);
```

ведут к ошибке, так как значение A и значение SENDER, указанные в условиях, соответствуют ситуации до восприятия сигнала.

Непрерывный сигнал имеет те же основные свойства, что и разрешающее условие, с той лишь разницей, что никакие сигналы не связаны с ним. Если процесс входит в некоторое состояние и при этом очередь оказывается пустой, то проверяется непрерывный сигнал. Если он истинен, то выполняется следующий за ним переход. Это можно проиллюстрировать примером на рис. D-3.8.34. Вначале процесс находится в состоянии S1, а экспортируемое значение X равно 9. Прибытие сигнала A вызывает переход в состояние S2. В течение перехода значение X меняется и становится равным 11; теперь экспортируется его новое значение. Так как в очереди нет других сигналов, то совершается переход в состояние S3.

Некоторыми важными свойствами непрерывного сигнала являются:

1) так же, как и для разрешающего условия, значение условия проверяется только по прибытии процесса в состояние или при пребывании в состоянии;

2) для каждого состояния разрешены кратные непрерывные сигналы. Если с состоянием связаны более чем один непрерывный сигнал, первым будет опрошен сигнал с наибольшим приоритетом (с наибольшим номером приоритета). Никакие два непрерывных сигнала не могут иметь одинаковые номера приоритетов. При всех условиях приоритет непрерывного сигнала ниже приоритета любого другого сигнала. И вновь это связано с системой, лежащей в основе SDL; метод моделирования в SDL непрерывных сигналов, использующий сигналы, посылаемые при экспортировании переменных, допускает наличие приоритетов в непрерывных сигналах и, по существу, делает их наличие обязательным для избежания неоднозначности, если имеется более одного непрерывного сигнала. Это иллюстрируется на рис. D-3.8.35. Первоначально процесс находится в состоянии S1, а его импортирующее выражение присваивает X и Y значение 10 и 11 соответственно. Так как оба непрерывных сигнала истинны, то выбирается тот из них, чей приоритет выше (номер приоритета ниже), и совершается переход в состояние S2. В состоянии S2 условие на Y более не выполняется, и хотя приоритет непрерывного сигнала для X ниже приоритета для Y, выполняется переход, следующий за этим последним, и процесс входит в состояние S3.

3) При выполнении перехода от непрерывного сигнала значение SENDER совпадает со значением SELF.

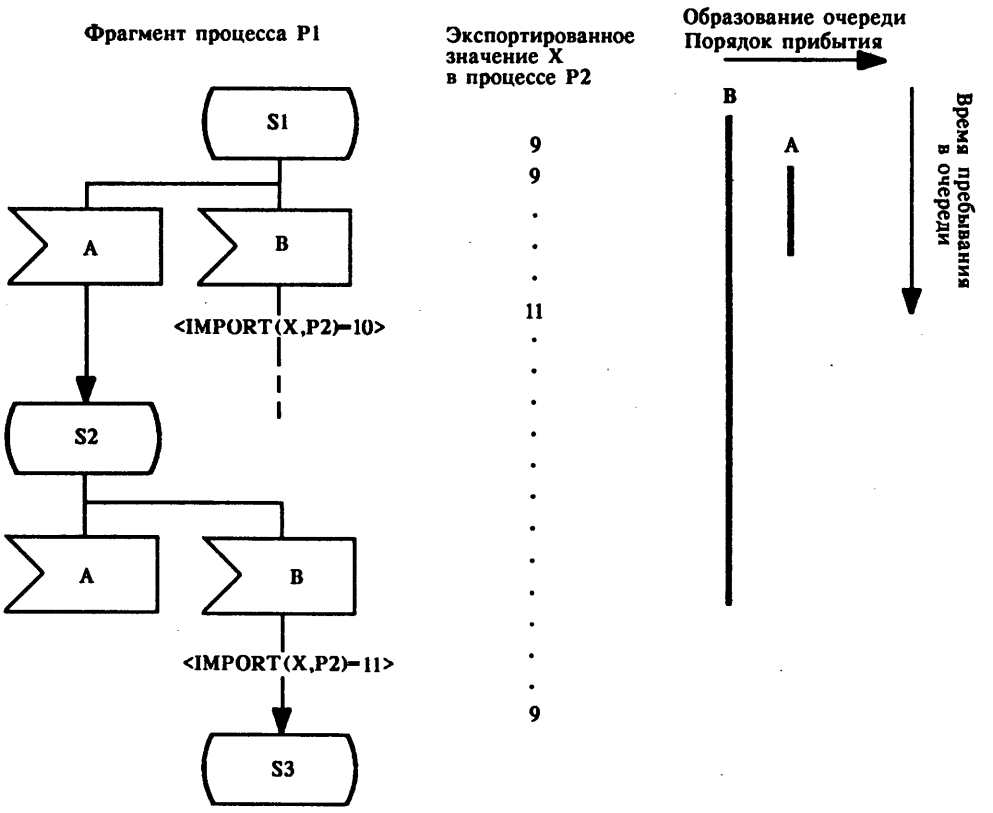


РИСУНОК D-3.8.31

Разрешающее условие

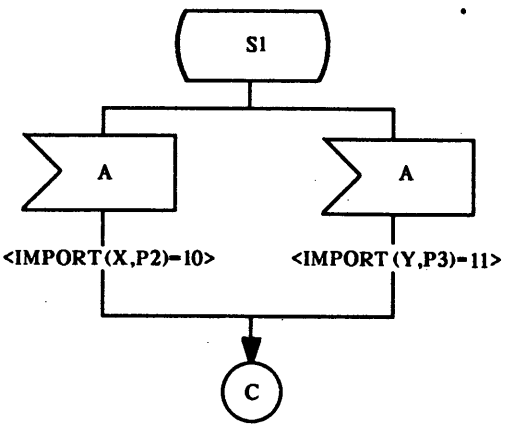


РИСУНОК D-3.8.32

Недопустимое разрешающее условие

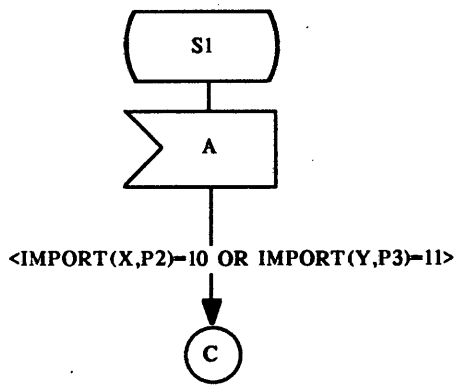


РИСУНОК D-3.8.33

Корректное изображение процесса на рисунке D-3.8.32

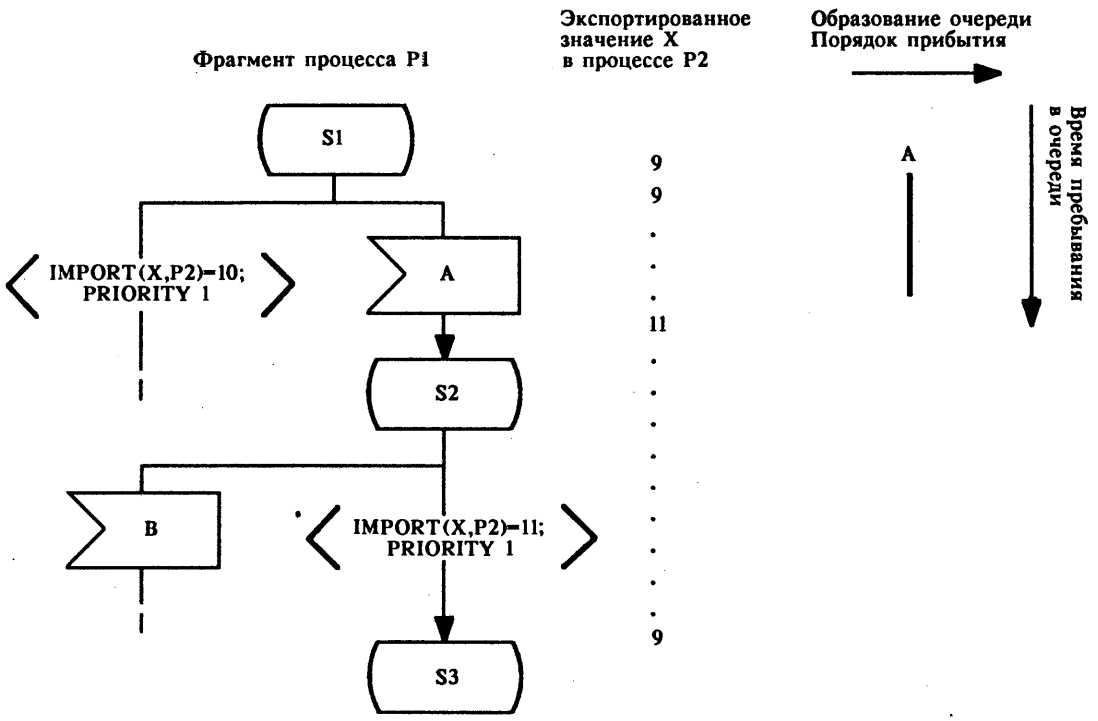


РИСУНОК D-3.8.34

Непрерывные сигналы

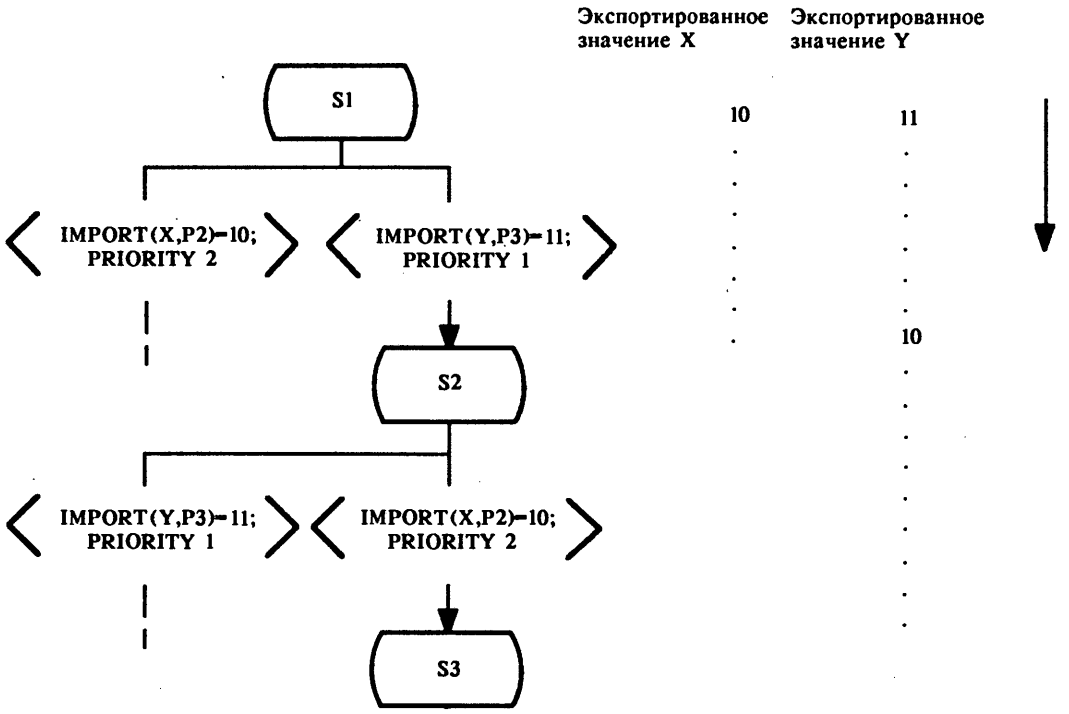


РИСУНОК D-3.8.35

Непрерывный сигнал с приоритетами

D.3.8.6 *Выводы*

Выводом является отправка сигнала от одного процесса к другому (или самому себе). Так как управление восприятием сигналов связано с получающим процессом (см. § D.3.8.3), то семантика непосредственно самого вывода относительно проста. С точки зрения посылающего процесса очень часто вывод может рассматриваться как мгновенное действие, которое, будучи выполненным, в дальнейшем не оказывает никакого влияния на посылающий процесс и он не будет непосредственно осведомлен о судьбе сигнала.

Выводное действие представляет собой отправку сигнала и связанных с ним значений. Значения могут быть связаны с выходным сигналом либо путем помещения значений в круглые скобки, либо путем помещения в скобки выражений, несущих значения (см. рис. D-3.8.37).

На SDL/PR выводное действие изображается с помощью ключевого слова OUTPUT, за которым следует список идентификаторов сигналов. С каждым идентификатором сигнала может быть связан заключенный в круглые скобки список фактических параметров. Если вывод не содержит фактического параметра, соответствующего некоторому конкретному сорту в определении сигнала, то соответствующая переменная в воспринимающем выводе получает "неопределенное" значение.

Экземпляр воспринимающего процесса должен быть выражен в предложении вывода с помощью ключевого слова TO, за которым следует выражение экземпляра процесса. Если экземпляр приемного процесса может быть однозначно определен из контекста, то клаузула TO может быть опущена. Дополнительно может быть обеспечено средство адресации в предложении вывода с помощью ключевого слова VIA, за которым следует список идентификаторов маршрутов сигналов или каналов.

На рис. D-3.8.36 приведено несколько примеров корректных предложений вывода.

```
OUTPUT sig1(2,true,10);
OUTPUT sig2,sig3(par1,par2) TO процесс_a;
OUTPUT sig4 TO процесс_b VIA канал_x,канал_y;
OUTPUT sig5 VIA маршрут_сигнала_z;
```

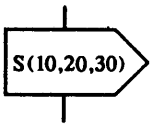
РИСУНОК D-3.8.36

Примеры предложений вывода

На SDL/GR вывод изображен с помощью символа вывода, содержащего спецификацию сигналов, фактических параметров и двух необязательных конструкций: приемника и/или VIA.

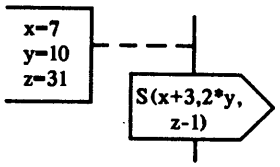
Каждый вывод должен направляться конкретному экземпляру процесса. Обычно при разработке спецификации невозможно знать экземпляр процесса любого процесса, поэтому, как правило, для направления сигналов применяют метод, заключающийся в том, что в ключевом слове TO используется переменная или выражение. Некоторые примеры приведены на рис. D-3.8.38, D-3.8.39 и D-3.8.40. На рис. D-3.8.38 параметру out_to процесса значение экземпляра процесса присваивается в момент создания процесса. Затем out_to используется в процессе в качестве связующего звена между этим процессом и адресуемым процессом. При разработке системы нужно убедиться, что тип процесса, заданного с помощью out_to, способен воспринять сигналы, которые будут ему посланы. На рис. D-3.8.39 встроенное выражение SENDER используется для отправки сигнала обратно тому процессу, который прислал сигнал. На рис. D-3.8.40 сигнал адресуется тому процессу, который является последним процессом, порожденным данным процессом.

a)



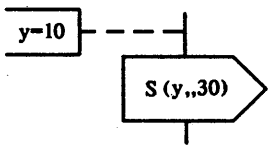
Примечание. — Сигнал S содержит три связанных с ним значения: 10, 20 и 30.

b)



Примечание. — В момент интерпретации вывода значения x, y и z равны (в этом примере) соответственно 7, 10, 31. Вывод посылает значения 10, 20, 30.

c)



Примечание. — В момент интерпретации вывода значение y равно (в этом примере) 10. Вывод посылает значение 10, неопределенное значение и значение 30.

РИСУНОК D-3.8.37

Вывод, содержащий связанные с ним значения

```
PROCESS X (0,5);
    FPAR (out_to PID);
    . . .
    OUTPUT sig TO out_to;
    . . .
```

РИСУНОК D-3.8.38

Направление сигналов с помощью формальных параметров

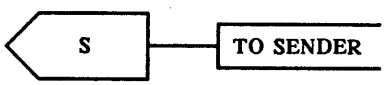


РИСУНОК D-3.8.39

Направление сигнала обратно к SENDER

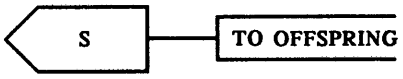


РИСУНОК D-3.8.40

Направление сигнала процессу-отпрыску

D.3.8.7 *Работа*

Работа используется в переходе либо для представления операций над переменными, либо для представления специальных операций с помощью неформального текста.

На SDL/PR работа изображается с помощью ключевого слова TASK, за которым следует список предложений или неформальных текстов, разделенных запятыми и завершающихся точкой с запятой. Предложением, входящим в работу, может быть просто присвоение. Неформальным текстом является фраза, заключенная в апострофы. На рис. D-3.8.41 приведено несколько примеров корректно написанных работ на SDL/PR.

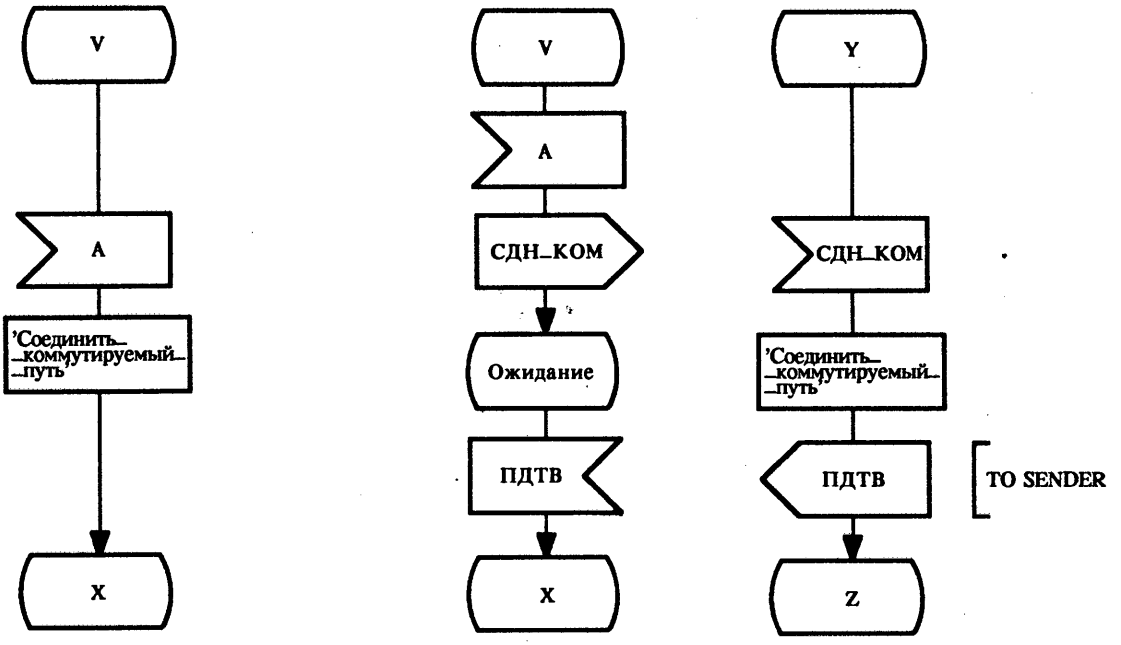
```
TASK a:=b;

TASK 'подключить абонента';

TASK c:=d+e;

TASK var1:=var2*var3,
     var4:=var5 MOD var6;
```

РИСУНОК D-3.8.41

Примеры работ

На SDL/GR работа состоит из символа работы, содержащего список предложений или неформальных текстов.

У пользователя SDL могут временами возникать трудности при решении, стоит ли изображать какие-то аспекты описываемой системы в виде работы или это лучше сделать с помощью отдельного процесса. Рассмотрим процесс, изображенный на рис. D-3.8.42; следует ли "соединить_коммутируемый_путь" изобразить в виде работы или в виде отдельного процесса? Если отдельный управляющий процесс для соединения коммутируемого пути не был определен, тогда подходящим будет символ работы [см. рис. D-3.8.42 a)]. Если же такой процесс определен, то тогда следует использовать сигналы, связывающиеся с управляющим процессом [см. рис. D-3.8.42 b)].



a) Решение, использующее один процесс

b) Решение, использующее два процесса

РИСУНОК D-3.8.42

Два возможных решения для 'соединить_коммутируемый_путь'

D.3.8.8 *Принятие решений*

Принятие решения является действием, выполняемым в течение перехода; действие заключается в выдаче вопроса, касающегося значения некоего выражения в момент выполнения действия. Процесс продолжается по одной из двух или большего числа ветвей; выбор ветви зависит от ответа. Разработчики SDL-диаграммы должны удостовериться в том, что процесс определен таким образом, что они не пытаются принять решение, для которого ответ не является доступным; такие решения привели бы к недействительности диаграммы и могли бы вызвать значительные осложнения.

Вопрос, стоящий в принятии решения, может иметь форму выражения или неформального текста. Ответы представляются либо в форме одного или нескольких возможных значений, полученных вычислением выражения, стоящего в вопросе, либо в форме одного или более неформальных текстов. Если вопрос или один из ответов являются неформальными, то тогда вся конструкция принятия решения является неформальной. Различные ответы разделены запятыми. Значения представлены постоянными выражениями, постоянными выражениями с операторами в качестве префикса или диапазона, верхние и нижние границы которых являются постоянными выражениями. Значения ответов должны быть того же сорта, что и вопрос, входящий в принятие решения.

Допустимо указывать некоторые ответы явно, а все остальные возможные ответы объединять в одну группу с помощью ключевого слова ELSE.

На SDL/PR принятие решения представляется с помощью ключевого слова DECISION, за которым следует спецификация вопроса и список возможных ответов, с каждым из которых связан соответствующий переход. Ответы помещаются в круглые скобки. Набор выходных переходов ограничен в конце ключевым словом ENDDECISION (см. рис. D-3.8.43).

```
DECISION вопрос;
     (ответ_a): . . .
     (ответ_b): . . .
     (ответ_c): . . .
     ELSE: . . .
ENDDECISION;
```

РИСУНОК D-3.8.43

Структура принятия решения

Некоторые примеры принятия решения приведены на рис. D-3.8.44.

```
. . .
DCL x INTEGER,a BOOLEAN;
. . .
DECISION x;
     (2): . . .;
     (2+5): . . .;
     (6+8,10+9): . . .;
     (-3): . . .;
     (20:30): . . .;
     (>=100): . . .;
     ELSE: . . .;
ENDDECISION;
. . .
DECISION a;
     (TRUE) : NEXTSTATE s1;
     (FALSE) : NEXTSTATE s2;
ENDDECISION;
. . .
DECISION 'Категория абонента';
     ('Межнациональный','Национальный'): . . .
     ('Локальный'): . . .
ENDDECISION;
. . .
```

РИСУНОК D-3.8.44

Примеры принятия решения на SDL/PR

Все переходы заканчиваются ключевым словом ENDDECISION. Продолжение перехода, не заканчивающегося предложением терминатора (то есть присоединением, следующим состоянием, стопом), является предложение, которое следует за ключевым словом ENDDECISION; это показано на двух эквивалентных ответвлениях на рис. D-3.8.45.

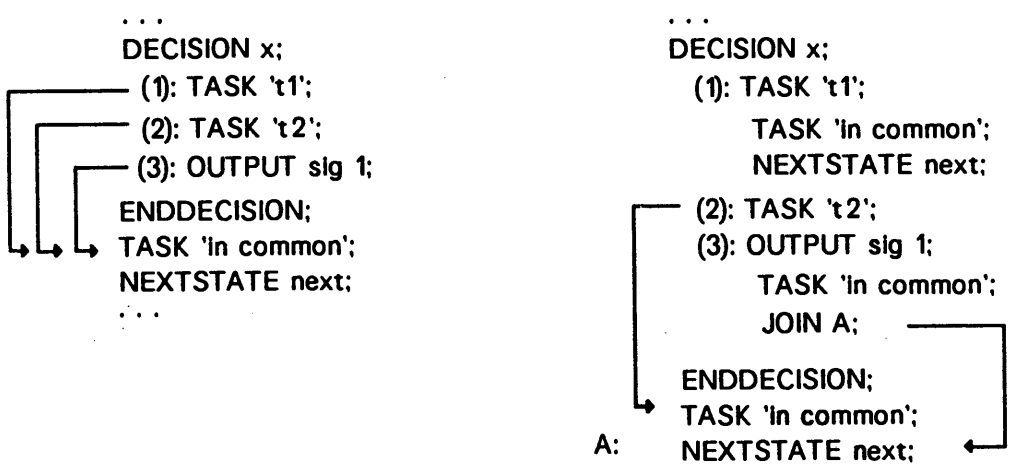


РИСУНОК D-3.8.45

Эквивалентные ответвления после принятия решения

Принятие решения может также использоваться для моделирования структур IF-THEN, DO-WHILE и LOOP-UNTIL, как в структурном программировании.

На SDL/GR принятие решения представляется с помощью символа принятия решения, содержащего текст вопроса. Символ должен иметь две или более ветви, связанные с соответствующими ответами. Каждый ответ должен быть расположен справа от соответствующей ветви или над ней; кроме того, можно помещать ответ над ветвью, прерывающей линию потока. На SDL/GR скобки, выделяющие ответы, не являются обязательными, однако во избежание недоразумений их рекомендуется использовать.

Некоторые примеры принятия решений на SDL/GR приведены на рис. D-3.8.46.

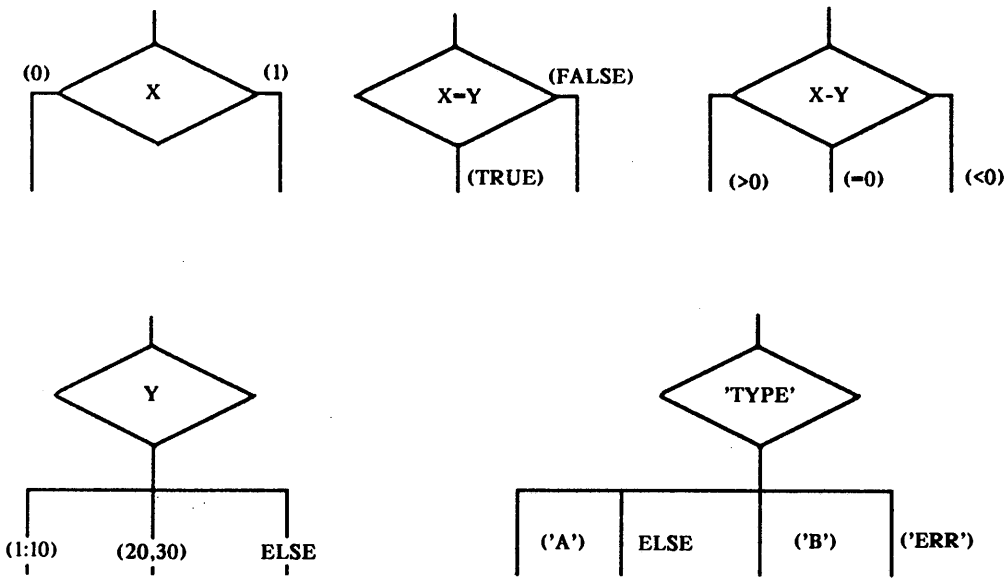


РИСУНОК D-3.8.46

Примеры принятия решения на SDL/GR

Если ответ ведет обратно к принятию решения в том же переходе, то должны быть выполнены какие-то действия, которые влияют на вопрос, стоящий в принятии решения. Однако даже при этом правиле могут возникнуть бесконечные циклы, как это показано на рис. D-3.8.47. Поэтому всегда надо проявлять осторожность, если имеются ответы, ведущие назад к принятию решения в том же переходе.

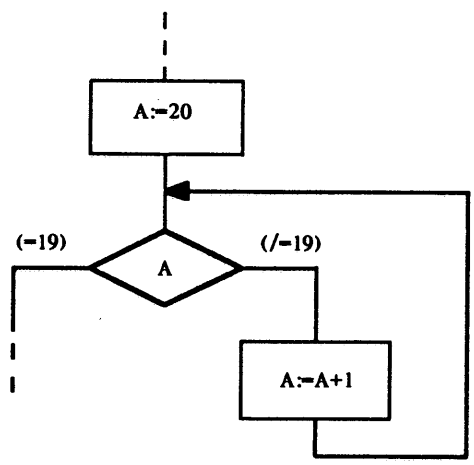


РИСУНОК D-3.8.47

Примеры допустимого принятия решения, образующего бесконечный цикл

В принятии решений можно использовать любые значения, которые в текущий момент доступны процессу, включая:

– значения, полученные в вводе;

– значения, переданные при создании процесса в качестве фактических параметров;

– совместно используемые значения.

Выражение в вопросе может содержать константы и любые из вышеуказанных значений.

D.3.8.9 *Присоединения и коннекторы*

Присоединения позволяют передавать управление из одного места тела процесса в другое (и, в равной мере, внутри тела процедуры или внутри тела сервиса).

На SDL/PR они эквивалентны предложению "GO TO". В качестве входных точек используются метки, связанные с предложениями, как это показано на рис. D-3.8.48. Внутри тела процесса (или тела процедуры) нельзя передавать управление (а следовательно, и связывать метки) на предложения одного из типов, указанных на рис. D-3.8.49. Метки всегда локальны по отношению к процессу, а потому передать управление от одного процесса к другому с помощью присоединения нельзя.

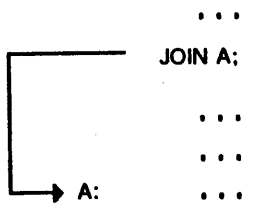


РИСУНОК D-3.8.48

Метка

STATE,
ENDSTATE,
INPUT,
SAVE,
ENDDECISION

РИСУНОК D-3.8.49

Недопустимые места присвоения меток

На SDL/GR присоединениям соответствуют коннекторы (входной и выходной коннекторы). Они могут использоваться для расчленения диаграммы, вызванного недостатком места, а также для избежания пересечений линий потока, которые могли бы привести к неясности. Кроме того, обычно, предпочтительнее рисовать SDL-диаграмму так, чтобы поток в диаграмме шел от верхней части страницы к нижней.

На GR любая линия потока может быть прервана двумя связанными коннекторами; при этом предполагается, что поток идет от выходного коннектора к входному коннектору. Каждый символ коннектора содержит имя; имена связанных коннекторов совпадают. Для каждого имени существует только один входной коннектор, содержащий это имя, но число выходных коннекторов с этим именем может быть один или более.

На GR желательно снабжать входной коннектор ссылкой на страницу соответствующего(их) выходного(ых) коннектора(ов) и, кроме того, снабжать выходные коннекторы ссылками на соответствующий входной коннектор. (См. пример на рис. D-3.8.50.)

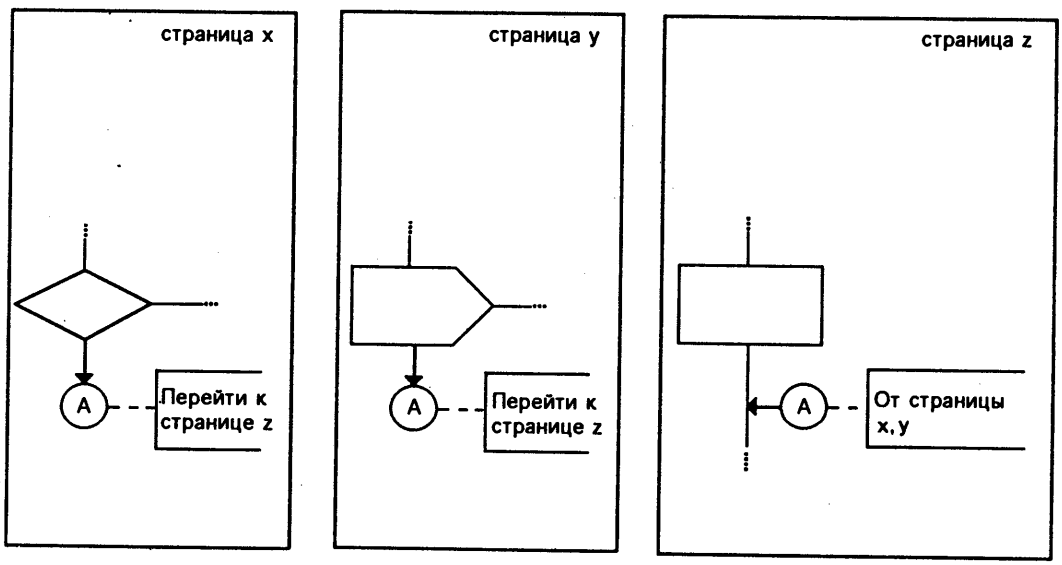


РИСУНОК D-3.8.50

Ссылки на страницы при коннекторах

D.3.9 *Процедуры*

Процедуры SDL подобны процедурам CHILL и других языков программирования. Назначением процедур является:

a) разрешить структурирование процесса на уровни разной степени детализации;

b) поддержать компактность спецификаций, разрешив рассматривать как один элемент сложный комплекс различных элементов, который может быть описан изолированно от остальных;

c) разрешить отдельное описание и повторное использование часто используемых комплексов элементов.

Определение процедуры может находиться только в определении процесса, определении сервиса или определении процедуры, а поэтому процедуры доступны только тому процессу или той процедуре, в которой они определены.

Определение процедуры состоит из следующих элементов (некоторые из которых не обязательны):

– Имени процедуры.

– Формальных параметров процедуры: списка имен переменных со связанными с ними сортами. Они используются для передачи информации к процедуре и от нее в момент вызова процедуры. Параметры процедуры могут быть переданы значениями (параметры IN) или ссылками (параметры IN/OUT). Если параметр передан значением, то спецификация формального параметра определяет переменную, локальную по отношению к процедуре; если он передан именем, то спецификация определяет синоним переменной.

– Определений процедуры: процедуры, которые могут быть вызваны только самой процедурой.

– Определений данных: определения данных, локальных по отношению к процедуре.

– Определений переменных: локальные переменные внутри процедуры.

– Тела процедуры: спецификация фактического поведения процедуры в терминах состояний и действий (подобно телу процесса).

Частичный пример определения процедуры на SDL/PR приведен на рис. D-3.9.1 (ключевые слова языка написаны прописными буквами). Обратите внимание, что формальные параметры, не имеющие явных атрибутов, имеют неявный атрибут IN (var 5 на рисунке).

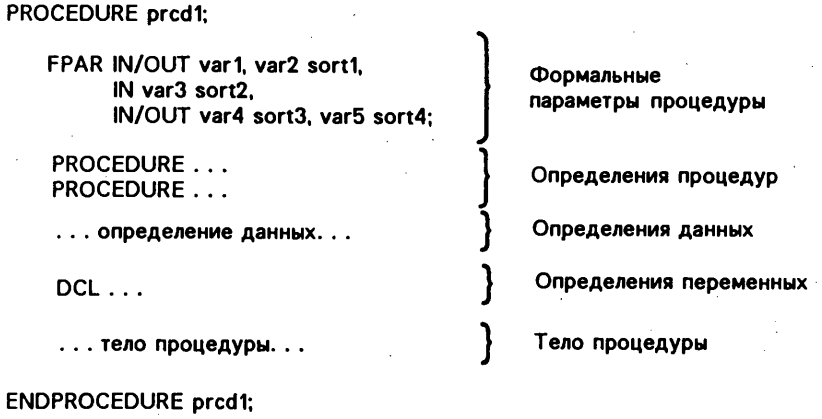


РИСУНОК D-3.9.1

Частичный пример определения процедуры на SDL/PR

D.3.9.1 *Тело процедуры*

Тело процедуры очень близко телу процесса, но со следующими отклонениями:

– Интерпретация процедуры заканчивается "возвратом", а не "стопом". На SDL/PR предложение возврата представляется ключевым словом RETURN.

– На SDL/GR символ начала процедуры несколько отличается от символа начала процесса.

Символы начала и возврата процедуры могут быть найдены в свободном описании SDL/GR.

Процедура может использовать конструкцию присоединения, но только для ссылок на метки внутри себя самой. Присоединение не может быть использовано ни для входа в процедуру, ни для выхода из нее.

На SDL/GR определение процедуры изображается с помощью диаграммы, весьма близкой к диаграмме процесса. Диаграмма процедуры состоит из следующих элементов:

– Необязательного символа кадра: прямоугольный по форме символ, содержащий все прочие символы.

– Заголовка процедуры: ключевое слово PROCEDURE, за которым следует имя процедуры и спецификация формальных параметров процедуры. Обычно заголовок процедуры помещают в верхний левый угол кадра или, если символ кадра не используется, то в левый верхний угол того носителя, на котором изображается диаграмма.

– Необязательной нумерации страниц. (Помещенной в правый верхний угол.)

– Символ текста: для случая диаграммы процедуры символы текста могут использоваться для помещения в них спецификаций формальных параметров, данных и определений переменных.

– Ссылок на процедуры: символ процедуры, каждая из которых содержит имя процедуры, представляющее локальную, отдельно определенную процедуру.

– Диаграмм процедур: для непосредственного определения локальных процедур.

– Области графа процедуры: спецификация поведения процедуры в терминах старта, состояний, вводов, выводов, работ, ... и ориентированных дуг.

На рис. D-3.9.2 приведен пример определения процедуры на SDL/GR. Процедура "TERM_P", на которую имеется ссылка, является локальной по отношению к вызывающей процедуре.

Как было отмечено для случая диаграмм процессов (см. § D.3.8), если диаграмма процедуры не помещается на одной странице, то диаграмма может быть изображена на нескольких страницах, причем каждая содержит символ кадра с заголовком процедуры и нумерацией страниц.

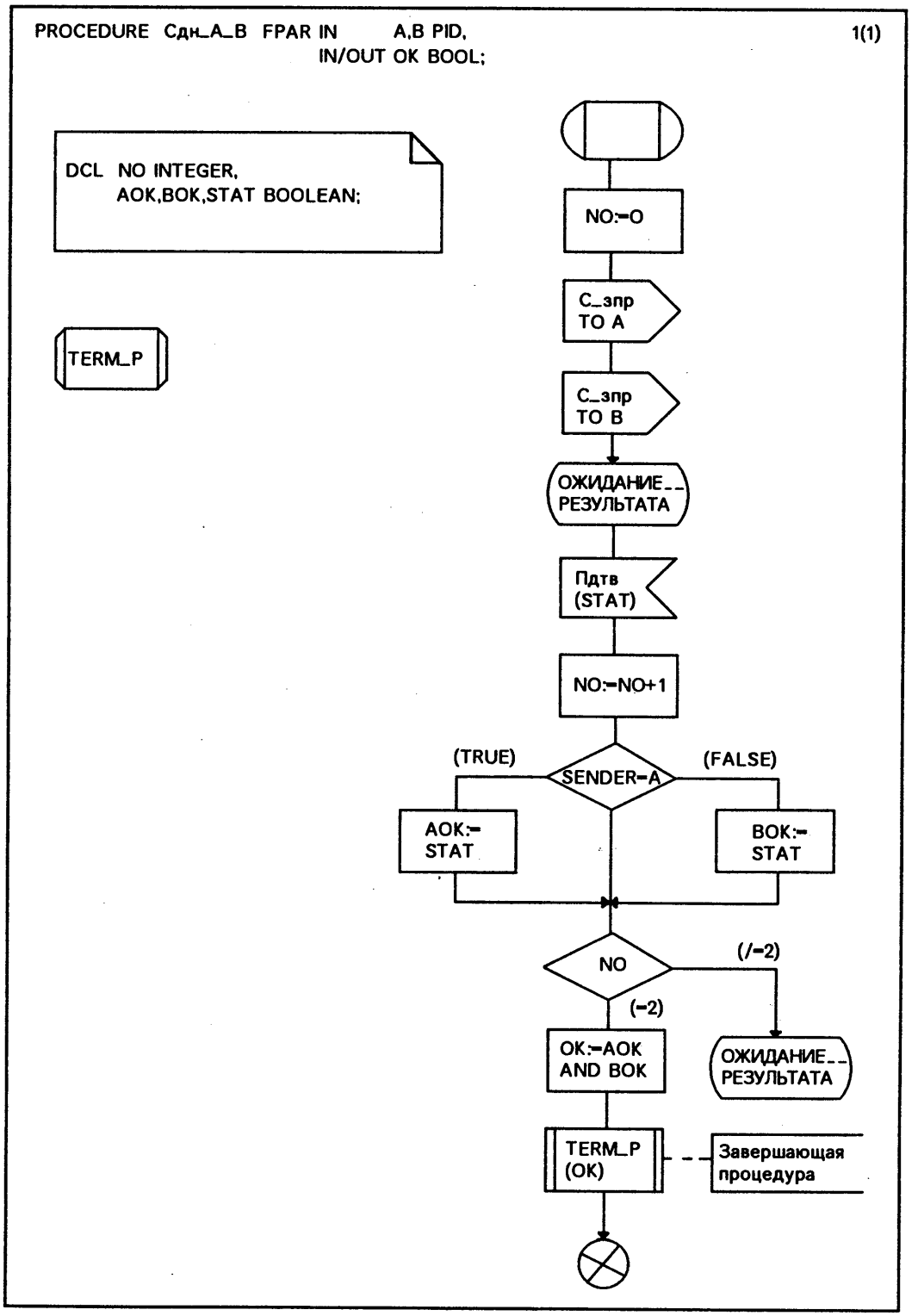


РИСУНОК D-3.9.2

Пример диаграммы процедуры

D.3.9.2 *Вызов процедуры*

Вызов процедуры может находиться всюду, где в графе процесса или процедуры может находиться работа. В каком-то смысле на вызов процедуры можно смотреть как на работу, но со следующими исключениями:

1) Процедура может содержать состояния и, если это имеет место, получать сигналы.

2) Процедура может посылать сигналы. Процесс, посылающий сигналы, является тем экземпляром, который осуществил вызов процедуры.

После вызова процедуры создается окружающая среда процедуры и начинается ее интерпретация. Интерпретация продолжается до тех пор, пока не будет достигнут RETURN. Пока идет интерпретация процедуры, все сигналы, адресованные процессу, либо неявно сохраняются, либо явно обрабатываются процедурой. У процедуры нет своей собственной входной очереди и она пользуется входной очередью того процесса, который вызвал ее.

На SDL/PR вызов процедуры представляется с помощью ключевого слова CALL, за которым следует идентификатор процедуры и взятый в круглые скобки список фактических параметров. Если некоторый параметр не задан, то его место должно быть указано двумя последовательными запятыми. В этом случае соответствующий формальный параметр получает "неопределенное" значение. Обратите также внимание на то, что объявление IN или IN/OUT делается в определении процедуры и, следовательно, его не нужно повторять в предложении вызова. Несколько примеров вызова процедуры на SDL/PR приведены на рис. D-3.9.3.

```
CALL proced1;
CALL proced2(var1,var2);
CALL proced3 (5,a+b,Pid1);
CALL proced4 (W,X,,Z);
```

РИСУНОК D-3.9.3

Примеры предложений вызова на SDL/PR

На SDL/GR вызов процедуры изображается с помощью символа вызова процедуры, содержащего имя процедуры и заключенный в круглые скобки список фактических параметров. Пример вызова процедуры на SDL/GR приведен на рис. D-3.9.2.

D.3.10 *Обработка данных*

D.3.10.1 *Объявления переменных*

Переменные являются локальными по отношению к экземплярам процессов, из чего следует, что для каждой переменной существует один, и только один экземпляр процесса, который владеет ею. Только экземпляр процесса, владеющий переменной, может изменить ее значение.

Переменные, объявленные на рис. D-3.10.1, являются локальными по отношению к каждому экземпляру процесса P и поэтому доступ к ним и изменение их значения возможны только данным экземпляром процесса P (каждый экземпляр процесса имеет доступ и право модификации только тех копий переменных, которыми он владеет).

```
SYSTEM S;
   . . .
   BLOCK B;
   . . .
      PROCESS P(3,10);
         DCL
         A Integer,
         D Integer;
         . . .
         . . .
      ENDPROCESS P;
   ENDBLOCK B;
   . . .
ENDSYSTEM S;
```

РИСУНОК D-3.10.1

Пример объявления переменных

Переменные могут быть инициализированы непосредственно после объявления, как это показано на рис. D-3.10.2.

```
DCL A Integer :=1;
```

РИСУНОК D-3.10.2

Инициализация переменной

SDL допускает два способа инициализации переменных. Во-первых, есть возможность объявить начальное значение всех переменных некоторого сорта с помощью предложения DEFAULT в определении типа данных (см. § D.6.4.5). В этом случае инициализация применяется ко всем переменным данного сорта.

Во-вторых, имеется возможность инициализировать каждую переменную некоторого сорта к конкретному значению, как это показано на рис. D-3.10.2. Если имеется как предложение DEFAULT, так и инициализация в объявлении, то последнее имеет бóльшую силу. Если переменная не инициализируется, то ее значение в системе считается "неопределенным".

Конечно, сокращенная нотация инициализации переменных, приведенная на рис. D-3.10.2, применима только для простых переменных или для типов данных, допускающих компактную конкретную нотацию переменных (другой пример приведен на рис. D-3.10.3).

```
PROCESS P1;
    NEWTYPE S Struct
        I Integer;
        B Boolean;
    ENDNEWTYPE;
    DCL
    A S :=(. 1,True .);
    . . .
ENDPROCESS P1;
```

РИСУНОК D-3.10.3

Другой пример инициализации переменной

Рис. D-3.10.3 иллюстрирует, что Структурный сорт имеет сокращенную конкретную нотацию для задания значения структуры. Для случая генератора массива рекомендуется явно инициализировать переменные, моделируя "конструкцию while" в цепочке начального перехода процесса (см. рис. D-3.10.4).

```
PROCESS P;
        NEWTYPE Arr1 Array (Nat1,Integer)
        ENDNEWTYPE Arr1;
        SYNTYPE Nat1 Natural CONSTANTS 1 : 3
        ENDSYNTYPE Nat1;
        DCL
          A Arr1,
          I Natural;
        START;
        TASK I:=1;
   Lab1: DECISION I<=3;
              (True): TASK A(I):=I;
                      TASK I:=I+1;
                      JOIN Lab1;
           (False):;
        ENDDECISION;
        ...
ENDPROCESS P;
```

РИСУНОК D-3.10.4

Более сложный пример инициализации переменной

D.3.10.2 *Раскрываемые/обозреваемые переменные*

Два процесса могут обмениваться информацией не только с помощью сигналов, но и с помощью других средств. Один процесс может обратиться к значению переменной, которой владеет другой процесс, с помощью операции VIEW (обозревание). Следует, однако, отметить несколько имеющихся правил:

– Оба процесса должны принадлежать одному и тому же блоку.

– Процесс, выполняющий операцию VIEW, должен специфицировать идентификатор обозреваемой переменной в определении обозревания.

– Процесс, раскрывающий переменную, должен объявить это с помощью атрибута REVEALED (раскрытие).

– Идентификаторы сорта (или идентификатор синонимичного типа) в объявлении переменной и в определении обозревания должны совпадать.

Значение, получаемое обозреваемым процессом посредством операции VIEW, совпадает со значением, которое получает раскрывающий процесс с помощью обычного доступа.

Так как обозревающий процесс не владеет обозреваемой переменной, то он не может изменить ее значения. Конечно, обозреваемое значение может быть присвоено переменной, которой владеет сам обозревающий процесс.

Пользователь SDL может обнаружить, что определение раскрываемой переменной обеспечивает легкий метод спецификации связи между двумя процессами. Однако при реализации специфицированных таким способом систем возникает ряд трудностей и в настоящем разделе предполагается научить пользователей, как избежать и преодолеть эти трудности. Описание на SDL уже существующих систем, которые реализовали раскрытие/обозревание значений, связано с меньшими трудностями, так как все проблемы были преодолены при реализации, что обеспечивает возможность отображения принятых решений на SDL.

В остальной части настоящего раздела предполагается, что процесс R (Revealer – "раскрыватель") владеет переменными и раскрывает их, а процесс V (Viewer – "обозреватель") ссылается на них в своем определении обозревания.

Попытка процесса обозревать переменные в процессе V до того, как создан процесс R, приводит к SDL-ошибке. Пользователь может избежать этой трудности одним из двух следующих способов:

– обеспечить создание экземпляра раскрывающего процесса R и инициализацию им соответствующих переменных до создания экземпляра обозревающего процесса V; или

– обеспечить процессу V невозможность выполнения перехода, использующего раскрываемые/обозреваемые переменные, до того как был создан и инициализировал соответствующие переменные процесс R.

Простой способ, осуществляющий первый подход, заключается в том, чтобы сделать R родителем (или предком) процесса V (как это сделано в примере на рис. D-3.10.5) или обеспечить создание R в момент создания системы (неявное создание). Для реализации второго метода достаточно сделать так, чтобы соответствующий переход в процессе V мог выполняться только по сигналу от процесса R.

Переменные процесса R не могут быть обозреваемы после того, как процесс R остановился. Любая попытка обозревать переменные будет SDL-ошибкой. Пользователь может избежать этой трудности одним из двух следующих способов:

– совсем не пользоваться стопом в R; или

– обеспечить осведомление процесса V о том, что процесс R собирается остановиться, с тем чтобы V более не делал попыток обозревать данные в процессе R.

С точки зрения реализации первое решение имеет тот недостаток, что при нем процесс R не освобождает ту память под данные, которую он использовал.

Пример раскрываемых/обозреваемых переменных приведен на рис. D-3.10.5.

D.3.10.3 *Экспортируемые/Импортируемые значения*

Процесс может объявить одну или более своих переменных "экспортируемыми", в результате чего все остальные процессы (независимо от того, какому блоку они принадлежат) могут по запросу импортировать копии значений таких переменных. Импортирующий процесс должен объявить переменные в своем определении импорта.

Когда экспортирующий процесс выполняет экспортную операцию, со значения переменной снимается копия и помещается в неявную переменную. Импортирующий процесс с помощью импортирующего выражения получает значение этой копии. Следовательно, значение, полученное с помощью импортирующего выражения, может отличаться от значения, получаемого владельцем переменной с помощью обычного доступа, даже если эти две операции выполняются одновременно.

Пример этого приведен на рис. D-3.10.5.

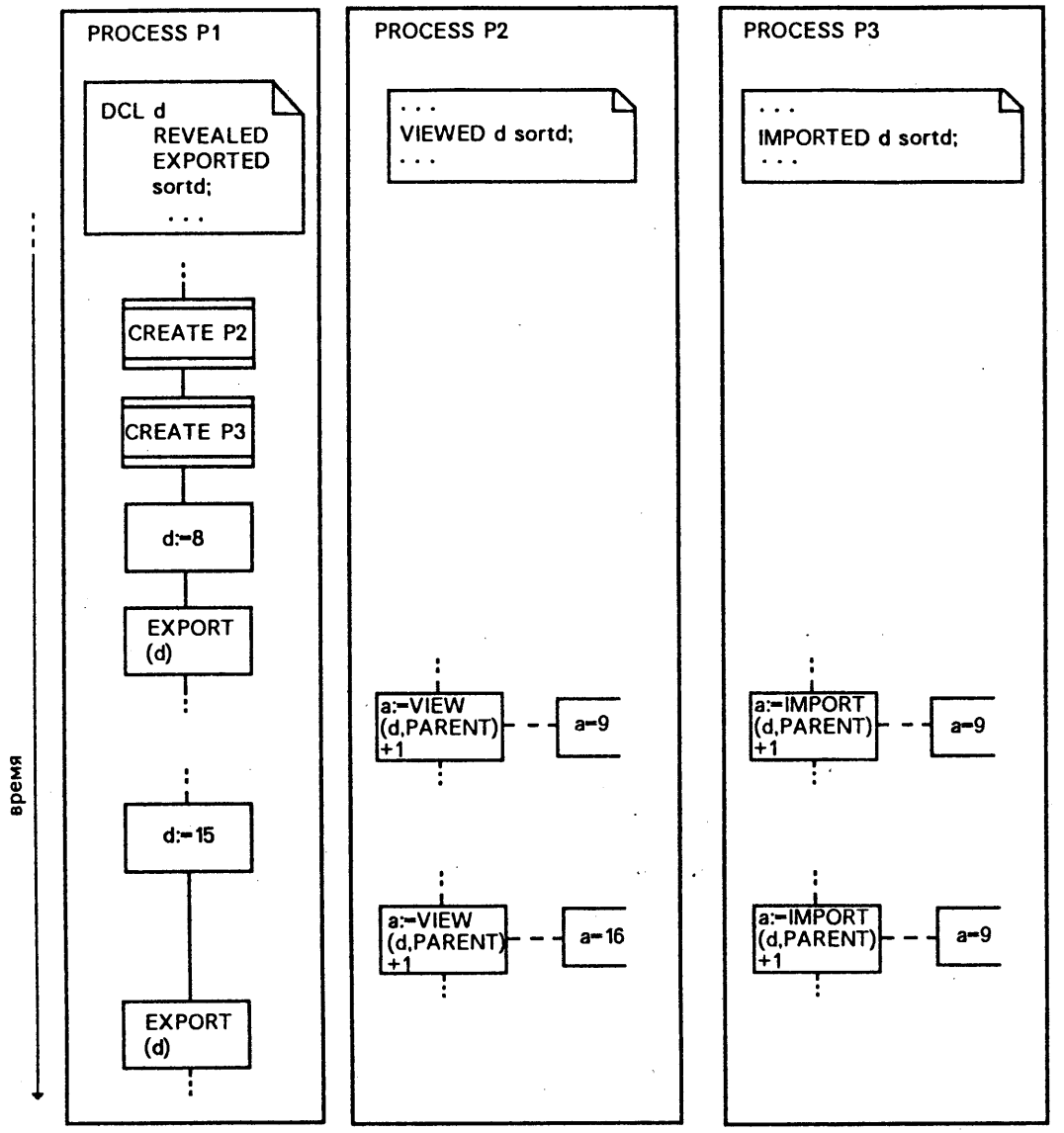


РИСУНОК D-3.10.5

Пример различий для использования раскрываемых/ обозреваемых и экспортируемых переменных

На рис. D-3.10.5 предполагается, что процесс P1 является родителем процессов P2 и P3; поэтому идентификатор экземпляра процесса в выражениях IMPORT и VIEW обозначен атрибутом PARENT.

D.3.10.4 *Выражения*

В SDL-процессе выражения могут использоваться в качестве формального текста в следующих конструкциях: принятие решения, альтернатива, избрание, работа, непрерывный сигнал, разрешающее условие и установка. Кроме того, выражения используются в качестве фактических параметров в выводе, вызове процедуры и конструкции запроса на создание. Выражения сорта PId используются в той части конструкции вывода, которая начинается с TO. Выражения в конструкциях альтернативы и избирания (вычисляемых статически) должны принадлежать одному из предварительно определенных сортов. Выражения могут содержать пассивные термы (то есть термы, содержащие представления только постоянных значений), а также термы, содержащие переменные.

В SDL имеется набор предварительно определенных инфиксных операторов. Эти операторы могут применяться к данным любых типов; они являются единственно допустимыми инфиксными операторами. Для этих операторов правила прецедентности также являются предварительно определенными и не могут быть изменены. Предварительно определенными инфиксными операторами являются:

=>, OR, XOR, AND, IN, /=, =, >, <, <=, >=, +, –, //, *, /, MOD, REM

Кроме того, SDL обеспечивает следующих предварительно определенных операторов:

–, NOT

являющиеся унарными префиксными операторами.

При использовании предварительно определенных операторов надо быть осторожным в отношении правил прецедентности, так как эти операторы, будучи предварительно определенными, независимо от области применения, могут привести к ошибке.

Например, рассмотрим новый тип и выражение, приведенные на рис. D-3.10.6. Учитывая правила прецедентности для операций умножения и сложения, приведенное выражение может быть вычислено как A=> ((B*C) + D). Но для AND и OR правила прецедентности будут уже другими, и такая замена может привести к ошибке. Более того, изменения такого типа могут привести к несогласованности аксиом и сделать недействительной SDL-спецификацию.

```
NEWTYPE Newbool
    INHERITS Boolean
    OPERATORS ("+"="AND", "*"="OR", "=>")
ENDNEWTYPE Newbool;
...
A=>B*C+D
```

РИСУНОК D-3.10.6

Пример нотации в выражении, приводящей к ошибке

Все остальные операторы, определенные пользователем, являются функциями и должны использоваться в префиксной нотации.

Два оператора: VIEW и IMPORT – имеют специальную семантику, которая была объяснена в предшествующих параграфах. Как бы то ни было, они возвращают значение некоторого конкретного сорта, которое в свою очередь может быть операндом выражения.

D.3.11 *Представление в SDL времени*

Необходимость для систем иметь возможность измерять время и запрашивать таймауты обеспечивается с помощью таймеров и набором операций, выполняемых над ними.

В SDL модель "Таймер" является мета-процессом, способным в соответствии с запросами посылать сигналы процессу. Использование таймера должно быть объявлено в определении таймера в пределах определения процесса. Для активизации таймеров используются операции "SET" и "RESET". Операция SET (установка) вызывает появление таймаута в определенный момент времени, а операция RESET (сброс) аннулирует специфицированный таймаут. (Заметьте, что операция SET неявным образом включает в себя операцию RESET по отношению к любому еще не истекшему таймауту от этого таймера.)

Конструкция установки содержит выражение времени запрашиваемой длительности, имя используемого таймера и необязательный список выражений. Этот список выражений специфицирует значения, которые будут содержаться в сигнале от таймера, причем в том же порядке.

Список выражений может быть специфицирован в конструкции сброса для сбрасывания каких-то конкретных экземпляров таймеров, имеющих те же значения.

Определение таймера должно содержать список идентификаторов сортов, соответствующих тем же сортам, которые используются в выражениях, входящих в установку/сброс.

Пример предложения Установка приведен на рис. D-3.11.1.

В конструкции установки должно быть специфицировано абсолютное время. Относительное время преобразуется в абсолютное время прибавлением примитивной функции "NOW", представляющей текущее время. Запрашиваемая задержка должна быть выражена с помощью выражения Время. Время является предварительно определенным сортом, унаследованным от сорта Вещественный.

Возможность получения таймаута специфицируется заданием во вводе имени таймера, как это показано на рис. D-3.11.1.

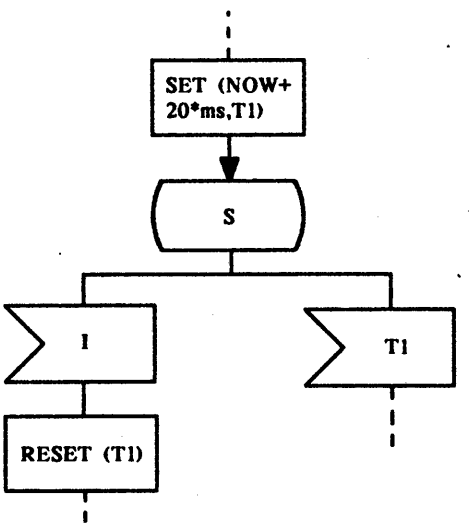


РИСУНОК D-3.11.1

Пример использования таймеров

Для индикации требующейся Длительности допускается определение Синонимы. Раз выбрав единицы измерения для Время и Длительности, пользователь может определить Синонимы, требующиеся для представления Длительности, как это показано на рис. D-3.11.2.

```
PROCESS p;
    TIMER T1 идент_абонента;
    . . .
    DCL Sa идент_абонента;
    . . .
    SYNONYM
        Sec Duration = 1000.0,
        Min Duration = 60000.0,
    . . .
    START;
        . . .
        SET (NOW + 20*Min + 30*Sec, T1 (Sa) );
        . . .
        RESET (T1 (Sa) );
        . . .
ENDPROCESS p;
```

РИСУНОК D-3.11.2

Синонимы для определения длительностей

В Рекомендации указывается, что допускается установка в таймере уже "прошедшего" времени. Это решение было принято для того, чтобы облегчить моделирование систем. Так как, однако, такая установка может привести к неясной спецификации, ее использования следует избегать.

D.3.12 *Использование квалификаторов*

В SDL квалификаторы используются для ссылок в спецификации на элементы в тех случаях, когда имена не определяют эти элементы однозначно. Конечно, при определении элемента должно специфицироваться только его имя, но при ссылке на элемент извне для вхождения его определения может потребоваться уже идентификатор, составленный из квалификатора и имени.

Это имеет место и в случае, когда используются удаленные определения: вхождение определения при ссылках на определение использует имя; удаленное определение использует квалифицированное имя для спецификации своего контекста.

На рис. D-3.12.1 приведен пример использования квалификаторов в процессе, который может получить два различных сигнала, имеющих одинаковые имена, но различные идентификаторы! В первом вводе делается ссылка на тип сигнала, определенного на уровне блока; во втором вводе делается ссылка на тип сигнала, определенного внутри определения процесса. Во втором вводе квалификация может быть опущена, так как для случая, когда квалификация опущена, используется само внутреннее определение.

```
SYSTEM s;
   . . .
   BLOCK b;
      SIGNAL x;
      . . .
      PROCESS p;
         . . .
         SIGNAL x;
         . . .
         STATE ожидание;
            INPUT  SYSTEM s/BLOCK b  x;
               . . .
            INPUT  PROCESS p  x;
               . . .
         ENDSTATE ожидание;
         . . .
      ENDPROCESS p;
   ENDBLOCK b;
ENDSYSTEM s;
```

РИСУНОК D-3.12.1

Пример использования квалификаторов

D.3.13 *Синтаксис имен*

В SDL имена могут состоять либо из одного слова, либо из списка слов, отделенных друг от друга разделителями (пробелами или управляющими знаками). Вторая спецификация способствует более легкому восприятию спецификации, когда используются длинные имена; это особенно важно в SDL/GR, так как графические символы имеют ограниченные размеры. Несколько примеров имен, состоящих из нескольких слов, приведены на рис. D-3.13.1.

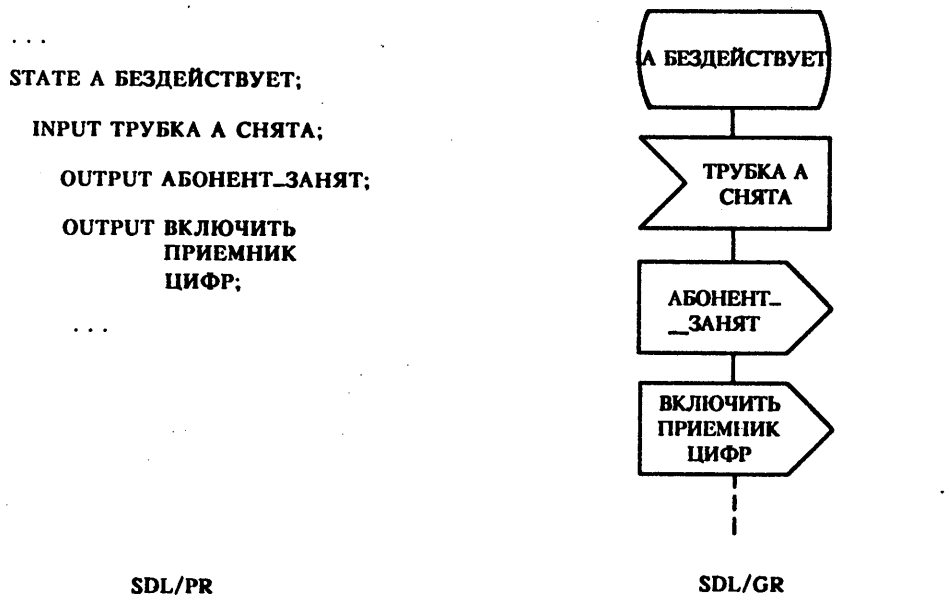


РИСУНОК D-3.13.1

Примеры имен, содержащих несколько слов

Если использование нескольких слов в имени приводит к неоднозначности (см. примеры на рис. D-3.13.2), то разделители между словами должны быть заменены знаком подчеркивания ("_"). Цепочка знаков, полученная заменой разделителей на знаки подчеркивания, обозначает то же имя; так, например, BUSY SUB обозначает то же имя, что и BUSY_SUB. На рис. D-3.13.3 приведено несколько примеров однозначного использования имен.

a) A := BLOCK B1 B / PROCESS P1 P C;

b) DCL A B C D;

c) NEWTYPE X Y Z (PAR) ENDNEWTYPE;

РИСУНОК D-3.13.2

Примеры неоднозначного использования имен

a) A := BLOCK B1_B / PROCESS P1_P C;

b) A := (BLOCK B1 B) / (PROCESS P1_P C);

c) DCL A B_C_D;

d) DCL A_B C_D;

e) DCL A_B_C D;

f) NEWTYPE X Y_Z(PAR) ENDNEWTYPE;

g) NEWTYPE X_Y Z(PAR) ENDNEWTYPE;

РИСУНОК D-3.13.3

Примеры однозначного использования имен

Важно отметить, что знак подчеркивания может использоваться в именах в качестве знака продолжения, что позволяет разбивать имена по нескольким строкам. В этом случае в имени, содержащем знак подчеркивания, за которым следует один или несколько разделителей, знаки подчеркивания и разделители могут быть опущены.

Примеры различных начертаний одного и того же имени приведены на рис. D-3.13.4.

a) CONNECT DIGIT RECEIVER

b) CONNECT_DIGIT_RECEIVER

c) CONNECT DIGIT_RECEIVER

d) CONNECT
DIGIT
RECEIVER

e) CONNECT DI_
GIT RECEIVER

f) CONNECT DI_ GIT RECEIVER

g) CONNECT_
DIGIT
_RECEIVER

РИСУНОК D-3.13.4

Примеры различного написания одного и того же имени

D.4 *Структурирование и уточнение SDL-систем*

D.4.1 *Общие положения*

В настоящей главе обсуждаются некоторые приемы и SDL-конструкции, которые обеспечивают возможность спецификации больших систем методом "сверху вниз". Обычно термины "разбиение" и "уточнение" используются применительно к этому методу в следующем смысле:

– Разбиение: разложение части системы на более мелкие подчасти, чье совокупное поведение эквивалентно всей части. Оно может быть применено к блокам (структурируя их на новые подблоки, каналы и подканалы), к каналам (структурируя их на блоки, новые каналы и подканалы) и к процессам (структурируя их на сервисы).

– Уточнение: добавление новых деталей к функциональным свойствам системы. При взгляде на систему из окружающей среды уточнение системы приводит к обогащению ее поведения, так как становится возможной обработка новых типов сигналов и информации.

Обратите внимание на то, что внутренняя структура части системы поставляет новые детали относительно структуры, но не обязательно детали относительно поведения системы. Концептуально можно разграничить аспект более детализированного представления поведения системы (например, обработка нового сигнала) и аспект более детализированной структуры системы, но обычно на практике эти два аспекта сливаются и в итоге вместе с новыми деталями о структуре системы мы предоставляем также новые детали о ее поведении.

Минимальной структурой SDL-системы является то, что описано в главе 2 Рекомендации, то есть что система состоит из блоков, соединенных каналами, а блоки содержат процессы.

Концепции разбиения на несколько уровней детализированного уточнения поведения системы и сигналов рассматриваются в главе 3 Рекомендации. Для систем, не требующих дальнейшего разбиения, такие концепции не являются необходимыми.

D.4.2 *Критерии разбиения*

Метод, при котором сначала приводится высокий уровень описания системы, а потом уже это описание расчленяется на легко обозримые части, называется разбиением. Этот процесс разбиения развивает структуру системы.

Существует несколько критериев разбиения общего представления системы, в том числе:

a) определение блоков или процессов умственно обозримых размеров;

b) выработка соответствия с фактической программной и/или аппаратной структурой;

c) следование естественному функциональному подразделению;

d) минимизация взаимодействий между блоками;

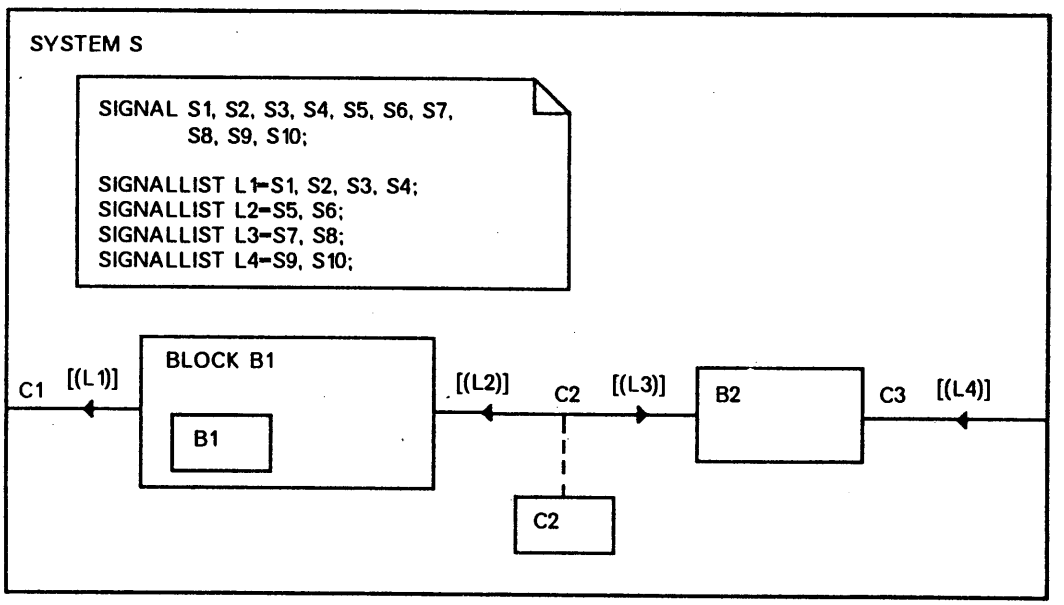
e) повторное использование уже имеющихся спецификаций (например, систем сигнализации).

Фактически принятый критерий может зависеть от целого ряда факторов, включая требуемую степень детализации.

Так как соотношения между уровнями детализации существенно зависят от выбранного критерия разбиения, то очень важно четко сформулировать этот критерий; это позволит облегчить понимание представления. Критерий разбиения зависит от пользователя, но существуют некоторые ограничения, призванные обеспечить корректность представления на SDL. Их рассмотрение приводится в последующих параграфах.

D.4.3 *Разбиение блоков*

Блок может быть разбит на блоки и каналы почти так же, как система разбивается на блоки и каналы. На SDL/GR это представляется с помощью диаграммы подструктуры блока. Пример диаграммы подструктуры блока приведен на рис. D-4.3.1. В первой из диаграмм этого рисунка имеется диаграмма блока B1, содержащая ссылку на его подструктуру. Вторая диаграмма является диаграммой подструктуры блока B1. Символ блока, присоединенный в первой диаграмме к каналу C2 пунктирной линией, изображает ссылку на подструктуру канала C2 (см. § D.4.5).



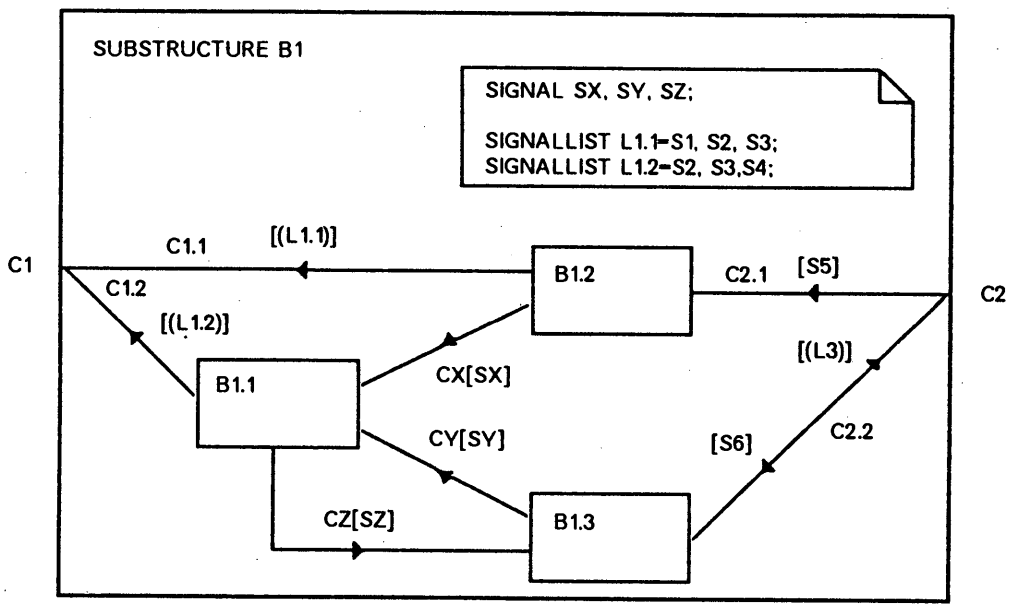


РИСУНОК D-4.3.1

Разбиение блока

На SDL/PR разбиение блока изображается набором определений, заключенным между ключевыми словами SUBSTRUCTURE и ENDSUBSTRUCTURE; это описание помещается внутри определения блока. Определения, входящие в подструктуру блока, совпадают с таковыми в определении системы; кроме того, соединения между каналами и подканалами должны быть представлены так, как это изображено на рис. D-4.3.2.

```
SUBSTRUCTURE B1;

    SIGNALLIST L1.1 =S1,S2,S3;
    SIGNALLIST L1.2 =S2,S3,S4;

    SIGNAL SX,SY,SZ;

    CHANNEL C1.1
        FROM B1.2 TO ENV WITH (L1.1);
    ENDCHANNEL C1.1;
    CHANNEL C1.2
        FROM B1.1 TO ENV WITH (L1.2);
    ENDCHANNEL C1.2;
    CHANNEL C2.1
        FROM ENV TO B1.2 WITH S5;
    ENDCHANNEL C2.1;
    CHANNEL C2.2
        FROM ENV TO B1.3 WITH S6;
        FROM B1.3 TO ENV WITH (L3);
    ENDCHANNEL C2.2
    CHANNEL CX
        FROM B1.2 TO B1.1 WITH  SX;
    ENDCHANNEL CX;
    CHANNEL CY
        FROM B1.3 TO B1.1 WITH SY;
    ENDCHANNEL CY;
    CHANNEL CZ
        FROM B1.1 TO B1.3 WITH SZ;
    ENDCHANNEL CZ;

    CONNECT C1 AND C1.1,C1.2;
    CONNECT C2 AND C2.1,C2.2;

    BLOCK B1.1 REFERENCED;
    BLOCK B1.2 REFERENCED;
    BLOCK B1.3 REFERENCED;

ENDSUBSTRUCTURE B1;
```

РИСУНОК D-4.3.2

Пример из рисунка D-4.3.1 на SDL/PR

Определение блока может содержать как спецификацию процесса, так и спецификацию подструктуры блока. В этом случае процесс, находящийся в блоке, изображает поведение блока с некоторой степенью детализации, а другие процессы, находящиеся в определениях подблоков, будут изображать то же поведение, но более детально. Пример блока, описанного как в терминах процессов, так и подструктур, приведен в § D.4.7 (см. рис. D-4.7.1).

Если блок не содержит процессов, а только подструктуру, и если подструктура блока представлена не ссылкой, то для упрощения диаграмм в SDL/GR имеется для этого случая сокращенная нотация. Эта нотация допускает вложенное представление блоков за счет рассмотрения кадра внешнего блока, как неявного кадра подструктуры. С помощью такой нотации пример из рис. D-4.3.1 может быть изображен как на рис. D-4.3.3.

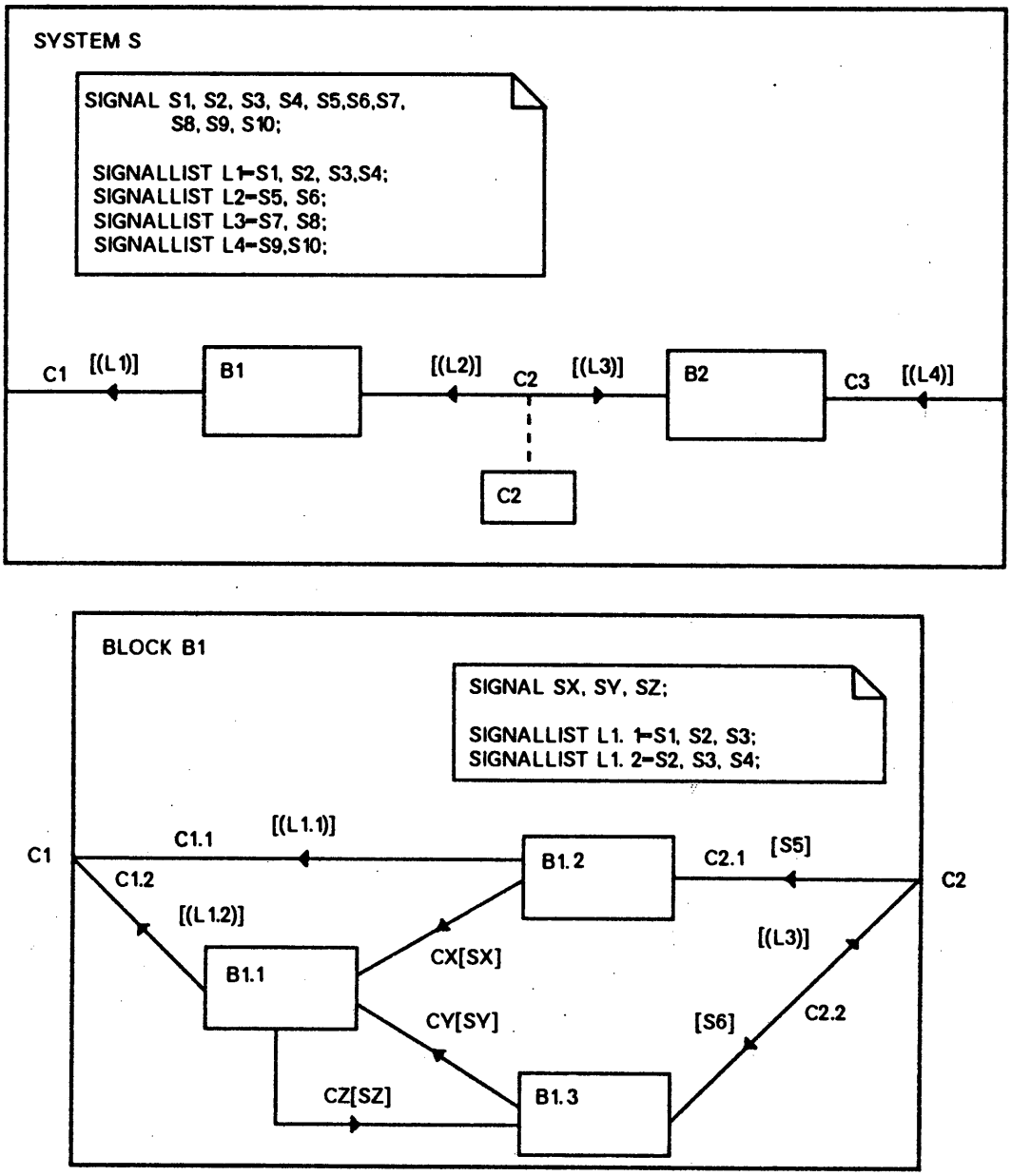


РИСУНОК D-4.3.3

Сокращенная нотация для изображения на SDL/GR разбиения блока

Каждый блок, полученный в результате разбиения некоего блока, может быть в свою очередь подвергнут разбиению, что приведет к иерархической древовидной структуре блоков и их подблоков. Вспомогательная диаграмма, называемая "диаграммой дерева блоков" и изображающая подобную общую структуру, разъясняется в § D.4.4.

Если уточнение сигнала не используется, то разбиение должно подчиняться следующим правилам:

1) Подканалы, присоединенные к входящему каналу, не должны содержать новых сигналов в своих списках сигналов, и, кроме того, эти списки сигналов должны содержать все сигналы, входившие в список сигналов исходного канала. (Для двустороннего канала должен быть рассмотрен список сигналов входного направления.) Так, например, по входным направлениям каналов C2.1 и C2.2 на рис. D-4.3.1 транспортируются все сигналы, входящие в список L2. Более того, ни один сигнал, передаваемый по C2.1, не может появиться во входном направлении канала C2.2.

2) Подканалы (например, C1.1 и C1.2), присоединенные к выходному каналу (например, C1), не должны содержать в своих списках сигналов имена новых сигналов, и их списки сигналов должны содержать все имена сигналов исходного канала. Так, например, списки L1.1 и L1.2 содержат имена всех сигналов, входящие в список L1. Списки L1.1 и L1.2 могут содержать одни и те же идентификаторы сигналов.

3) Если исходный блок содержит процессы, то возможны два варианта. Во-первых, копии каждого процесса могут быть определены в том или другом новом подблоке. Во-вторых, в подблоках могут быть определены новые процессы, но так, чтобы интерфейс остался неизменным.

4) Определения данных в родительском блоке доступны в его подблоках и, следовательно, каждый из них может пользоваться данными, определенными в родительском блоке, без того, чтобы вновь определять их у себя.

5) Если тип данных, определенный в родительском блоке, вновь определен под тем же именем в подблоке, то новое определение относится только к подблоку, в котором оно произведено, в то время как старое определение имеет силу в остальных подблоках. Следует отвергать переопределение, выполняемое исключительно для того, чтобы охарактеризовать подблок, так как читатель может проглядеть это переопределение и будет считать, что все еще действительно старое определение. Однако в некоторых случаях такое переопределение должно выполняться, в частности, тогда, когда бывает вовлечено уточнение поведения. Нужно соблюдать осторожность и сопровождать такое переопределение соответствующей аннотацией.

D.4.4 *Диаграмма дерева блоков*

Диаграмма дерева блоков изображает структуру системы в терминах блоков и подблоков. Цель, преследуемая диаграммой, заключается в предоставлении читателю возможности обозреть общую структуру системы. Диаграмма является иерархическим деревом символов блока, расчлененным по уровням, как это изображено на рис. D-4.4.1.

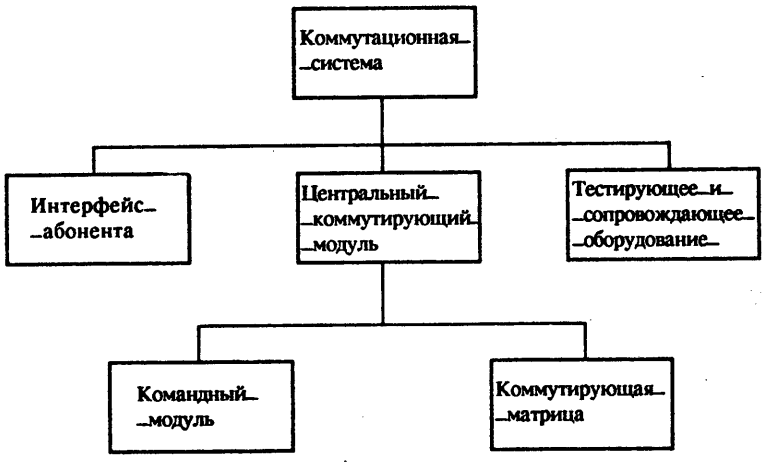


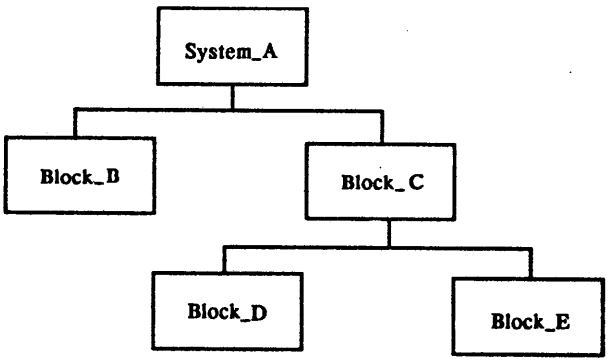
РИСУНОК D-4.4.1

Пример диаграммы дерева блоков

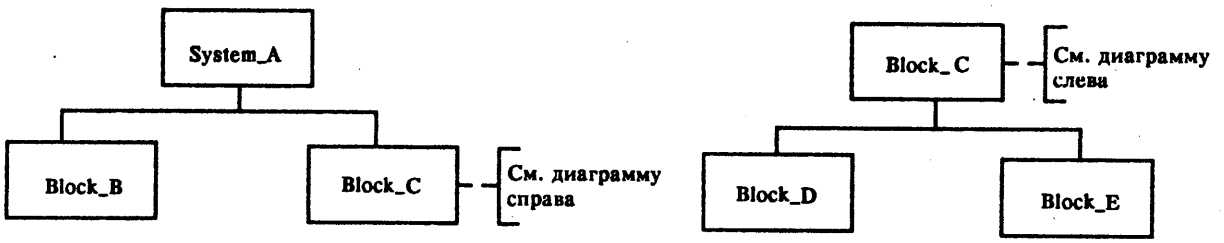
Желательно размеры всех блоков в диаграмме сохранять равными друг другу. Это позволит изображать блоки, находящиеся на одном уровне разбиения, в виде единообразного уровня диаграммы. Если диаграмма столь велика, что ее изображение требует более одной страницы, то ее следует разбивать на "частичные" диаграммы дерева блоков, как это изображено на рис. D-4.4.2.

Очень часто такое разбиение диаграммы дерева блоков на частичные диаграммы оказывается весьма полезным.

Для разбиения на несколько частичных диаграмм первая диаграмма, для которой система является корнем, обрубается так, чтобы все остальные блоки, подвергающиеся дальнейшему разбиению, выглядели бы как не расчлененные. Блоки, под которыми обрубалась исходная диаграмма, выступают в качестве корней диаграммы, показывающих ход дальнейших разбиений.



a) Диаграмма, не подвергшаяся разбиению



b) Диаграмма, подвергшаяся разбиению

РИСУНОК D-4.4.2

Пример частичных диаграмм дерева блоков

Если используются частичные диаграммы и если дальнейшее разбиение блока не очевидно и/или если не очевидно, где найти продолжающие разбиение диаграммы, то в диаграммы следует включать соответствующие ссылки, используя для этого символы комментария.

D.4.5. *Разбиение канала*

Канал может подвергнуться разбиению независимо от тех блоков, которые он соединяет. Это позволяет изобразить поведение канала. В некоторых случаях для точного изображения того, как канал передает сигнал, требуется изобразить поведение канала. Это достигается за счет того, что канал рассматривается как самостоятельный элемент, для которого окружающей средой являются соединяемые им блоки. Рассматривая канал с этих позиций, мы можем изобразить его структуру в терминах блоков, каналов и процессов.

На SDL/GR разбиение канала изображается с помощью диаграммы подструктуры, как это изображено на рис. D-4.5.1. (На примере изображена подструктура канала C2, приведенного на рис. D-4.3.1.)

Диаграмма подструктуры канала описывает, каким образом канал разбит на подкомпоненты. Диаграмма напоминает диаграмму системы (если не считать связей между блоками). Все указания, приведенные в § D.4.3, сохраняют свою силу и для диаграммы подструктуры канала.

В диаграмме взаимодействия блоков, где имеет место разбиение канала, должна стоять ссылка на диаграмму подструктуры канала, описывающую это разбиение.

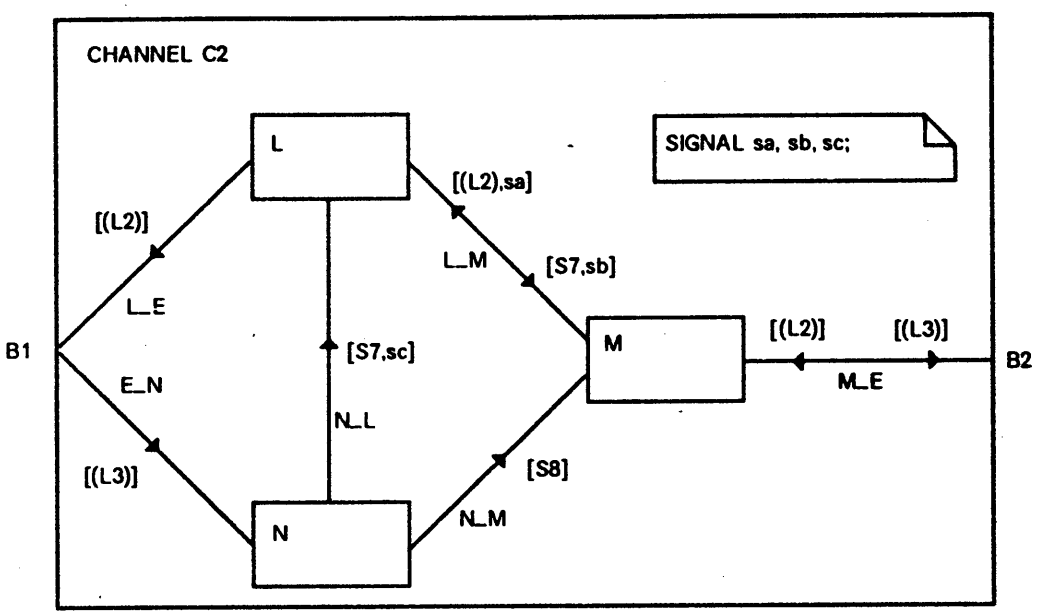


РИСУНОК D-4.5.1

Пример диаграммы подструктуры канала

На SDL/PR форма описания разбиения аналогична форме определения подструктуры блока; единственное различие заключается в том, что в предложении CONNECT оконечные каналы подсоединяются к внешним блокам (блокам B1 и B2 на рис. D-4.5.2), а не к внешним каналам.

```
SUBSTRUCTURE C2;

    SIGNAL sa,sb,sc;

    CHANNEL L_E
        FROM L TO ENV WITH (L2);
    ENDCHANNEL L_E;

    CHANNEL E_N
        FROM ENV TO N WITH (L3);
    ENDCHANNEL E_N;

    CHANNEL M_E
        FROM M TO ENV WITH (L3);
        FROM ENV TO M WITH (L2);
    ENDCHANNEL M_E;

    CHANNEL L_M
        FROM L TO M WITH S7,sb;
        FROM M TO L WITH (L2),sa;
    ENDCHANNEL L_M;

    CHANNEL N_M
        FROM N TO M WITH S8;
    ENDCHANNEL N_M;

    CHANNEL N_L
        FROM N TO L WITH S7,sc;
    ENDCHANNEL N_L;

    CONNECT B1 AND L_E,E_N;
    CONNECT B2 AND M_E;

    BLOCK L REFERENCED;
    BLOCK M REFERENCED;
    BLOCK N REFERENCED;

ENDSUBSTRUCTURE C2;
```

РИСУНОК D-4.5.2

Пример из рисунка D-4.5.1 на SDL/PR

Между блоками и подструктурой канала допускается импорт/экспорт значений. Это делает возможным непосредственное представление модели ВОС, для которой равноуровневая связь моделируется с помощью обмениваемых сигналов, а связь между смежными уровнями – с помощью совместно используемых значений.

D.4.6 *Изображение системы в случае разбиения*

В тех случаях, когда система изображается как совокупность блоков, соединенных каналами, а поведение каждого блока выражается с помощью одного или более процессов, мы получаем одноуровневое представление системы. Это означает, что мы можем обозревать все элементы представления на одном и том же уровне. Однако, вводя разбиение, мы тем самым вводим иерархические взаимоотношения между различными документами. Мы получим документ, содержащий изображение структуры системы, при котором система состоит из "n" блоков.

В каком-то другом документе система может быть представлена как составленная из различных наборов блоков, причем некоторые из них выведены из блоков, содержащихся в предыдущем документе (некоторые блоки в предыдущем документе оказались замененными подблоками, полученными разбиением блоков). Должно быть установлено отношение этого второго документа к предыдущему.

Для того, чтобы получить полное представление системы, недостаточно установить отношения документов друг к другу; помимо этого, они должны быть организованы таким образом, чтобы был возможен подуровневый доступ к описанию системы, причем в направлении от общего обзора системы ко все более и более детальному представлению. Это предполает объединение различных документов в группы таким образом, чтобы они могли образовывать различные уровни представления системы.

Не все уровни обязаны содержать одни и те же элементы. На первом уровне представление системы может состоять из представления блоков и каналов, но без процессов, характеризующих поведение каждого блока. На более низком уровне мы можем захотеть включить представление поведения некоторых, но не всех блоков. Самый нижний уровень представления (самый детализованный) должен содержать полное представление поведения всех блоков, то есть полный набор всех процессов, представляющих это поведение.

Рассматривая дерево блоков, мы можем высказать несколько соображений.

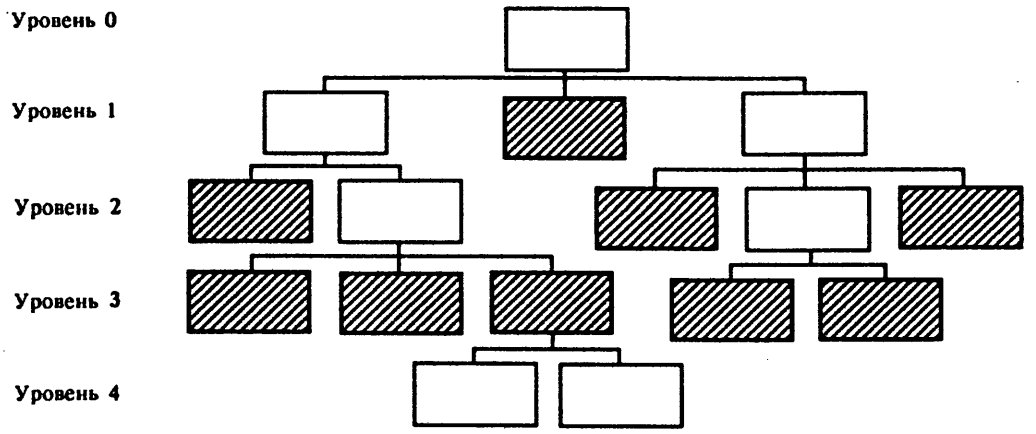
Во-первых, дерево всегда имеет один, и только один, корень – систему. Это имеет место даже в тех случаях, когда система с самого начала изображается как состоящая из нескольких блоков. В дереве блоков эти блоки будут изображены, например, на уровне 1. Корень может состоять только из имени системы. Определения каналов могут быть приведены для блоков уровня 1, хотя это обязательно только в том случае, если блоки содержат процессы.

Второе соображение возникает при рассмотрвнии листьев деревьев: не все они обязаны находиться на одном и том же уровне. Это может явиться следствием различного числа разбиений блоков дерева. Число разбиений зависит от различных обстоятельств, большинство из которых является результатом субъективных оценок разработчика/проектировщика.

Согласно требованиям SDL блок, являющийся листом дерева, не нуждается в разбиении, только если его поведение полностью специфицировано (иными словами, если определение процессов, связанных с блоком, достаточно для представления его поведения). Следовательно, блок, являющийся листом, должен содержать хотя бы один процесс.

Если представление системы проведено на нескольких уровнях абстракции, то для представления системы мы можем выбрать любой из этих уровней. Выбор некоторого уровня означает, что должны быть рассмотрены блоки этого уровня совместно со связанными с ними процессами, а также все блоки, являющиеся листьями более высоких уровней, совместно со связанными с ними процессами (см. рис. D-4.6.1).

Зачастую бывает удобно выбирать различные уровни для различных целей, например, обзорный уровень для получения представления и более детализованный уровень для реализации.



Примечание. — Уровень представления 3 изображен с помощью заштрихованных блоков.

РИСУНОК D-4.6.1

Уровень представления 3

Представление системы на некотором уровне может быть неполным в том смысле, что некоторые из блоков этого уровня могут не иметь связанных с ними процессов.

Причинами обсуждения уровней представления и осуществления выбора некоторого уровня представления системы являются:

– В нашей разработке мы могли достигнуть некоторого "уровня" детализации, который изображается данным уровнем представления (в этом случае блоки этого уровня являются листьями, но они не являются полными, так как требуется дальнейшая проработка).

– Нам может потребоваться рассмотреть представление системы на некотором уровне детализации, поэтому мы выбираем тот уровень представления, который наилучшим образом соответствует требуемой нам абстракции. Обратите внимание на то, что в каких-то случаях некоторый уровень представления может состоять из документов разного уровня абстракции. Часть системы может быть представлена очень детализованно уже на втором уровне, в то время как другая часть будет достаточно абстрактной даже на четвертом уровне. Это означает, что если мы выберем третий уровень представления, то часть системы будет представлена очень детально, а другая часть – лишь в качестве обзора.

– Метод проектирования/представления может быть таким, что каждый уровень имеет конкретное значение: например, первый уровень соответствует спецификации, второй уровень сообщает общую структуру системы, третий уровень обеспечивает модульную структуру (панели, функции программного обеспечения), четвертый уровень обеспечивает детальную структуру (печатные платы, процедуры, программные модули). В этом случае выбор некоторого уровня соответствует какому-то конкретному требованию читателя; следует подчеркнуть, что в этих условиях описываемый метод исключает расхождения в степени детализованности отдельных частей, образующих некоторый уровень представления.

D.4.6.1 *Содержательное подмножество разбиения*

Помимо полной спецификации системы и спецификации, задаваемой уровнями, в SDL существует еще и концепция "содержательного подмножества системы". Ее можно рассматривать как одноуровневую спецификацию, при которой все блоки могут быть взяты из любого уровня структуры системы, но при следующем условии:

– Блок может быть выбран входящим в подмножество содержательного представления системы, если его можно считать листом (то есть либо он просто является листом, либо он содержит все связанные с ним процессы, необходимые для представления его поведения).

– Если выбран некоторый блок, то должны быть выбраны все блоки, образовавшиеся в результате разбиения его родительского блока, и при том либо непосредственно они сами, либо их сыновья.

– Должны быть представлены все документы, определяющие сигналы, текущие по каналу, соединяющему блок, входящий в выбранное представление системы. Из этого, в зависимости от выбранной стратегии разбиения, может следовать, что если взят некоторый блок, то должен быть взят и его родитель, или хотя бы часть родителя, касающаяся определения сигналов и данных.

Если некоторые каналы подверглись разбиению, то, рассматривая их как системы, надлежит обеспечить каждую из таких "систем" спецификацией. Их спецификация состоит из документов такого же типа, как обычное представление систем. Кроме того, должны быть добавлены обозначения, связывающие эти системы со спецификацией той основной системы, к которой они принадлежат. Таким образом, мы можем рассматривать такие подвергшиеся разбиению каналы как внутренние системы. Каждая такая система может содержать несколько уровней представления и в свою очередь содержать внутренние системы, если некоторые из каналов, входящих в рассматриваемую систему, также были подвергнуты дальнейшему разбиению.

D.4.7 *Уточнения*

Механизм уточнения был введен в SDL с двумя целями: "скрыть" сигналы нижних уровней от верхних уровней абстракции и обеспечить возможность спецификации поведения системы методом "сверху вниз".

Уточнение позволяет пользователю разбивать сигналы на подсигналы, в результате чего возникает такая же иерархическая структура, как для блоков и подблоков. Иными словами, внутри определения сигнала можно определить совокупность новых сигналов, которые называются уточненными сигналами или подсигналами того сигнала, в определение которого они входят. Аналогично дереву блоков для определения блока в целях разъяснения может быть составлено дерево сигналов для определения сигнала.

Уточнение тесно связано с разбиением блоков, так как уточнение применимо только к сигналам, которые транспортируются по каналам, подсоединенным к блокам, подвергшимся разбиению. Иными словами, сигнал, содержащийся в списке каналов, может быть замещен своими подсигналами только тогда, когда подвергается разбиению присоединенный блок. Соответствующие подканалы, возникшие при разбиении блока, будут специфицировать подсигналы в своих списках сигналов.

Если для какого-то сигнала определяется канал, по которому будет передаваться данный сигнал, то тем самым автоматически этот канал становится средой передачи всех подсигналов сигнала, даже если некоторые подсигналы будут передаваться в обратном направлении (в этом случае канал считается неявно двусторонним).

Пример уточнения приведен на рис. D-7.4.1. На этом примере изображена система, в которой процесс в одном блоке посылает текстовые файлы процессу в другом блоке. На самом высоком уровне уточнения это достигается посылкой сигналов, причем каждый из них представляет текстовый файл (сигнал sf). На следующем уровне уточнения мы хотим специфицировать тот факт, что текстовый файл состоит из некоторого числа записей, которые посылаются поодиночке (сигнал sr), и что получатель должен посылать назад ответ (сигнал nr) после восприятия каждой записи. В конце передачи отправитель посылает сигнал конец-файла (сигнал eof). В этом примере на самом высоком уровне уточнения процессы в блоках B1 и B2 взаимодействуют, используя сигнал sf; на следующем более низком уровне процессы в B11 и B21 взаимодействуют, используя сигналы sr, nr и eof.

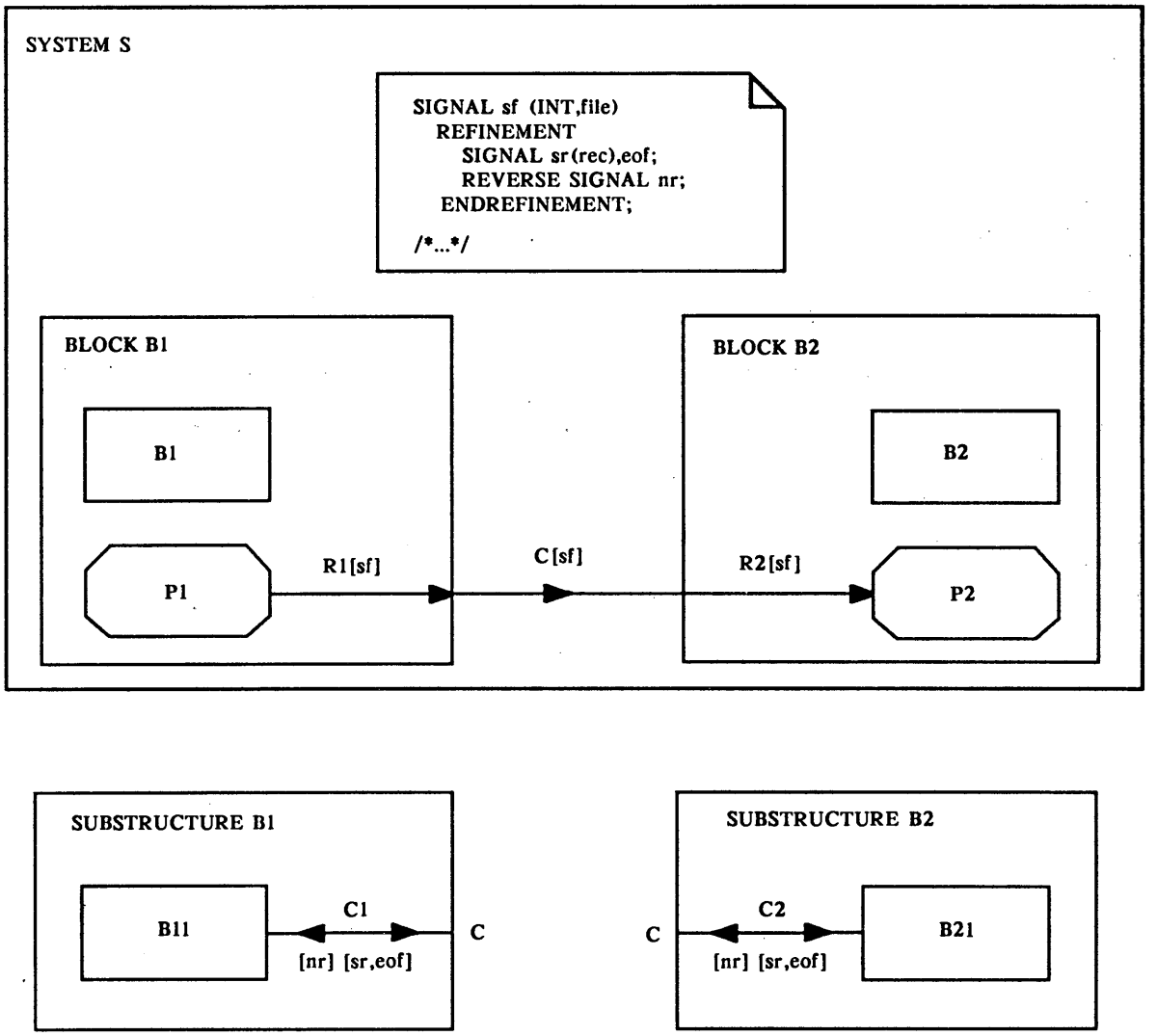


РИСУНОК D-4.7.1

Пример уточнения

Несколькими реальными ситуациями, в которых может быть применена концепция уточнения, являются следующие:

– Преобразование имени: сигнал подвергается уточнению в другой сигнал с новым именем. Это является взаимооднозначным преобразованием, при котором меняются только имена сигналов. Обычно эта возможность используется тогда, когда мы стараемся сделать каждый уровень полностью самим по себе понятным (может оказаться удобным адаптировать имя сигнала к контексту).

– Преобразование с расщеплением: это многозначное преобразование, при котором в результате более подробного описания один сигнал расщепляется на несколько сигналов. Например, порождающий сигнал "Сбой" расщепляется на "Сбой_в_Регистре", "Сбой_в_Центральном_Процессоре" и "Сбой_у_абонента".

– Преобразование алгоритма: исходный сигнал преобразуется в набор сигналов, активизующих алгоритм, в целях поставки исходной информации.

D.4.7.1 *Содержательное подмножество уточнения*

Если в спецификации системы применяется уточнение, то на концепцию содержательного подмножества разбиения накладываются ограничения; это делается с целью исключить возможность связи между различными уровнями уточнения с помощью сигналов различных уровней. В этом случае говорят, что определение системы содержит несколько содержательных подмножеств уточнения. Если содержательное подмножество уточнения содержит процесс, взаимодействующий с помощью подсигналов, то такой процесс может не взаимодействовать с помощью родительского сигнала и процесс на другом конце линии взаимодействия должен взаимодействовать с помощью тех же подсигналов.

D.4.7.2 *Преобразования между сигналами и подсигналами*

У пользователя может возникнуть необходимость описать преобразования одних уровней уточнения в другие в целях либо моделирования, либо проверки поведения системы на различных уровнях детализации. Это можно осуществить неформальными методами с помощью дополнительного SDL-процесса, описывающего динамические преобразования сигналов в подсигналы или наоборот.

На рис. D-4.7.2 для описания преобразования, изображенного на рис. D-4.7.1, введены два процесса. Процесс уточнения (retine_sf) определяет, как происходит уточнение сигнала высокого уровня в набор сигналов следующего более низкого уровня. Восстановительный (retrieve_sf) процесс описывает обратное преобразование.

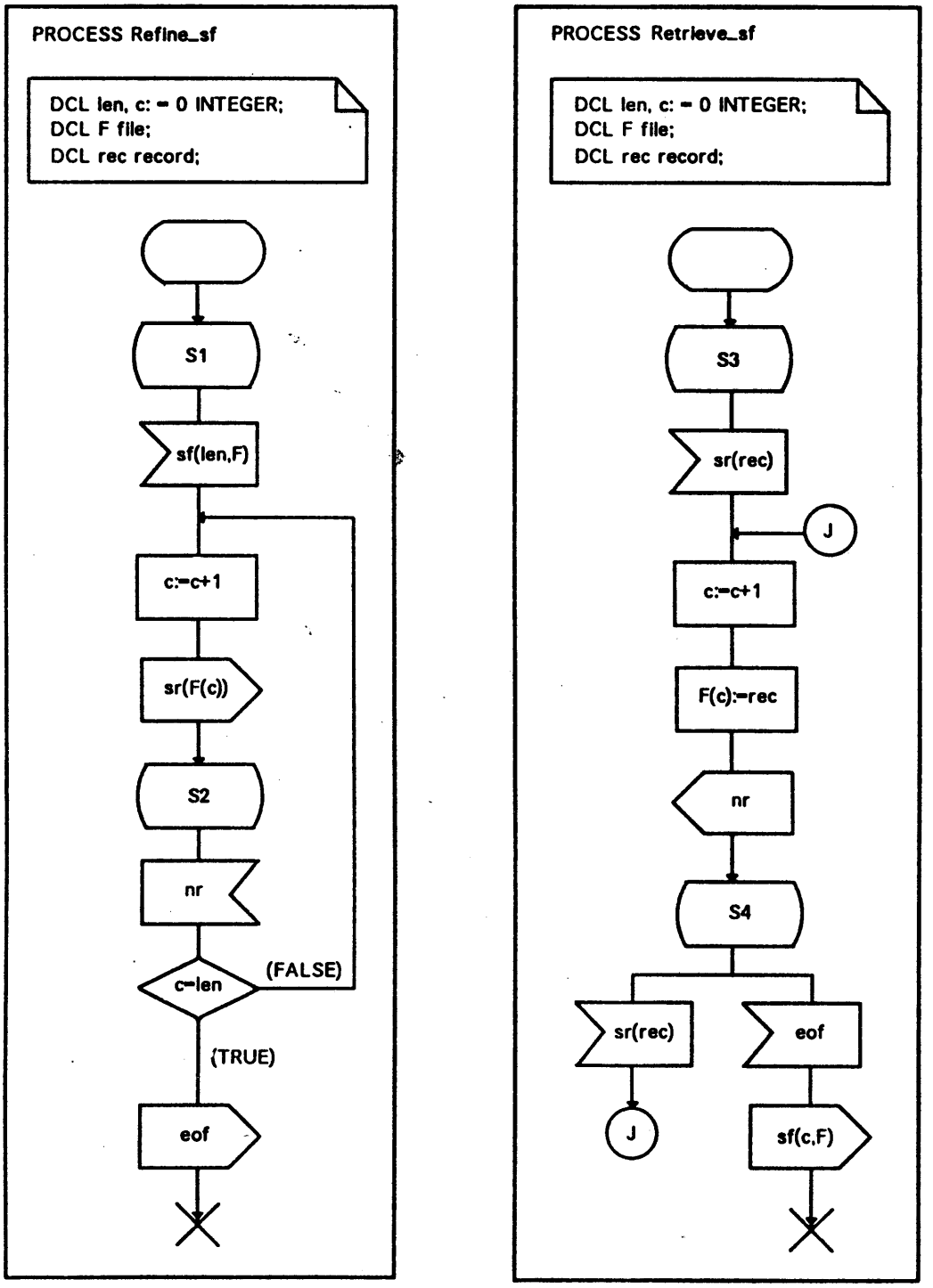


РИСУНОК D-4.7.2

Пример спецификации преобразования сигнала

D.5 *Дополнительные концепции*

D.5.1 *Макрокоманды*

Конструкция, называемая макрокомандой, является средством, позволяющим обращаться с повторами и/или структурировать описание. Она состоит из определения макрокоманды, содержащем часть SDL-спецификации, к которому можно обратиться (осуществить вызов макрокоманды) из какого-либо другого места спецификации системы.

Определение макрокоманды можно поместить в любом месте, в котором можно поместить определение данных. Однако макрокоманды не входят ни в какое подразделение. Таким образом, к макрокоманде, определенной в некотором блоке, можно обращаться извне этого блока.

В SDL/PR определение макрокоманды может заменять любую последовательность лексических единиц. Это отличает SDL/PR от SDL/GR, в котором определение макрокоманд может заменять только синтаксические единицы.

Для отображения между SDL/PR и SDL/GR документов SDL, содержащих макрокоманды, должны соблюдаться следующие ограничения на использование макрокоманд SDL/PR:

1. Макрокоманда может заменять одну или более нижеследующих синтаксических конструкций (которые соответствуют символам графического SDL/GR):

   старт
   состояние
   ввод
   разрешающее условие
   непрерывный сигнал
   сохранение
   предложение действия
   предложение терминатора

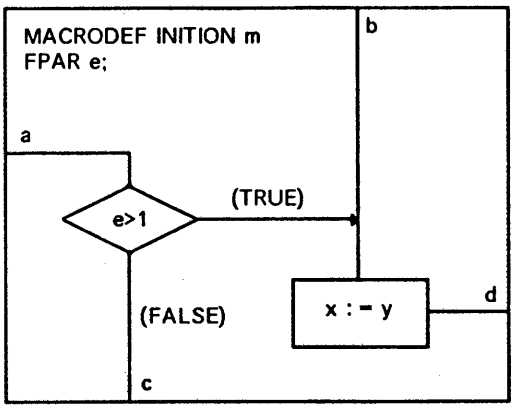
2. Формальные параметры макрокоманды не должны использоваться в местах, в которых определяется тип конструкций SDL/PR. Иными словами, они не должны использоваться в тех местах, в которых используются ключевые слова SDL/PR. Аналогичным образом фактические параметры не должны содержать те ключевые слова, которые соответствуют символам SDL/GR: START, STATE, PROCEDURE, INPUT, TASK, OUTPUT, DECISION, CREATE, STOP, PROVIDED, CALL, COMMENT, JOIN, RETURN, SAVE или OPTION.

3. Любое предложение, входящее в определение макрокоманды, должно быть достижимо по крайней мере из одной входной точки.

В SDL/PR любая макрокоманда всегда имеет не больше одной входной точки и одной выходной точки, поэтому для представления на SDL/PR макрокоманд SDL/GR, имеющих более одной входной или выходной точки, необходимо пользоваться метками и присоединениями соответственно. Если макрокоманда будет возбуждаться в более чем одном месте, то метки должны передаваться как параметры.

В SDL/GR существует два разных способа представления входных и выходных точек определения макрокоманды. Пользователь может либо нарисовать рамку макрокоманды и соединить входные/выходные точки с рамкой, либо использовать явные символы входных и выходных точек (тогда рамка макрокоманды не обязательна).

Пример на рис. D-5.1.1 иллюстрирует макрокоманду на SDL/GR и SDL/PR, содержащую две входные и две выходные точки. (В этом примере входные и выходные точки присоединены к рамке макрокоманды.)




```
MACRODEF INITION m
    FPAR e, a, b, c, d;
    a: DECISION e>1;
        (TRUE): JOIN b;
        (FALSE): JOIN c;
        ENDDECISION;
    b: TASK x : = y;
    JOIN d;
ENDMACRO m;
```

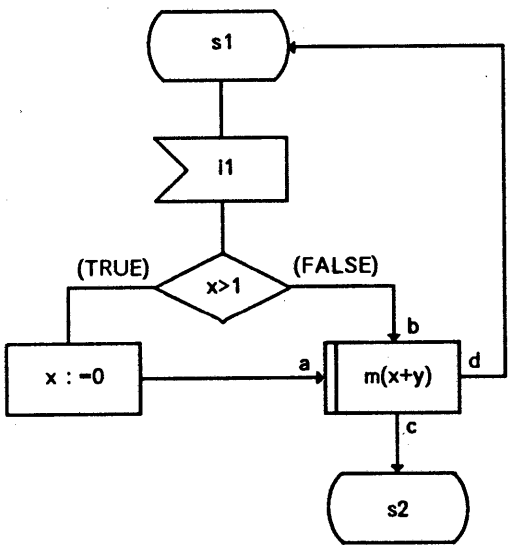
SDL/GR

SDL/PR

РИСУНОК D-5.1.1

Пример определения макрокоманды

Это означает, что в основной спецификации на SDL/PR (рис. D-5.1.2) имеются соответствующие присоединения и метки. Обратите внимание на то, что для каждого вызова макрокоманды метки, вероятно, будут различными.




```
...
STATE s1;
    INPUT I1;
        DECISION x>1;
        (FALSE): JOIN BB;
        (TRUE): TASK x:=0;
                JOIN AA;
        ENDDECISION;
        MACRO m (x+y,AA, BB, CC, DD);
        CC: NEXTSTATE s2;
        DD: NEXTSTATE -;
...
```

SDL/GR

SDL/PR

РИСУНОК D-5.1.2

Пример вызова макрокоманды

На рис. D-5.1.3 приведены два определения макрокоманды, описывающей механизм синхронизации. Обратите внимание на использование псевдоформального параметра MACROID для обеспечения уникальности имен состояний. Тот же пример, но использующий символы входной и выходной точек, повторен на рис. D-5.1.4.

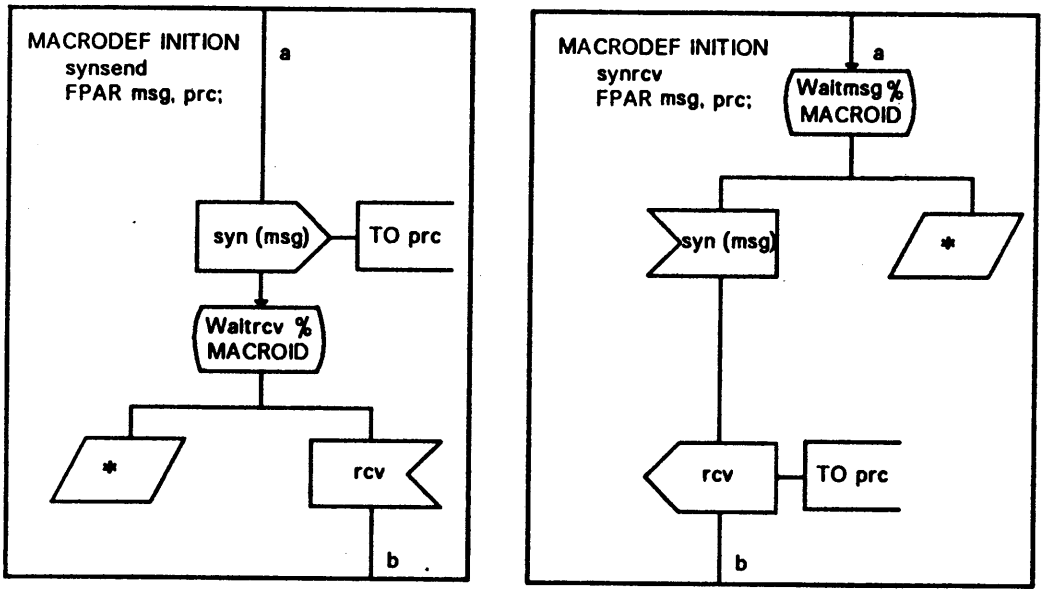


РИСУНОК D-5.1.3

Пример двух макрокоманд, использующих MACROID

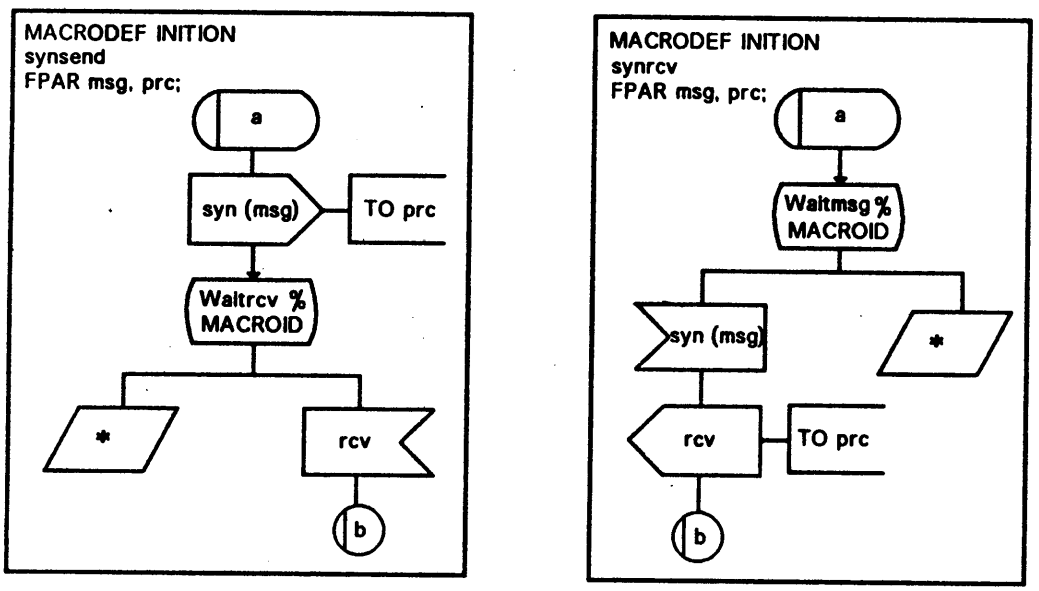


РИСУНОК D-5.1.4

Пример использования символов входных и выходных точек

Следует отметить, что в случае вложенных макрокоманд, то есть когда определение макрокоманды содержит один или более вызовов макрокоманды, расширение внешних макрокоманд никак не влияет на расширение внутренних макрокоманд. Точнее говоря, расширение вложенных макрокоманд должно выполняться так, как если бы расширение макрокоманды завершалось до того, как начиналось расширение возможных внутренних вызовов макрокоманд.

D.5.2 *Порождающие системы*

SDL допускает определение различных систем с помощью одной-единственной спецификации; это достигается с помощью системных параметров. Системные параметры имеют неопределенные значения, которые могут быть предоставлены извне системы; тем самым достигается определение конкретной системы, соответствующее потребностям пользователя.

Фактически системные параметры являются внешними синонимами и могут использоваться всюду, где могут использоваться синонимы. Конечно, перед интерпретацией системы всем ее внешним синонимам должны быть присвоены некоторые значения. На рис. D-5.2.1 представлено несколько примеров корректного использования внешних синонимов.

```
SYSTEM s;
    ...
    SYN число экземп.INTEGER=EXTERNAL;
    SYN приращ.показ. REAL=EXTERNAL;
    ...
    BLOCK b;
        PROCESS p (число экземп.); /* номер параметрического экземпляра*/
            ...
            val:=val + приращ.показ.; /*параметрическое приращение*/
        ENDPROCESS p;
    ENDBLOCK b;
ENDSYSTEM s;
```

РИСУНОК D-5.2.1

Пример использования системных параметров

Кроме того, SDL обеспечивает еще две конструкции, поставляющие более мощные средства выбора, обусловленного внешними синонимами:

– Конструкцию SELECT: условный выбор фрагмента спецификации. Условие специфицируется с помощью булевского выражения, которое должно быть вычисляемо статически, до интерпретации системы. Фрагмент спецификации выбирается тогда, когда значение выражения равно TRUE.

– Конструкцию ALTERNATIVE: допускает условный выбор одного фрагмента спецификации из двух или большего числа фрагментов. Может быть использована только для выбора различных переходов в теле процесса, процедуры или сервиса.

D.5.3 *Сервисы*

D.5.3.1 *Общие положения*

Назначением концепции сервисов является предоставление возможности расчленения определения процесса, не вводящего параллелизм. Каждый сервис можно рассматривать как "функцию", поставляемую процессом. Это является частичным описанием процесса, изображающим "подповедение" процесса. Такое "подповедение" является какой-то одной частью общего поведения. Следовательно, использование концепции сервиса является средством структурирования процесса.

Имеется много доводов в пользу структурирования процесса, например, снижение сложности и повышение доступности. Но структурирование, кроме того, является средством выделения отдельных частей процесса и их отдельного описания. Такие части могут быть "подчастями" функции, поставляемой системой. Таким образом, с помощью концепции сервисов функция системы может быть изолирована за счет описания набора подчиненных сервисов одного или нескольких процессов.

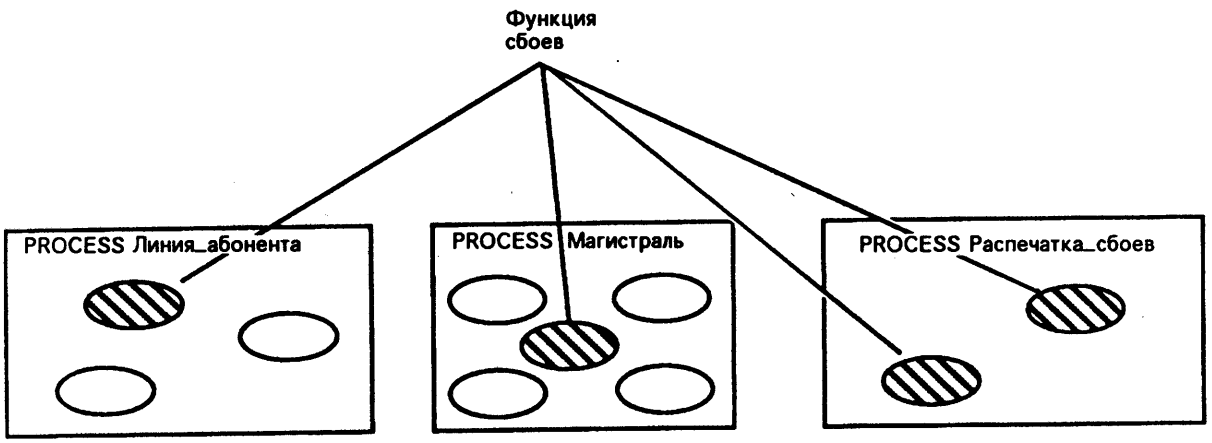


РИСУНОК D-5.3.1

Неформальный пример, показывающий, как сервисы различных процессов могут образовывать функцию системы более высокого уровня

Обратите внимание на то обстоятельство, что язык SDL не обладает никаким средством формирования функции системы за счет собрания сервисов из нескольких процессов. Такое формирование предоставляется пользователю и находится полностью вне языка.

Поскольку сервис отделяется от остальных сервисов и описывается как машина с конечным числом состояний, имеющая свое собственное пространство состояний, то сервис можно разрабатывать и видоизменять, не оказывая при этом никакого влияния на другие сервисы. Следует, однако, отметить, что сервисы процесса имеют зачастую общие данные, а это означает, что они могут оказывать влияние друг на друга за счет манипулирования данными.

Поскольку поведение процесса не меняется от разложения его на сервисы, то сервисы являются всего лишь альтернативным описанием того же поведения. Конечно, проектировщик может в какой-то момент решить смоделировать поведение не одним процессом, а несколькими. Однако это приведет к отличному поведению, так как несколько процессов функционируют параллельно и не могут совместно использовать (читать и писать) данные без сложного обмена сигналами.

Обратите внимание, что структурирование процесса в форме сервисов не означает полного исчезновения процесса. Это означает только, что тело процесса, описывающее поведение процесса, было полностью заменено несколькими телами сервисов. Формальные параметры, объявления и определения на уровне процесса останутся без изменения. В добавление к ним в определения сервиса могут быть добавлены локальные определения и объявления.

Определение сервиса состоит из следующих элементов, причем некоторые из них не являются обязательными:

– Имя сервиса.

– Набор действительных входных сигналов: список идентификаторов сигналов, определяющий те сигналы, которые могут быть получены сервисом.

– Определения процедур: определение процедур, локальных по отношению к сервису. Может использоваться также ссылка на процедуру.

– Определения данных: спецификация определенных пользователем новых типов, синонимичных типов и генераторов, локальных для данного сервиса.

– Определения переменных: объявление переменных, локальных для данного сервиса. Эти переменные не могут быть раскрываемы или экспортируемы другому процессу (не допускаются ключевые слова REVEALED и EXPORTED). Для каждой объявляемой переменной должен быть специфицирован идентификатор ее сорта. Возможна (но не обязательна) спецификация начальных значений.

– Определения обозревания: объявление идентификаторов переменных, которые могут использоваться для получения значения переменных, которыми владеют другие процессы. Для каждого идентификатора переменной должен быть специфицирован сорт переменной.

– Определения импорта: спецификация идентификаторов тех переменных, которыми владеют другие процессы и которые хочет импортировать данный сервис. Для каждого идентификатора должен быть специфицирован сорт переменной.

– Определения таймера: разъясняются в § D.3.11.

– Определения макрокоманд: разъясняются в § D.5.1.

– Тело сервиса: спецификация фактического поведения сервиса в терминах состояний, вводов, выводов, работ и т.д. По сравнению с телом процесса тело сервиса может также содержать посылку и получение приоритетных сигналов (см. § D.5.3.2).

На SDL/GR определение сервиса представлено диаграммой сервиса. Диаграмма сервиса состоит из следующих элементов:

– Необязательного символа кадра: символ, имеющий форму прямоугольника, охватывающего все прочие символы.

– Заголовка сервиса: ключевое слово SERVICE, за которым следует идентификатор сервиса. Заголовок сервиса помещается в левый верхний угол кадра.

– Необязательной нумерации страниц (помещаемой в правый верхний угол).

– Символов текста: символы текста используются для помещения в них определений данных, переменных, обозревания, импорта и таймеров, локальных для данного сервиса.

– Ссылок на процедуры: символ процедуры, содержащий имя процедуры, представляющее процедуру, локальную для данного сервиса и определяемую отдельно.

– Диаграмма макрокоманд: см. указания в § D.5.1.

– Графа сервиса: спецификация фактического поведения сервиса в терминах состояний, вводов, выводов, работ и т.д. По сравнению с графом процесса граф сервиса может также содержать посылку и получение приоритетных сигналов (см. § D.5.3.2).

Пример концепции сервиса приведен на рис. D-5.3.2 для простого процесса "Таймер". Процесс структурирован на два сервиса, на которые в диаграмме имеются ссылки; сервисы определены в двух диаграммах сервиса (рис. D-5.3.3).

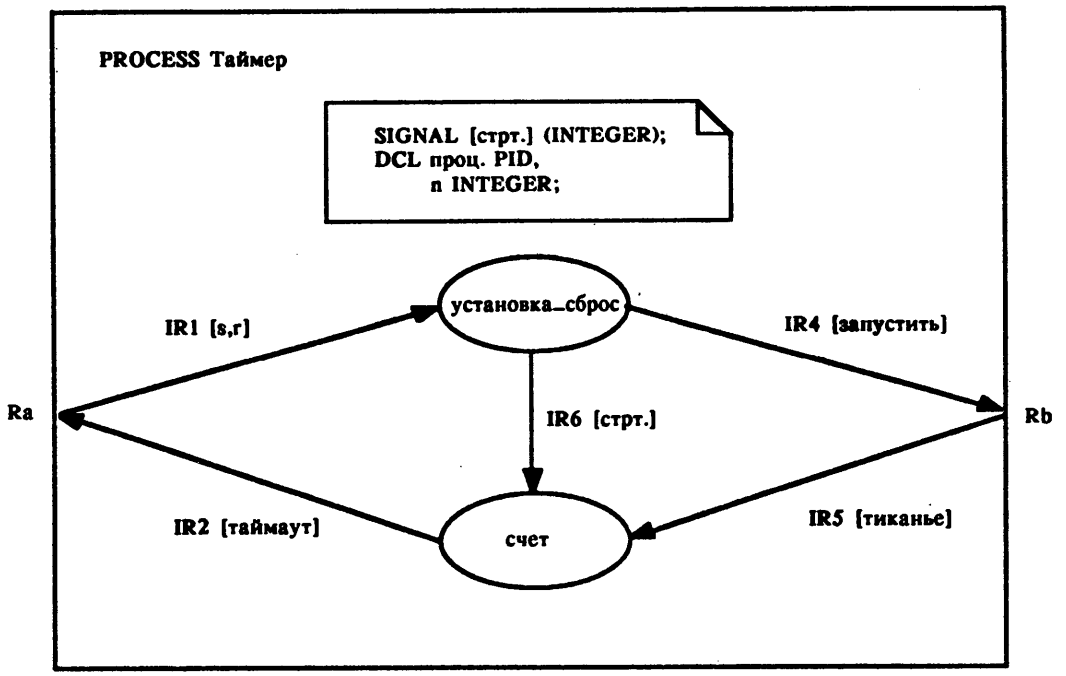


РИСУНОК D-5.3.2

Диаграмма процесса, содержащая ссылки на сервисы

Между двумя сервисами имеется внутренний интерфейс, осуществляемый маршрутом сигнала 'IR6', передающем "приоритетный сигнал" (см. § D.5.3.2).

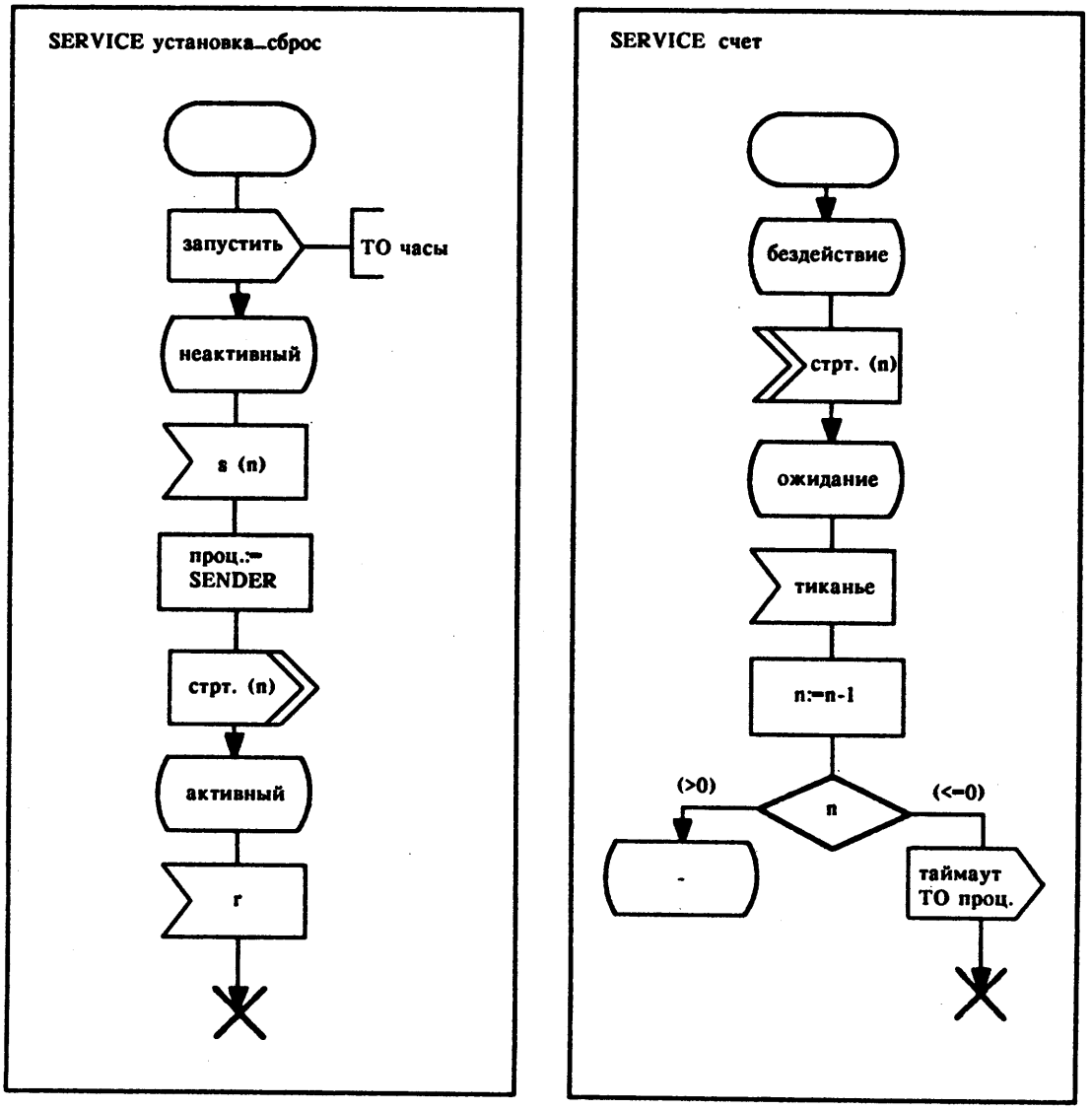


РИСУНОК D-5.3.3

Диаграммы сервисов

Следует отметить, что, поскольку в стартовый переход первого сервиса включены действия (вывод), то в стартовом переходе второго сервиса никакие действия не допустимы.

Еще один пример концепции сервиса приведен в § D.10.

D.5.3.2 *Приоритетные сигналы*

Приоритетные сигналы используются для связи между двумя переходами в различных сервисах одного и того же процесса в тех случаях, когда не разрешено восприятие никаких внешних сигналов (от других процессов) в интервале времени между переходами. Таким способом осуществляется "конкатенация" переходов.

Нижеследующий рисунок (рис. D-5.3.4) иллюстрирует конкатенацию переходов при использовании приоритетов приоритетных сигналов.

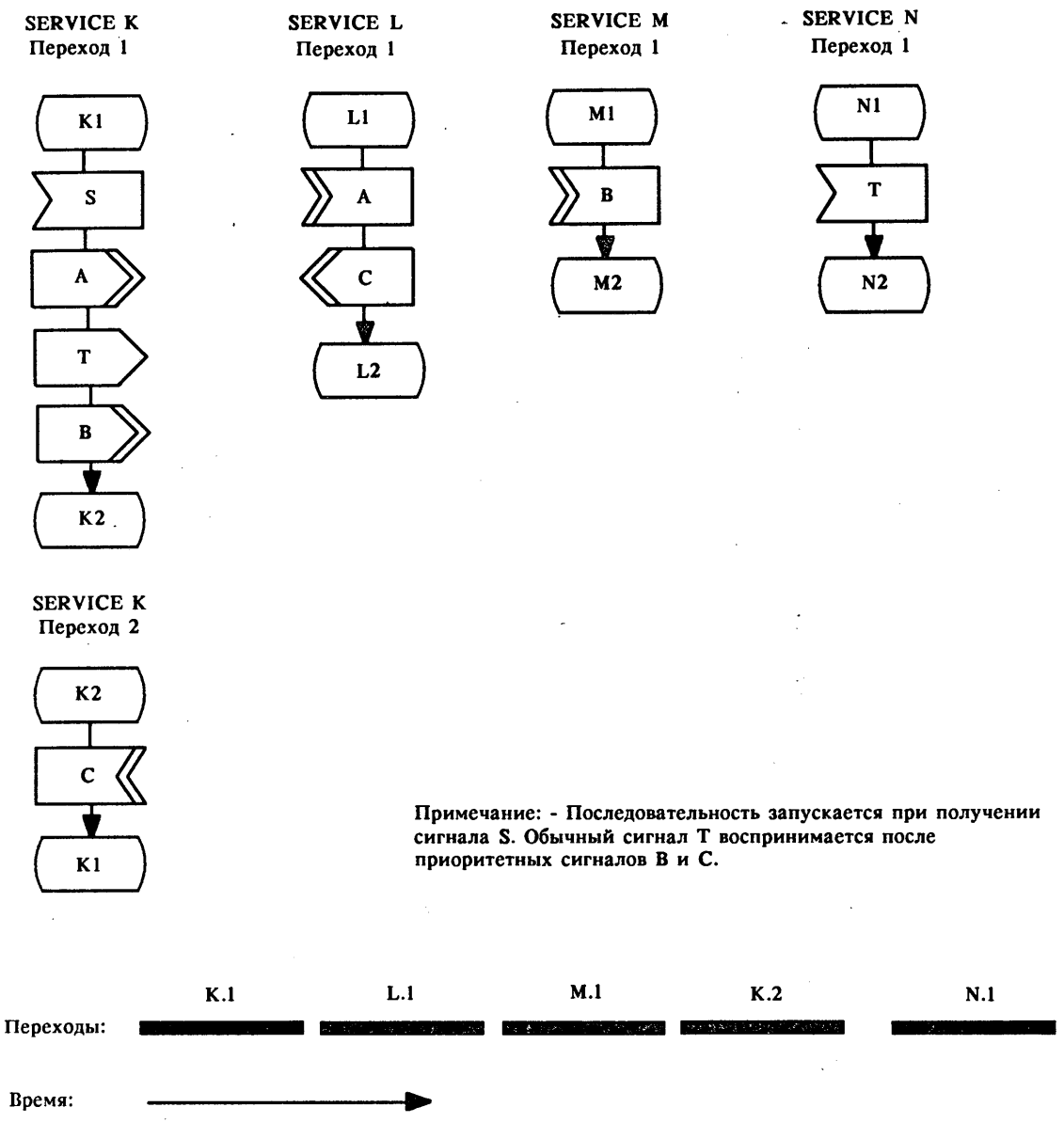


РИСУНОК D-5.3.4

Конкатенация переходов в различных сервисах процесса (неформальный SDL)

Конструкция получения приоритетного сигнала при использовании обычной входной очереди разъясняется в рекомендации. Посылка приоритетных сигналов предварительному состоянию позволяет рассматривать их как обычные сигналы, вызывающие переход к основному состоянию (см. также D.5.3.3.1).

Нижеследующий пример является иллюстрацией результата использования приоритетных сигналов. При разъяснении примера не используется модель "предварительное/основное состояние".

Временная схема (рис. D-5.3.5) взята из взаимодействия между сервисами процесса 'SUBSCRIBER_LINE', описываемого в № D.10.2.

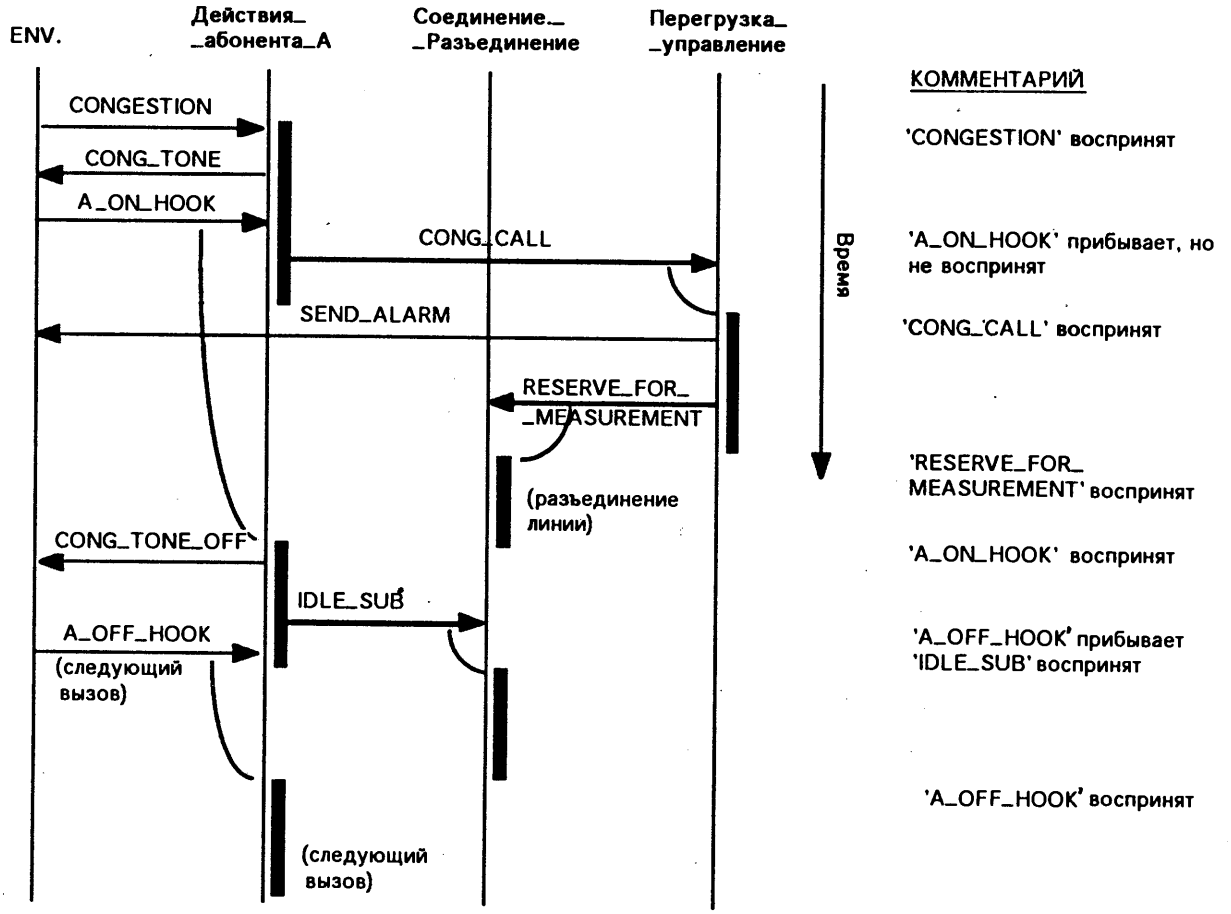


РИСУНОК D-5.3.5

Временная схема, иллюстрирующая то, как приоритетные сигналы могут изменить порядок переходов. Каждая жирная вертикальная линия обозначает выполнение одного перехода

Схема содержит как сигналы к/от окружающей среды процесса, так и приоритетные сигналы между сервисами. Линии стрелок приоритетных сигналов толще обычных. Порядок стрелок сигналов (сверху вниз) показывает, как сигналы помещаются в очереди. Жирные вертикальные линии показывают, как упорядочено во времени выполнение переходов.

Последовательность действий начинается с поступающего из регистра сигнала 'CONGESTION', что означает, что вызов должен быть отвергнут. Это означает также, что будет отвергнут и поступающий немедленно следующий вызов.

Рисунок показывает, что переход, активизированный приоритетным сигналом, например 'RESERVE_FOR_ MEASUREMENT', начинается до перехода, активизируемого обычным сигналом, например 'A_ON_HOOK', невзирая на обратный порядок в очереди. Переход, активизированный сигналом 'RESERVE_FOR_MEASUREMENT', разъединяет линию абонента, что означает, что будет немедленно отвергнут следующий вызов (сигнал 'A_OFF_HOOK').

В нижеследующей временно́й схеме рассматривается та же ситуация, но в диаграммах приоритетные сигналы не используются. Сервисы взаимодействуют с помощью обычных сигналов.

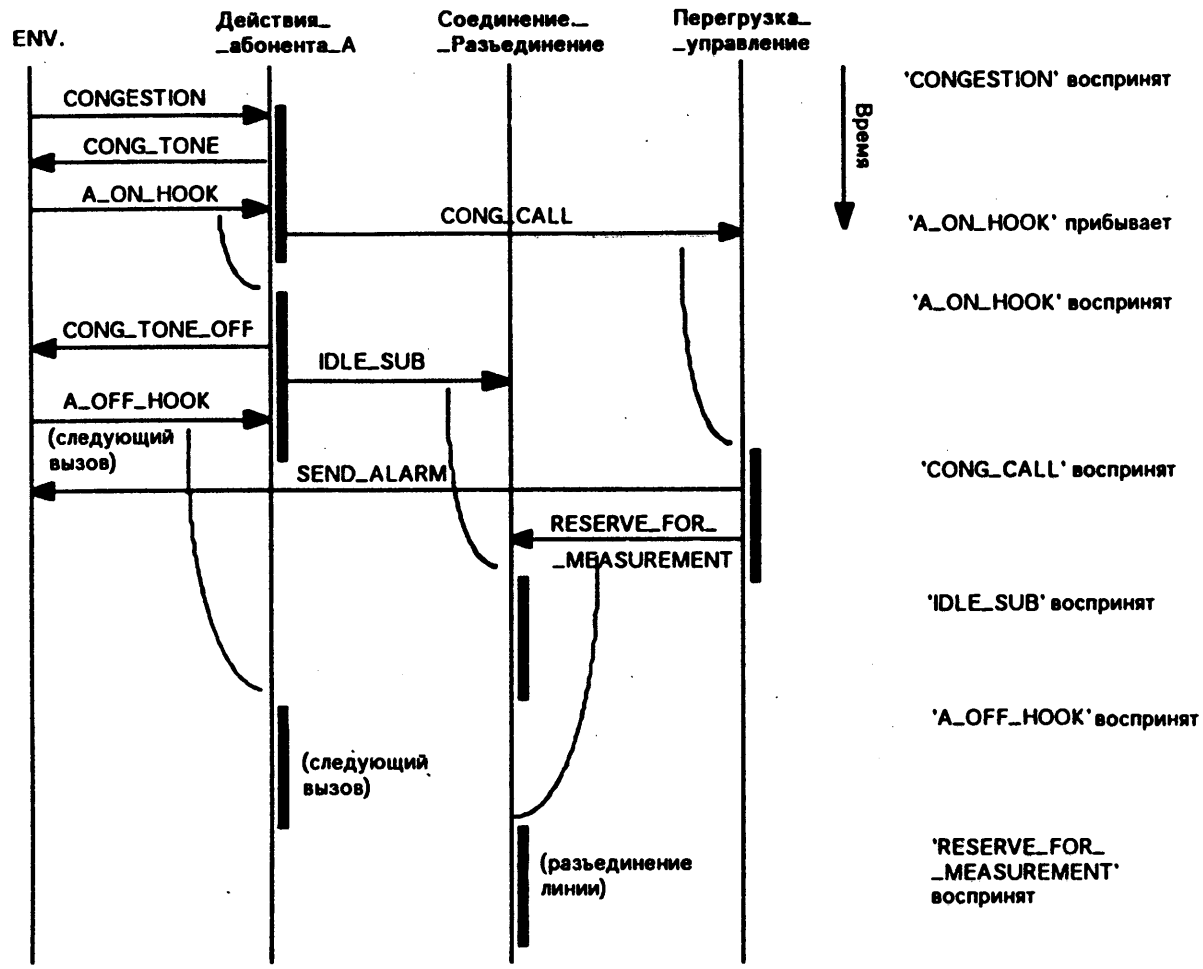


РИСУНОК D-5.3.6

Временна́я схема, содержащая обычные сигналы и естественный порядок выполнения переходов

Сравнив с рис. D-5.3.5, мы увидим, что порядок выполнения переходов изменился. Следующий вызов поступает до того, как линия абонента была разъединена; это означает, что вызов не будет немедленно отвергнут.

D.5.3.3 *Преобразования*

Сервисы и приоритетные сигналы считаются дополнительными концепциями, так как они составлены (смоделированы) из более основных концепций SDL, таких как процесс и сигнал. Для возможности семантической интерпретации сервисов и приоритетных сигналов они должны быть преобразованы обратно к этим основным концепциям. Обычно пользователь не должен выполнять такие преобразования, и во всяком случае вручную, но, конечно, он должен быть осведомлен о таких преобразованиях. Инструментарий семантической проверки сервисов осуществляет такое преобразование автоматически.

После преобразования сервисы исчезают и заменяются расширенным определением процесса, которое может быть семантически интерпретировано. Преобразование определило поведение.

D.5.3.3.1 *Преобразование состояний*

В рекомендации приводятся специальные правила для преобразования состояний. В результате выполнения этих правил число состояний процесса оказывается равным произведению числа состояний сервисов.

Кроме того, наличие приоритетных сигналов удвоит это число, так как каждое преобразованное состояние расщепляется на предварительное и основное. Это удвоение числа состояний, связанное с наличием приоритетных сигналов, не рассматривается в нижеследующем примере преобразования состояний.

Во многих случаях большое число состояний в "трансформированном" процессе не является существенным и пространство состояний может быть сокращено. Это рассматривается в нижеследующем примере, взятом из § D.10.2.

В процессе 'SUBSCRIBER_LINE' имеется 4 сервиса: 'Действия_абонента_A', 'Действия_абонента_B', 'Соединение. Разъединение' и 'Перегрузка_управление'. Число состояний равно 10, 5, 3 и 2, то есть всего 20.

Применяя правила преобразования состояний этих сервисов, мы получим процесс, содержащий 300 состояний (10*5*3*2), что означает наличие 300 составных имен. Размер такой составной *n*-ки равен 4 (4 сервиса). Порядок в *n*-ке определен в соответствии со следующим рисунком (рис. D-5.3.7).

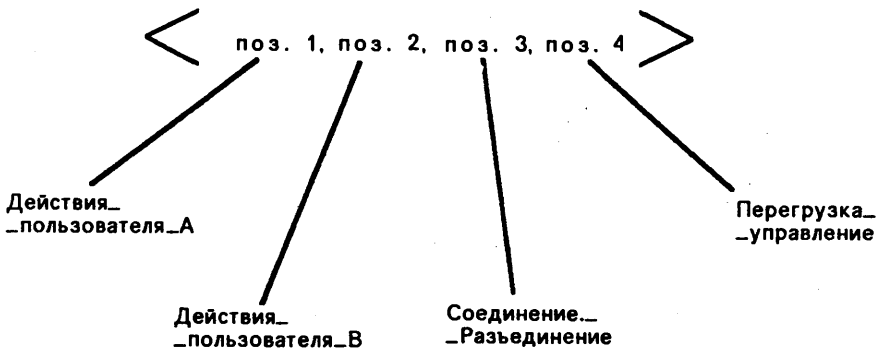


РИСУНОК D-5.3.7

Расположение позиций в именной n-ке

Все возможные имена состояний в различных позициях дают 300 комбинаций.

Ex. <A_IDLE, B_IDLE, CONNECTED, NO_ALARM>
<AWAIT_CONN, B_IDLE, CONNECTED, NO_ALARM>
<AWAIT_FIRST_DIGIT, B_IDLE, CONNECTED, NO_ALARM>
. . . . .
<AWAIT_A_ON_HOOK_2, B_IDLE, CONNECTED, NO_ALARM>

<A_IDLE, B_RINGING, CONNECTED, NO_ALARM>
<A_IDLE, B_CONVERSATION, CONNECTED, NO_ALARM>
. . . . . .
<A_IDLE, AWAIT_B_ON_HOOK, CONNECTED, NO_ALARM>
. . . . . .
. . . . . .
<AWAIT_A–ON_HOOK_2, AWAIT_B_ON_HOOK, SEIZED, ALARM>

Многие из этих комбинаций не действительны, так как линия абонента не может использоваться как сторона A_абонента, так и сторона B_абонента в одном и том же вызове. Это означает, что 10 состояний в 'Действия_абонента_A' могут быть комбинированы только с одним состоянием в 'Действия_абонента_B', а именно с состоянием 'B_IDLE'. Кроме того, это означает, что 5 состояний в 'Действия_абонента_B' могут быть комбинированы только с одним состоянием в 'Действия_абонента_A' (A_IDLE). Таким образом, число существенных состояний равняется (10*1*3*2) + (1*5*3*2) = 90.

Сокращение пространства состояний, такое как в рассмотренном примере, зависит от конкретного случая и не может быть формализовано в виде общих правил, приложимых к автоматическому преобразованию. Если, однако, преобразование осуществляется вручную, то, по всей вероятности, такое сокращение пространства состояний возможно. Следовательно, если выполняется сравнение мощности процесса, разработанного без расчленения на сервисы, со сложностью того же процесса, расчлененного на сервисы, то для корректности сравнения должен рассматриваться процесс с сокращенным числом состояний. В большинстве случаев использование сервисов существенно сокращает число состояний.

D.5.3.3.2 *Преобразование переходов*

В рекомендации приводятся специальные правила для преобразования переходов. Нижеследующий пример является иллюстрацией такого преобразования. В примере используется процесс, специфицирующий простой протокол. Процесс разбит на два сервиса: Отправка и Получение.

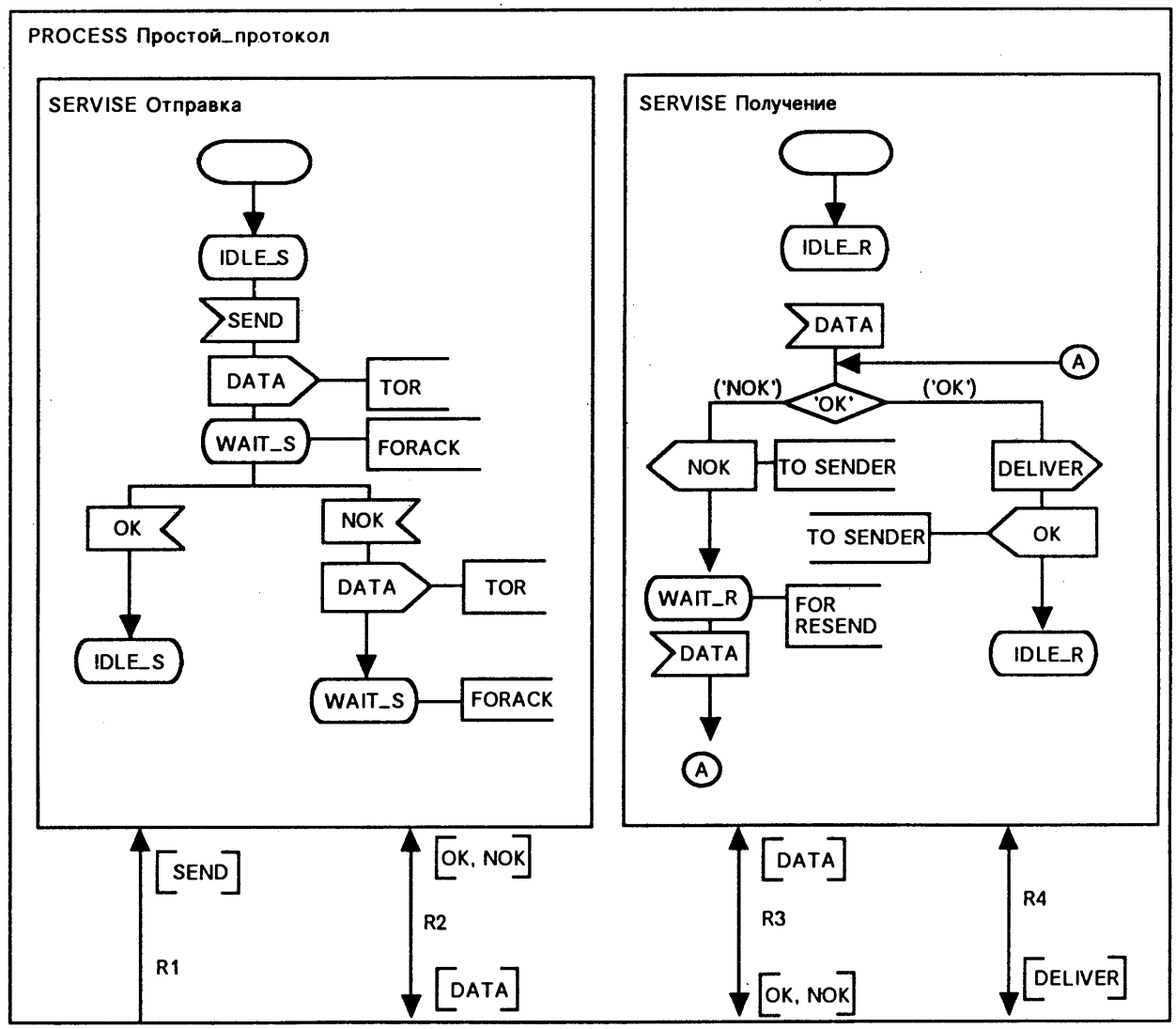


Диаграмма процесса, содержащего два сервиса

На нижеследующем рисунке (D-5.3.9) приведены обзорные диаграммы состояний двух сервисов. Переходы пронумерованы от 1 до 7.

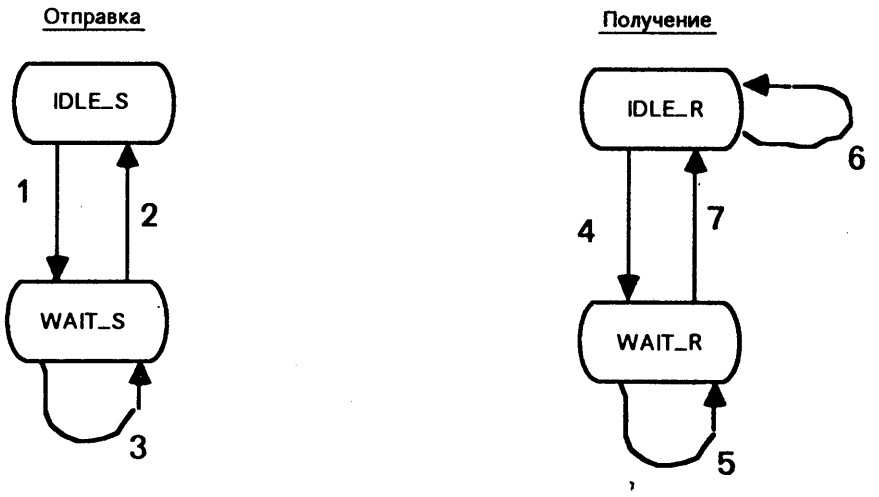


РИСУНОК D-5.3.9

Диаграмма обзора состояний с нумерованными переходами

Применив формальные правила преобразования состояний и переходов, мы получим нижеследующую диаграмму обзора состояний процесса. Поскольку приоритетные сигналы не используются, то нет необходимости в дополнительном расщеплении состояний на предварительные и основные; поэтому такое расщепление на рисунке не указано.

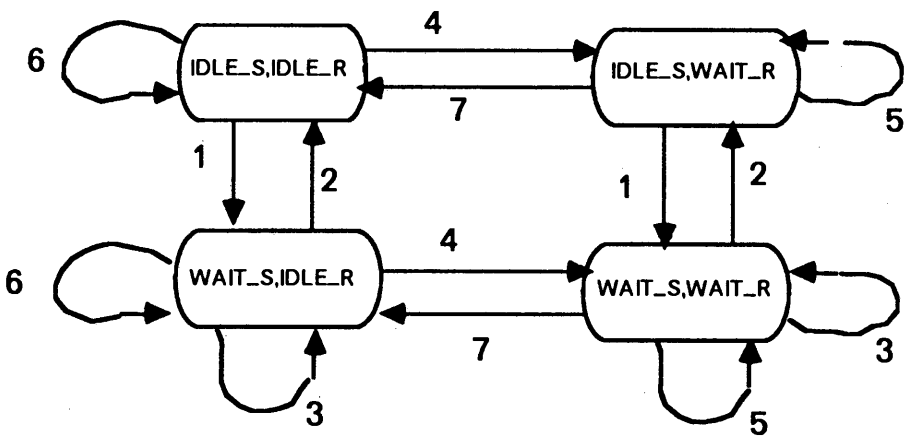


РИСУНОК D-5.3.10

Диаграмма обзора состояний для преобразования сервисов

Число состояний не может быть понижено и граф процесса изображен на рис. D-5.3.11. Имена состояний в графе процесса соответствуют парным именам на рис. D-5.3.10. Пример: IS.IR соответствует паре IDLE_S,IDLE_R.

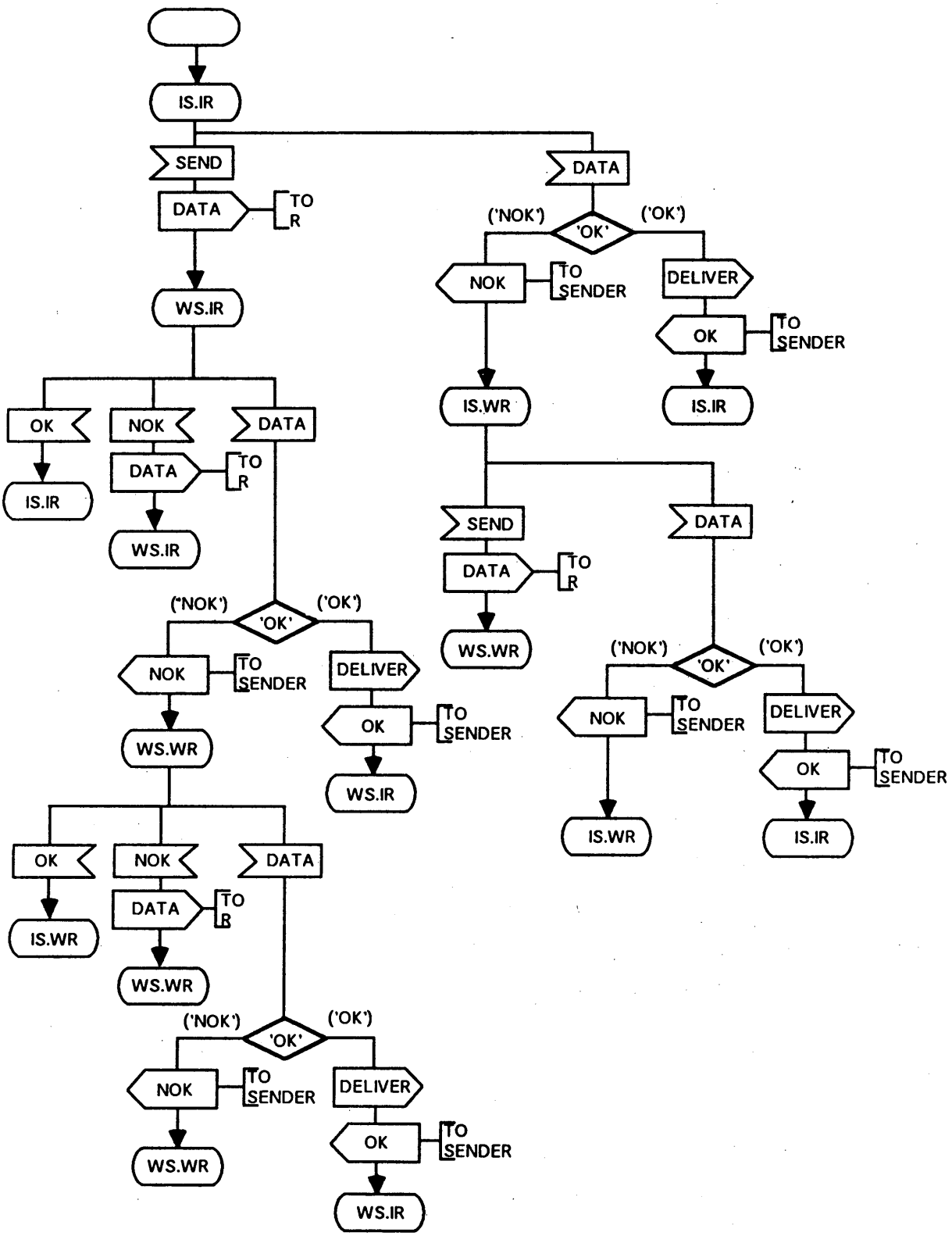


РИСУНОК D-5.3.11

Граф процесса, полученный преобразованием двух графов сервисов

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.5

D.5.4 *Указания по изображению, ориентированному на состояния, и картинные элементы*

В настоящей главе разъясняются изображения на SDL/GR, ориентированные на состояния, и картинные элементы.

Первый раздел (параграф D.5.4.1) содержит общие комментарии, касающиеся изображения, ориентированного на состояния; его свойств, подходящих приложений и различных вариантов.

Второй раздел (параграф D.5.4.2) разъясняет метод описания состояний с помощью картинных элементов.

D.5.4.1 *Общие комментарии к изображению, ориентированному на состояния*

SDL/GR предоставляет три различные версии описания диаграммы процессов. Первая версия SDL/GR называется ориентированной на переходы; в ней переходы описываются исключительно с помощью явных символов действия.

Вторая версия SDL/GR называется ориентированной на состояния или ориентированным на состояния картинным расширением SDL, в котором состояния процесса описываются с помощью картин состояний, а переходы выступают неявно за счет разницы между исходным и завершающим состояниями.

Последняя версия называется комбинированной и представляет собой комбинацию двух вышеуказанных.

Примеры этих трех версий приведены в Приложении E настоящего Выпуска.

Версия, ориентированная на переходы, удобна в тех случаях, когда важна последовательность действий, а детализованное описание состояний не имеет столь важного значения.

Версия, ориентированная на состояния, описывает состояния в виде объявлений, причем детально, поэтому такая версия удобна тогда, когда состояния процесса более важны, чем последовательность действий, совершаемых при переходе, когда желательны интуитивные объяснения в виде картинок и когда вызывают интерес ресурсы и их взаимоотношения в связи с состояниями процесса.

Картинки состояний обычно описываются с помощью картинных элементов, которые обозначают ресурсы, относящиеся к текущему состоянию процесса. Они удобны для тех приложений, для которых определены подходящие картинные элементы. Поэтому пользователь в случае необходимости может применить этот метод представлений к любому приложению, подобрав подходящие картинные элементы.

Комбинированный способ оказывается удобным тогда, когда рассмотрению подлежат как последовательность действий во время перехода, так и детальное описание состояний.

D.5.4.2 *Картина состояния и картинный элемент*

D.5.4.2.1 *Картинный элемент и пояснительный текст*

Если выбрана версия картин состояний, то картина состояния состоит из картинных элементов и пояснительного текста, изображенных на рис. D-5.4.1 a) – d).

Такая комбинация делает понятной картину состояний. Например, на рис. D-5.4.1, пример a), разъясняется смысл обрабатываемого процессом приемника номера, набранного с номеронабирателя, а на примере b) разъясняется смысл генератора тона номеронабирателя, посылающего окружающей среде непрерывный сигнал.

Обратите внимание, что выходные сигналы (но не непрерывные сигналы) и соответствующий источник не описываются в картине состояния; выходные сигналы могут быть описаны в диаграмме переходов.

На примере c) изображен таймер, истечение срока которого всегда представлено вводом. Обратите внимание, что рекомендуемый картинный элемент для изображения таймера включает соответствующий входной сигнал t1.

Последний пример d) означает, что устройство записи речи в настоящий момент ведет запись.

Имя ресурса может быть сильно сокращено; всюду, где это возможно, желательно помещать его внутрь соответствующего картинного элемента. В таком случае вполне очевидно, какие картинные элементы поясняются.

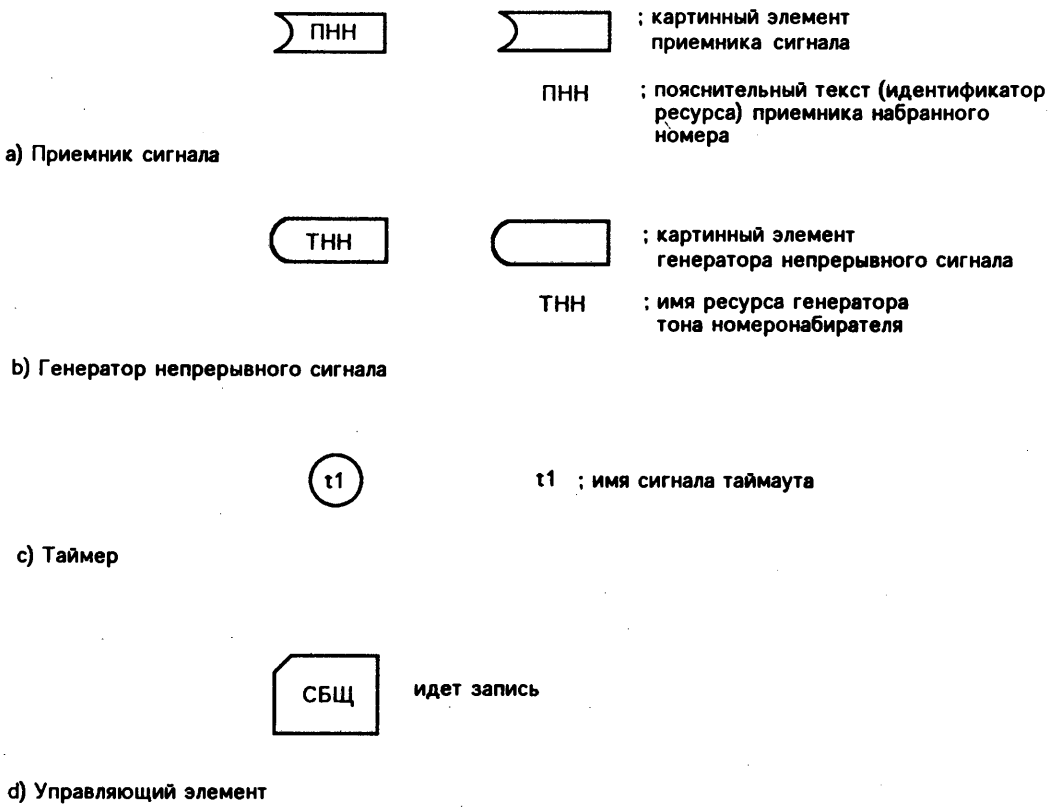


РИСУНОК D-5.4.1

Примеры картинных элементов с пояснительным текстом

D.5.4.2.2 *Полнота картины состояния*

Число картинных элементов и пояснительного текста, включаемых в каждую картину состояния, должно быть достаточным, чтобы показать:

a) какие объекты или ресурсы рассматриваются процессом в текущем состоянии. Примерами ресурсов являются: коммутируемые каналы, приемники сигналов, источники непрерывных сигналов и коммутирующие модули;

b) сколько – один или более – таймеров управляют в текущий момент данным процессом;

c) если данный процесс занят обработкой вызова, то выполняется ли в данный момент начисление платы за вызов и на каких абонентов была начислена плата в данном состоянии вызова;

d) какие объекты, принадлежащие другим процессам (окружающей среде), рассматривались как имеющие отношение к ресурсам процесса в данном состоянии;

e) непрерывные выходные сигналы, посылаемые в данном состоянии;

f) соотношения между сигналами и ресурсами в данном состоянии;

g) состояние ресурсов, имеющих отношение к текущему состоянию процесса.

D.5.4.2.3 *Пример*

На рис. D-5.4.2 приведен пример картины состояния, иллюстрирующий применение вышеизложенных принципов. Как видим, в течение этого состояния:

a) ресурсы, рассмотренные процессом, состоят из приемника набранной на номеронабирателе цифры, генератора тона номеронабирателя, телефонного аппарата абонента, принадлежащего окружающей среде, и коммутируемых каналов, соединяющих эти элементы;

b) процессом управляет таймер t0;

c) начисление платы не осуществляется;

d) абонент идентифицируется как сторона _ A, но никакая другая информация не принята в расчет;

e) ожидаются следующие входные сигналы: A_трубка_положена (то есть сигнал "трубка телефонного аппарата не снята"), цифра (то есть цифра, набранная на номеронабирателе) и t0 (то есть работает управляющий таймер t0);

f) до этого состояния и в течение этого состояния выдавался непрерывный выходной сигнал DT (то есть тон номеронабирателя).

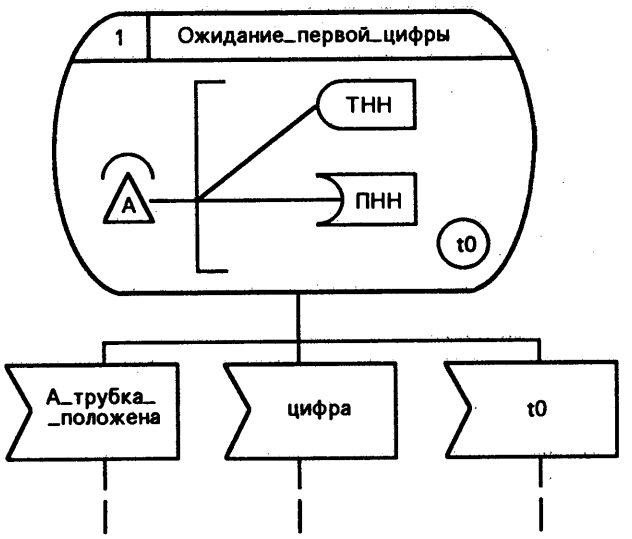


РИСУНОК D-5.4.2

Пример одного из состояний процесса обработки вызова

D.5.4.2.4 *Свойство проверки согласованности SDL-диаграмм, содержащих картинные элементы*

Очевидно, что картина состояния более компактна и в каком-то смысле открывает взору читателя большую информацию; но с другой стороны, тому же читателю предстоит очень тщательно рассмотреть все ресурсы, чтобы выработать точный набор действий, выполняемых при переходе.

Кроме того, из одного лишь рассмотрения картины состояния невозможно вывести порядок действий в переходе.

Картинные элементы, помещенные вне границ символа блока, являются элементами, не управляемыми непосредственно данным процессом, но картинные элементы, помещенные внутри границ символа блока, являются теми элементами, которыми непосредственно управляет данный процесс. Например, процесс вызова, частично специфицированного на рис. D-5.4.3, может занять или освободить генератор тона звонка, генератор звонка и каналов диалога, остановить или запустить таймер t4, но он не может непосредственно управлять состоянием трубки телефонного аппарата.

При выработке логики перехода, исходя из SDL-спецификации, содержащей картинные элементы, на действия процесса, выполняемые в цепочке перехода, окажут влияние только картинные элементы, помещенные внутри границ блока. Картинные элементы, помещенные вне границ блока, обычно включают в картину состояния:

a) поскольку они задают ресурсы и состояние окружающей среды, которые связаны с сигналами, входными для процесса в течение данного состояния;

b) чтобы повысить интеллектуальность диаграммы.

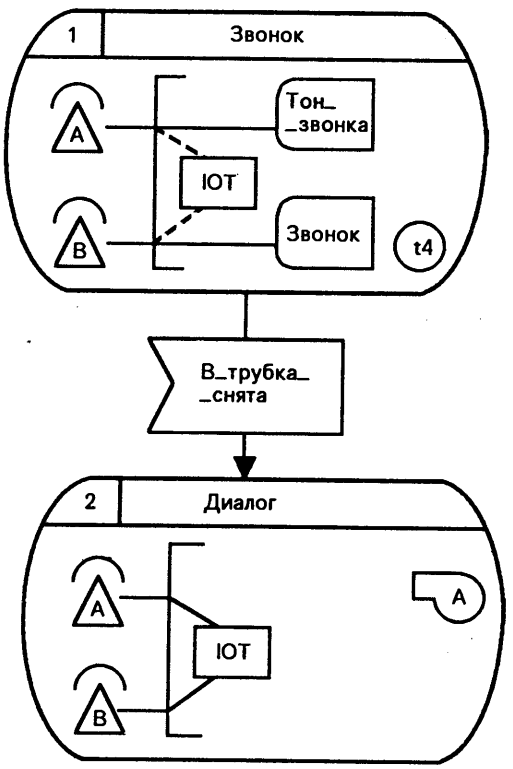


РИСУНОК D-5.4.3

Пример перехода между двумя состояниями, при котором все действия процесса неявно заданы разницей в картинах состояний

D.5.4.2.5 *Использование символа таймера*

Независимо от того, используются или нет картинные элементы, истечение срока таймера всегда изображается в виде ввода.

Наличие символа таймера в картине состояния означает, что в процессе этого состояния таймер работает.

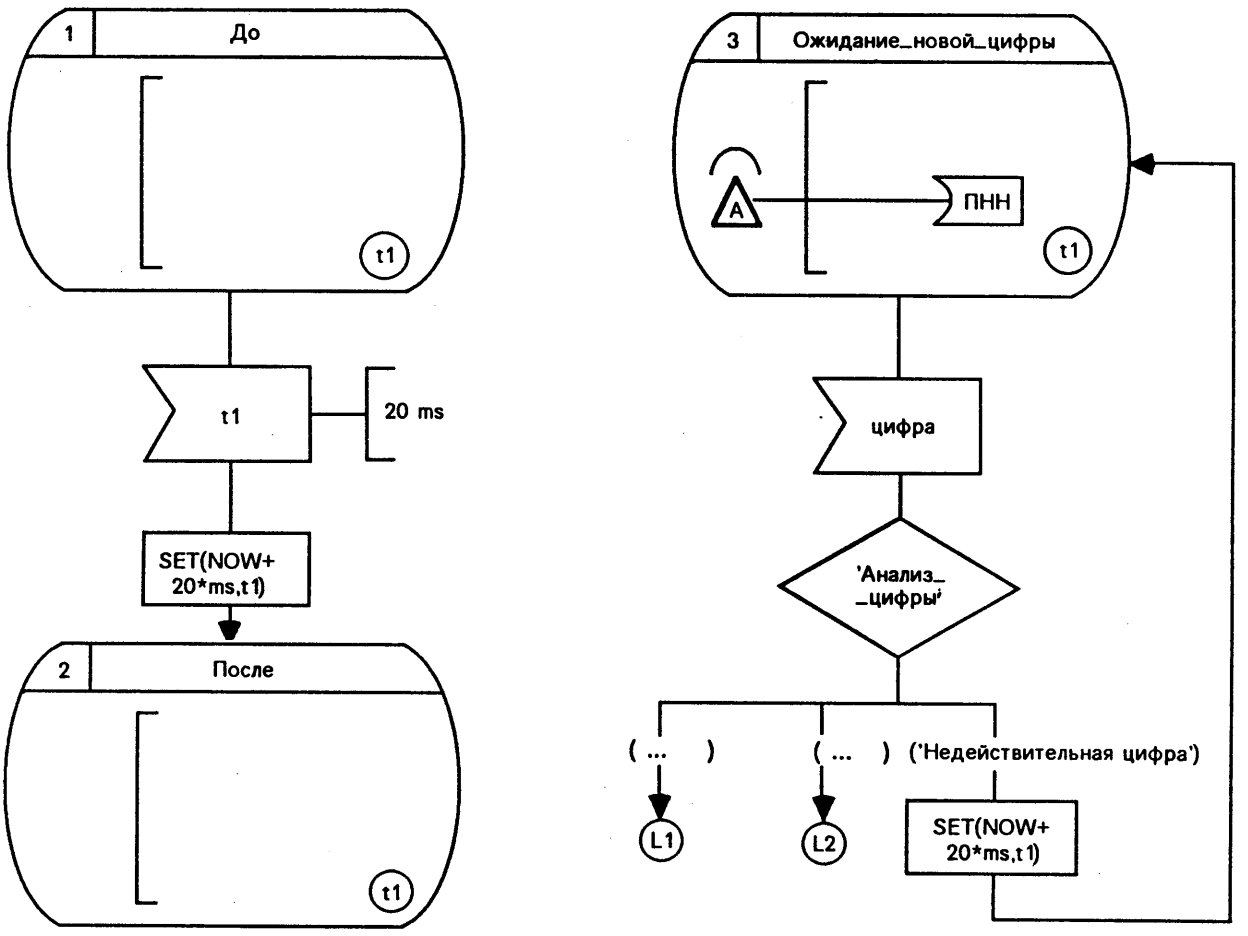
В соответствии с общими принципами, принятыми в Рекомендациях, запуск, остановка, повторный запуск и истечение срока таймеров изображаются с помощью картинных элементов следующим образом:

a) Чтобы показать, что таймер начал работать в течение данного перехода, символ таймера должен быть помещен в картину состояния, соответствующую концу этого перехода, а не в картину состояния, соответствующую началу перехода.

b) Чтобы показать, что таймер был остановлен в процессе некоторого перехода, символ таймера должен быть помещен в картину состояния, соответствующую началу этого перехода, но не в картину состояния, соответствующую концу этого перехода.

c) Чтобы показать, что в процессе перехода таймер был вновь запущен, в этом переходе должен быть явно изображен символ работы. (Два примера приведены на рис. D-5.4.4.)

d) Истечение срока некоторого конкретного таймера изображается с помощью символа ввода, ассоциированного с тем состоянием, картина которого содержит символ соответствующего таймера. Конечно, в случае необходимости управлять процессом могут более чем один таймер, как это показано на рис. D-5.4.5.



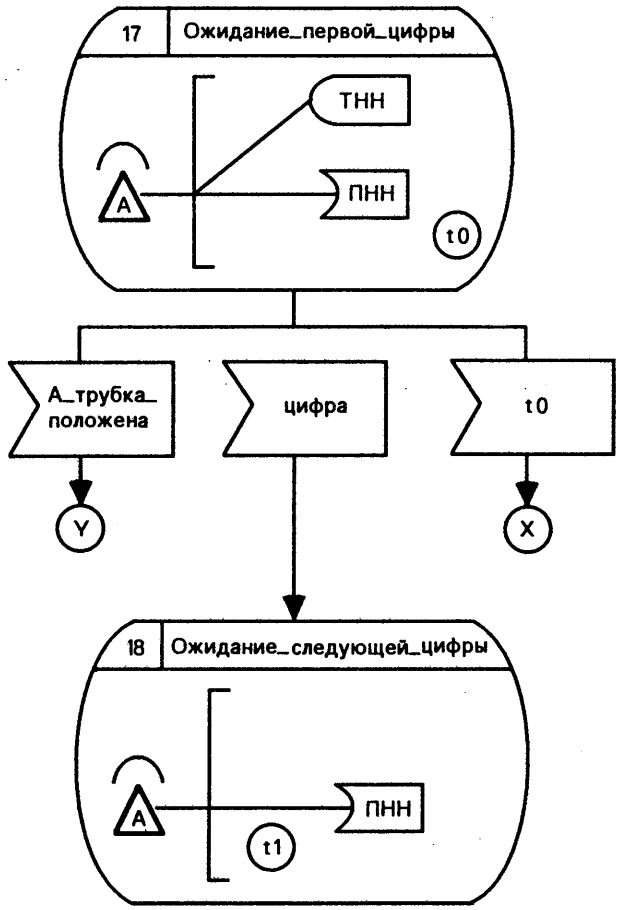
a) Повторный запуск таймера после истечения его срока

b) Повторный запуск таймера, срок которого еще не истек

Примечание. – Каждый таймер t1 имеет два взаимоисключающих состояния: активное и неактивное.

РИСУНОК D-5.4.4

Повторный запуск таймера



Примечание. – Таймер t0 управляет поступлением первой цифры, после чего тон набора номера устраняется из вызова и таймер t0 останавливается. Таймер t1 продолжает управлять поступлением достаточного числа цифр, обеспечивающих возможность маршрутизации вызова.

РИСУНОК D-5.4.5

Пример использования двух управляющих таймеров в одном и том же состоянии

D.5.5 *Вспомогательные диаграммы*

Чтобы облегчить чтение и понимание больших и/или сложных диаграмм процесса, автор может использовать неформальные вспомогательные диаграммы. Назначение таких документов – предоставить обзор или упрощенное описание поведения процесса (или его части). Вспомогательные диаграммы не заменяют документы на SDL, а образуют введение к ним.

В настоящем разделе приводятся примеры общеизвестных вспомогательных диаграмм, а именно диаграмм обзора состояний, матриц состояний/сигналов и временных схем. (В равной мере вспомогательной диаграммой является диаграмма дерева блоков, описанная в § D.4.4.)

D.5.5.1 *Диаграмма обзора состояний*

Диаграмма обзора состояний имеет целью дать представление о состояниях процесса и переходах, возможных между этими состояниями. Поскольку целью является дать всего лишь обзор, "менее важные" состояния и переходы могут быть опущены.

Диаграммы составляются из символов состояний, ориентированных дуг, изображающих переходы, и необязательных символов старта и стопа.

Символ состояния должен содержать имя, к которому отсылает данное состояние. Символ может содержать несколько имен состояний, а звездочка (*) может быть использована в качестве нотации всех состояний.

С каждой ориентированной дугой должны быть связаны имя сигнала или набор сигналов, вызывающих данный переход, а также имена возможных выводов, посылаемых в течение данного перехода.

Пример диаграммы обзора состояний приведен на рис. D-5.5.1.

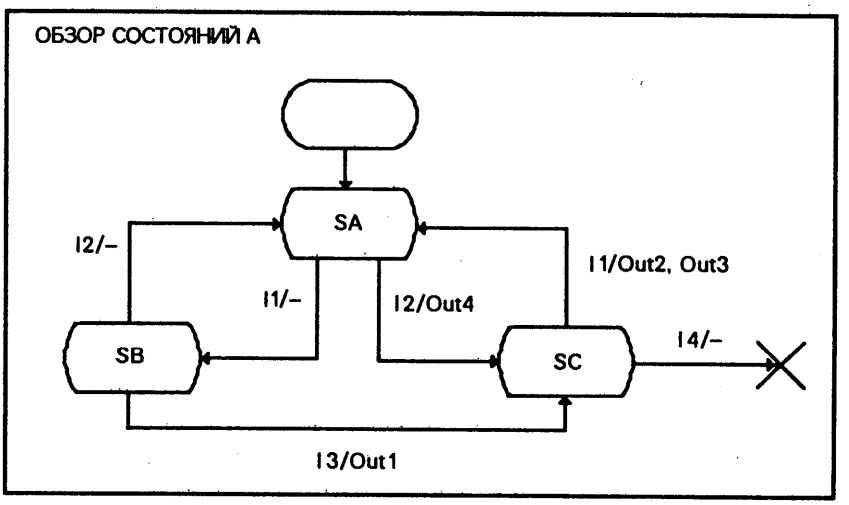


РИСУНОК D-5.5.1

Пример диаграммы обзора состояний

Диаграмма обзора состояний какого-либо процесса может быть разделена на несколько отдельных диаграмм, каждая из которых посвящена отдельным аспектам процесса, например "нормальный случай", "обработка_ошибок" и т.д.

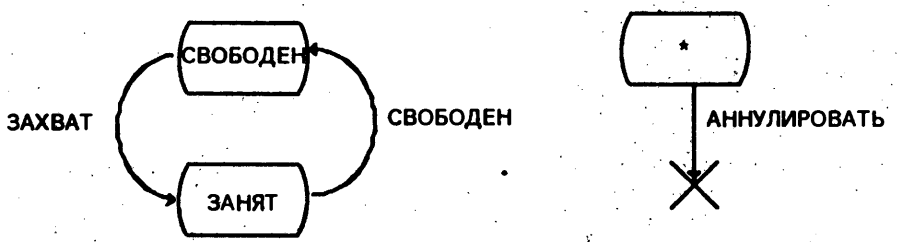


РИСУНОК D-5.5.2

Пример расчлененной диаграммы обзора состояний

D.5.5.2 *Матрица состояний/сигналов*

Матрица состояний/сигналов является альтернативной диаграмме обзора состояний, содержащей точно такую же информацию.

Диаграмма является двумерной матрицей, индексированной по одной оси всеми состояниями процесса, а по другой – всеми действительными входными сигналами данного процесса. Каждый элемент матрицы содержит следующее состояние совместно с возможными выводами в течение перехода. Может быть приведена ссылка на то место, где можно найти комбинацию, если таковая существует, соответствующую индексам.

Элемент, соответствующий фиктивному состоянию "START" и фиктивному сигналу "CREATE", используется для того, чтобы указать, которое из состояний является начальным.

СОСТОЯНИЕ/СИГНАЛ A

СОСТОЯНИЕ / СИГНАЛ	"СТАРТ"	SA	SB	SC
"СОЗДАНИЕ"	SA/–	–	–	–
I1	–	SB/–	–	SA/Out2,Out3
I2	–	SC/Out4	SA/–	–
I3	–	–	SC/Out 1	–
I4	–	–	–	"STOP" /–

РИСУНОК D-5.5.3

Пример матрицы состояний/сигналов

Матрица может быть разделена на подчасти, находящиеся на различных страницах. Ссылки являются обычными ссылками, используемыми пользователями в документации.

Желательно группировать состояния и сигналы таким образом, чтобы каждая часть матрицы соответствовала некоторому аспекту поведения процесса.

D.5.5.3 *Временна́я схема*

Можно предоставить временные схемы, показывающие допустимые последовательности сигналов, обмениваемых между сервисами, процессами, блоками и их окружающей средой.

Назначение этих схем – дать возможность обозреть и оценить сигналы, обмениваемые между частями системы. В зависимости от того, какие аспекты системы требуется высветить, может быть представлен либо полный обмен сигналами, либо только некоторые его части.

Столбцы схемы обозначают компоненты, осуществляющие связь (сервисы, процессы, блоки или окружающая среда).

Их взаимодействие изображается набором ориентированных линий (стрелок), причем каждая из них представляет один какой-то сигнал или набор сигналов.

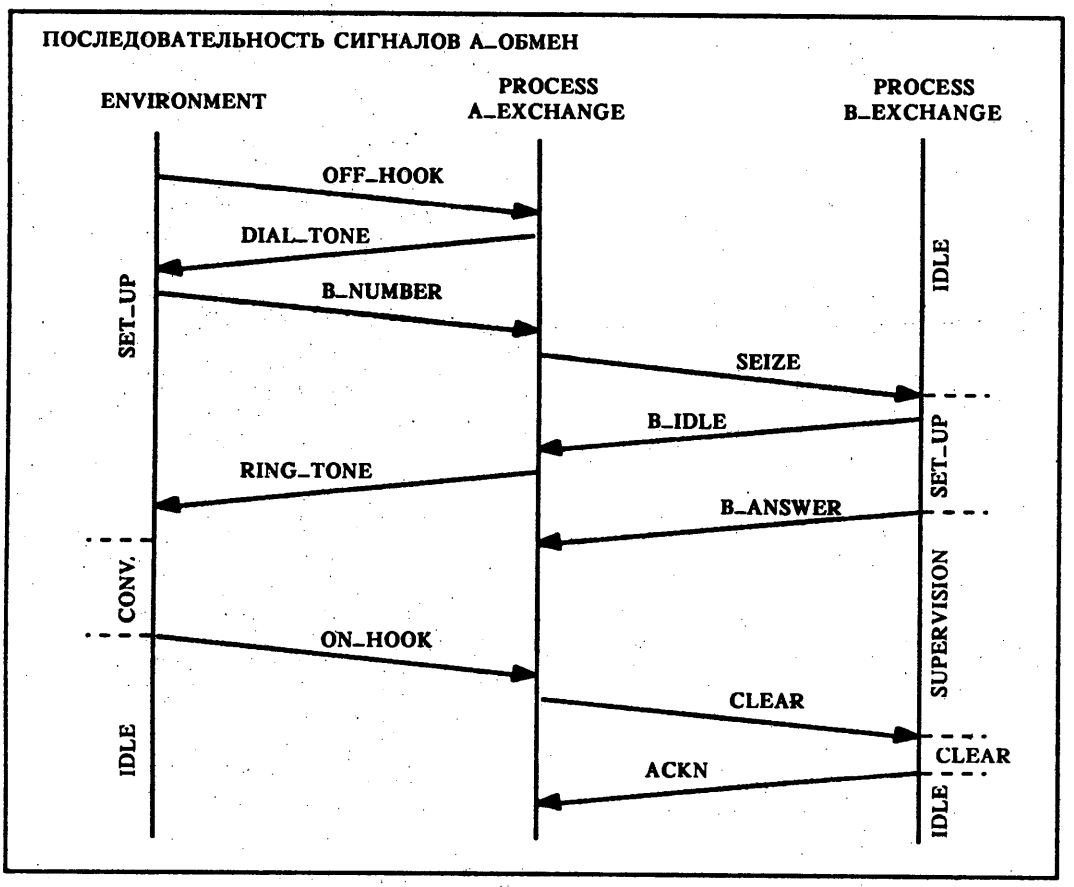


РИСУНОК D-5.5.4

Пример временно́й схемы

Каждой схеме может быть придана аннотация, поясняющая, какой набор информации обменивается здесь. Такая аннотация придается каждой линии, объясняя связанную с ней информацию (имена сигналов или процедуры).

На столбцах схемы можно помещать символы принятия решений, показывая тем самым, что идущая после символа последовательность сигналов имеет место только в том случае, если указанное условие истинно. Обычно в таких случаях символ принятия решения входит несколько раз, задавая различные последовательности, соответствующие различным значениям условия.

Такие диаграммы могут содержать либо полностью все последовательности обмениваемых сигналов, либо их наиболее значимую часть.

Полезным применением таких диаграмм является представление с помощью двустороннего взаимодействия сервисов, возникающего из расчленения процесса.

Обычно временные схемы не охватывают все возможные последовательности; очень часто они лишь предваряют полное определение.

D.6 *Определение данных в SDL*

D.6.1 *Руководство по данным в SDL*

В этом разделе дается дополнительная информация о концепциях, определенных в § 5 Рекомендации по SDL. Основное отличие настоящего раздела от прежней Рекомендации Z.104 состоит в том, что в нем достигнута бóльшая ясность и согласованность с требованиями ISO. Часть ключевых слов изменена, и сделаны некоторые добавления, но семантика та же, что и в Красной книге. Использование данных в § 2, § 3 и § 4 новой рекомендации (бывших Z.101 – Z.103) остается без изменений!

D.6.1.1 *Общее введение*

Подход к типам данных в SDL основан на 'абстрактном типе данных'. При этом подходе описывается не то, КАК должен быть реализован тип данных, но лишь то, КАКИМ будет результат применения операторов к значениям.

Когда определяются абстрактные данные, каждая часть определения, называемая "частичным определением типа", вводится ключевым словом NEWTYPE. Каждое частичное определение типа действует на остальные так, что все частичные определения типа на уровне системы составляют единое определение типа данных. Если на более низком уровне (например, на уровне блока) вводятся новые частичные определения типа, то они образуют вместе с определениями более высокого уровня одно определение типа данных. Это означает, что в каждой точке спецификации имеется ровно одно определение типа данных.

По существу, определение типа данных состоит из трех частей:

a) определений Сортов,

b) определений Операторов,

c) Равенств.

Каждая из этих частей разъясняется ниже. Определение типа данных разбито на частичные определения типа данных, каждое из которых вводит один сорт. Определения операторов и равенства разбросаны по частичным определениям типа данных.

D.6.1.2 *Сорта*

Сорт – это множество значений. Это множество может иметь как конечное, так и бесконечное число элементов, но не может быть пустым.

Примеры:

a) Множеством значений сорта Булевский является { Истина, Ложь } .

b) Множеством значений Натурального сорта является бесконечное множество натуральных чисел { 0, 1, 2, ... } .

c) Множеством значений сорта Основной_цвет является { Зеленый, Красный, Синий } .

Пользователь не должен задавать непосредственно каждый элемент сорта (в случае натуральных чисел это заняло бы неопределенное время), но имя сорта должно быть приведено. В конкретном синтаксисе непосредственно за ключевым словом NEWTYPE следует имя сорта (некоторые другие возможности будут рассмотрены в дальнейшем). Это имя используется в основном в определениях операторов, как это объясняется в § D.6.1.3, и в объявлениях переменных.

D.6.1.3 *Операторы, литералы и термы*

Значения сорта могут быть определены тремя способами:

a) перечислением; значения определяются в разделе литералов,

b) операторами; значения образуются как результат 'применения' операторов,

c) комбинацией перечисления и операторов.

Комбинация литералов и операторов дает термы. Соотношения между термами выражаются посредством равенств. В следующих разделах рассматриваются литералы, операторы и термы; в § D.6.1.4 рассматриваются равенства.

D.6.1.3.1 *Литералы*

Литералы – это перечисленные значения сорта. Частичное определение типа не обязательно содержит литералы; все элементы сорта могут быть определены с помощью операторов. Литералы можно рассматривать как операторы без аргументов. Соотношения между литералами могут быть выражены равенствами. В конкретном синтаксисе литералы вводятся после ключевого слова LITERALS.

Примеры:

a) Определение Булевского сорта содержит два литерала, а именно Истина и Ложь. Так что определение булевского типа выглядит следующим образом:

```
NEWTYPE Булевский
    LITERALS Истина, Ложь
    ...
ENDNEWTYPE Булевский;
```

b) Натуральный сорт может быть определен с использованием одного литерала, нуля. Прочие значения могут быть порождены с помощью этого литерала и операторов;

c) Значения сорта Основной_цвет могут быть определены аналогичными булевскими литералами:

```
NEWTYPE Основной_цвет
    LITERALS Красный, Зеленый, Синий
    ...
ENDNEWTYPE Основной_цвет;
```

d) В § D.6.1.3.2 в третьем примере (c) приводится частичное определение типа, не содержащее литералов.

D.6.1.3.2 *Операторы*

Оператор – это математическая функция, отображающая одно или несколько значений (возможно, различных сортов) в результирующее значение. Операторы вводятся после ключевого слова OPERATORS, для каждого оператора указываются сорта аргументов и сорт результата (это называется сигнатурой оператора).

Примеры:

a) В булевском типе может быть определен оператор "не", имеющий единственный аргумент Булевского сорта и результат также Булевского сорта. В определении булевского типа он приводится следующим образом:

```
NEWTYPE Булевский
    LITERALS Истина, Ложь
    OPERATORS "Не": Булевский => Булевский;
    ...
ENDNEWTYPE Булевский;
```

b) Упомянутый в предыдущем разделе оператор, требуемый для построения всех натуральных чисел, – это оператор Следующий. Этот оператор отображает аргумент натурального сорта в значение натурального сорта (следующее по величине за значением аргумента), которое и будет результатом.

c) Для цветов может быть определен новый тип, не имеющий литералов, но использующий литералы сорта Основной_цвет и некоторые операторы:

```
NEWTYPE Цвет
    OPERATORS
        Взять: Основной_цвет => Цвет;
        Смешать: Основной_цвет, Цвет => Цвет;
    ...
ENDNEWTYPE Цвет;
```

Подразумевается, что берется основной цвет и рассматривается просто как цвет, а затем с ним смешиваются основные цвета для получения других цветов.

D.6.1.3.3 *Термы*

Используя литералы и операторы, для каждого сорта может быть построено множество термов следующим образом:

1. Собрать все литералы данного сорта во множество, отвечающее этому сорту – каждый литерал является термом.

2. Для каждого оператора строится новое множество термов применением этого оператора ко всем комбинациям термов допустимых сортов из уже построенных множеств.

   a) Для Булевского сорта множеством литералов является { Истина, Ложь } . Результатом данного шага будет { Не (Истина) , Не (Ложь) } , поскольку у нас имеется только оператор Не.

   b) Для Натурального сорта результатом данного шага является { Следующий (0) } .

   c) Для сорта Цвет множество литералов пусто, но результатом данного шага является { Взять (Красный) , Взять (Зеленый) , Взять (Синий) } .

3. Термы, принадлежащие множествам, построенным на предыдущем шаге, имеют сорт результата применявшегося оператора, например, все результаты оператора Не булевского сорта. Теперь берутся объединения всех множеств одного сорта, как исходных, так и только что построенных.

   a) Для Булевского сорта такое объединение приводит ко множеству { Истина, Ложь, Не (Истина) , Не (Ложь) } .

   b) Для натуральных чисел на этом шаге получается множество { 0, Следующий (0) } .

4. Последние два шага повторяются вновь и вновь, определяя в общем случае бесконечное множество термов.

   a) Множеством булевских термов, порожденных литералами Истина и Ложь и оператором Не, является { Истина, Ложь, Не (Истина) , Не (Ложь) , Не (Не (Истина)) , Не (Не (Ложь)) , Не (Не (Не (Истина))) , ...} .

b) Множеством натуральных термов, порожденных литералом 0 и оператором Следующий, является { 0, Следующий(0), Следующий(Следующий(0)), Следующий(Следующий(Следующий(0))), ... }.

c) Множеством цветных термов, порожденных литералами Красный, Зеленый и Синий сорта Основной_цвет и операторами Взять и Смешать, является { Взять(Красный), Взять(Зеленый), Взять(Синий), Смешать(Красный, Взять(Красный)), Смешать(Красный, Взять(Зеленый)), Смешать(Красный, Взять(Синий)), Смешать(Зеленый, Взять(Красный)), Смешать(Зеленый, Взять(Зеленый)), Смешать(Зеленый, Взять(Синий)), Смешать(Синий, Взять(Красный)), Смешать(Синий, Взять(Зеленый)), Смешать(Синий, Взять(Синий)), ... } .

D.6.1.4 *Равенства и аксиомы*

В общем случае число термов, порожденных как в предыдущем разделе, больше числа значений сорта. Например, имеется два булевских значения, но множество булевских термов имеет бесконечное число элементов. Имеется, однако, возможность задать правила, по которым устанавливается, какие термы считать обозначающими одинаковые значения. Эти правила называются равенствами, и они разъясняются в следующем подразделе. Два специальных вида равенств, аксиомы и условные равенства, трактуются в § D.6.1.4.2 и § D.6.1.4.3.

В конкретном синтаксисе все равенства, аксиомы и условные равенства даются после ключевого слова AXIOMS. Это ключевое слово сохраняется по историческим причинам.

D.6.1.4.1 *Равенства*

Равенство устанавливает, какие термы рассматриваются как обозначающие одно и то же значение. Равенство связывает два терма, между которыми ставится символ-эквивалентности "= =".

Например, "Не(Истина) = = Ложь" устанавливает, что термы Не(Истина) и Ложь эквивалентны; где бы ни было расположено Не(Истина), вместо него можно подставить Ложь, не изменяя смысла, и наоборот.

Если заданы некоторые равенства, множество термов разбивается на непересекающиеся подмножества термов, обозначающих одно и то же значение. Эти подмножества называются классами эквивалентности. В обыденной речи классы эквивалентности отождествляются со значениями.

Примеры:

a) Множество термов Булевского сорта разбито на два класса эквивалентности двумя следующими аксиомами:

Не(Истина) = = Ложь;
Не(Ложь) = = Истина;

В результате получаются следующие классы эквивалентности:

{ Истина, Не(Ложь), Не(Не(Истина)), Не(Не(Не(Ложь))),
Не(Не(Не(Не(Истина)))), Не(Не(Не(Не(Не(Ложь))))), ...}
и

{ Ложь, Не(Истина), Не(Не(Ложь)), Не(Не(Не(Истина))),
Не(Не(Не(Не(Ложь)))), Не(Не(Не(Не(Не(Истина))))), ... } .

Эти два класса эквивалентности отождествляются со значениями Истина и Ложь.

b) В случае цветов хочется потребовать, чтобы при смешивании основного цвета с цветом, содержащим этот основной цвет, получался бы тот же самый (составной) цвет. Кроме того, порядок, в котором смешиваются основные цвета, должен быть несущественным. Это может быть задано равенствами следующим образом:

```
Смешать(Красный, Взять(Красный))                    = = Взять(Красный);
Смешать(Красный, Смешать(Красный, Взять(Зеленый))) = = Смешать(Красный, Взять(Зеленый));
Смешать(Красный, Смешать(Красный, Взять(Синий)))   = = Смешать(Красный, Взять(Синий));
Смешать(Красный, Смешать(Зеленый, Взять(Красный))) = = Смешать(Зеленый, Взять(Красный));
Смешать(Красный, Смешать(Синий, Взять(Красный)))   = = Смешать(Синий, Взять(Красный));
                                                      и так далее.
```

Это требует большой работы, поскольку для всех перестановок Красного, Зеленого и Синего возникают равенства одного и того же вида. Поэтому в SDL имеется конструкция-FOR-ALL, в которой вводятся имена значений, используемые вместо произвольных классов эквивалентности (или значений, ассоциированных с этими классами эквивалентности). В вышеприведенной ситуации она может быть весьма полезной; все уже выписанные равенства, равно как и обозначенные "и так далее", могут быть записаны в несколько строк следующим образом:

```
FOR ALL p1, p2 IN Основной_цвет
/*1*/  (  Смешать(p1, Взять(p1)) = = Взять(p1);
/*2*/     Смешать(p1, Смешать(p1, Взять(p2))) = = Смешать(p1, Взять(p2));
/*3*/     Смешать(p1, Смешать(p2, Взять(p2))) = = Смешать(p2, Взять(p1));
/*4*/     Смешать(p1, Взять(p2)) = = Смешать(p2, Взять(p1));
          FOR ALL c IN Цвет
/*5*/     ( Смешать(p1, Смешать(p2, c)) = = Смешать(p2, Смешать(p1, c));
/*6*/       Смешать(p1, Смешать(p2, c)) = = Смешать(Смешать(p1, Взять(p2)), c))
       )
```

Приведенные выше равенства перекрывают друг друга, но это не приводит к затруднениям, поскольку эти равенства не противоречат друг другу.

Приведенные выше равенства порождают 7 классов эквивалентности во множестве термов сорта Цвет, так что вместе с этими равенствами получаются 7 значений-цветов. Следующие термы лежат в различных классах эквивалентности:

```
Взять(Красный), Взять(Зеленый), Взять(Синий),
Смешать(Красный, Взять(Зеленый)),
Смешать(Зеленый, Взять(Синий)),
Смешать(Синий, Взять(Красный)),
Смешать(Синий, Смешать(Зеленый, Взять(Красный))).
```

Все прочие термы сорта Цвет эквивалентны одному из вышеприведенных термов.

В примерах равенств с конструкцией-FOR-ALL, так называемых явно квантифицированных равенствах, информация о том, что p1 и p2 являются значениями сорта Основной_цвет, избыточна; аргумент оператора Взять и первый аргумент оператора Смешать могут быть только лишь сорта Основной_цвет. Вообще, явно квантифицированные равенства более удобны для прочтения, но квантификацию можно опустить, если сорт идентификаторов значений может быть выведен из контекста. В этом случае равенства называются неявно квантифицированными.

Пример:
Равенства 4 и 5, приведенные выше, можно записать так:

```
Смешать(p1, Взять(p2))      = = Смешать(p2, Взять(p1));
Смешать(p1, Смешать(p2, c)) = = Смешать(p2, Смешать(p1, c));
```

D.6.1.4.2 *Аксиомы*

Аксиомы – это просто частный вид равенств, введенный по той причине, что многие равенства в конкретных случаях связаны с булевским сортом. В этом случае равенства, как правило, имеют вид ... = = Истина; то есть они устанавливают, что некоторый терм эквивалентен значению Истина.

Пример:
Предположим, что для типа цвет определен оператор: Содержит: Цвет, Основной_цвет –> Булевский; принимающий значение Истина, если (составной) цвет содержит основной цвет, и Ложь в противном случае. С ним будут связаны некоторые равенства, например:

```
FOR ALL p IN Основной_цвет
(    Содержит(Взять(p), p)              = = Истина;
     FOR ALL c IN Цвет
     (    Содержит(Смешать(p, c), p) = = Истина)
)
```

Часть '= = Истина' в этих равенствах может быть опущена, и то, что получится, называется аксиомой. Аксиомы можно узнать по отсутствию символа-эквивалентности "= =", и они обозначают термы, эквивалентные значению Истина Булевского сорта.

Конструкция второго равенства может показаться несколько неестественной. После введения некоторых полезных конструкций будет показан лучший способ записи таких равенств.

D.6.1.4.3 *Условные равенства*

Условные равенства – это средство для записи уравнений, справедливых только при выполнении некоторых условий. Условия записываются с использованием того же синтаксиса, что и безусловные равенства. Они отделяются от равенства, справедливость которого зависит от выполнения условия, символом = =>.

Пример:
Стандартным примером условного равенства является определение деления для вещественного типа, где

```
FOR ALL x, z IN Вещественный
(    z/= 0 = = Истина          ==>      (x/z)* z = = x)
```

устанавливает, что если условие "z не равно 0" выполнено, то деление на z и последующее умножение на z приводит к исходному значению. Это условное равенство не утверждает ничего в случае, когда значение Вещественного сорта делится на ноль. Если требуется описать, что происходит в случае деления на ноль, нужно задать условное равенство вида

```
FOR ALL x, z IN Вещественный
(    z = 0 = = Истина          ==>      (x/z)* z = = ...)
```

Однако в таких случаях для удобства прочтения рекомендуется использовать в правой части равенства так называемый 'условный терм'. Для вышеприведенного случая равенство будет иметь вид:

```
FOR ALL x, z IN Вещественный
(    (x/z)*z                    = =  IF z/=0
                                     THEN x
                                     ELSE ...
                                     FI
)
```

D.6.1.5 *Еще о равенствах и аксиомах*

В двух следующих разделах разъясняются некоторые трудности, с которыми можно столкнуться, когда операторы принимают значения из уже определенного сорта. В § D.6.1.5.3 объясняется понятие ошибки как терма в равенстве.

D.6.1.5.1 *Согласованность с иерархией*

В каждой точке SDL спецификации имеется одно, и только одно, определение типа данных. Это определение типа данных содержит предварительно определенные сорта, операторы и равенства и все сорта, операторы и равенства, определенные пользователем в частичных определениях типа, доступных в данной точке. (Вот почему текст NEWTYPE... ENDNEWTYPE называется частичным определением типа.)

Это имеет некоторые последствия для определений типа на более низких уровнях. Они могут нежелательным образом воздействовать на тип. Например, можно ошибочно специфицировать два терма как эквивалентные, сделав их тем самым эквивалентными, тогда как они не эквивалентны в объемлющей части спецификации.

Недопустимо задавать уравнения таким образом, чтобы:

a) значения сорта, являющиеся различными в части спецификации более высокого уровня, становились эквивалентными;

b) к сорту, определенному в части спецификации более высокого уровня, добавлялись новые значения.

Это означает, например, что в блоке на уровне системы в специфицированных пользователем частичных определениях типа, содержащих оператор, результат которого имеет предварительно определенный сорт, всем термам, полученным с помощью этого оператора, должны отвечать значения сорта результата.

Примеры:

a) Предположим, что по какой-либо причине задается аксиома:

```
FOR ALL n, m IN Целый
(     (Факт(n) = Факт(m))   =>     (n = m)),
```

которая должна специфицировать, что если значения оператора Факт совпадают, то совпадают и аргументы. (Отметим, что => является символом булевской импликации, имеющим мало общего с символом условного равенства ==>.) Затем случайно эти значения объединяются. Тогда из равенств предыдущего примера может быть выведено, что Факт(0) = Факт(1), что в свою очередь будет означать, что 0 и 1 являются различными обозначениями одного значения. Из этого последнего равенства можно будет получить, что число элементов Целого сорта уменьшится до единицы.

При помощи условного равенства может быть выражено, что, если n и m отличны от нуля, совпадение результатов применения оператора Факт к n и m влечет n = m. В SDL:

```
FOR ALL n, m IN Целый
(     n/= 0, m/= 0 ==> (Факт(n) = Факт(m)) => (n = m)).
```

Отметим, что последнее равенство не добавляет ничего к семантике Целых; это теорема, которая может быть выведена из других равенств. С другой стороны, добавление доказуемого равенства не причинит вреда.

b) Предположим, что при определении некоторого типа возникла потребность в операторе, вычисляющем факториалы. В частичном определении типа для этого типа вводится оператор Факт:

```
Факт: Целый => Целый;
```

и для определения этого оператора даются следующие равенства:

```
Факт (0) = = 1;
FOR ALL n IN Целый
(     n > 0 = => Факт (n) = = n* Факт (n-1))
```

Приведенные равенства не определяют терм Факт(-1), так что он будет термом Целого сорта, никак не связанным с другими термами этого сорта. Поэтому Факт(-1) является новым значением Целого сорта (как и Факт(-2), Факт(-3), и т.д.). Это недопустимо. В примере b) § D.6.1.5.3 приведено корректное определение оператора Факт.

D.6.1.5.2 *Операторы равенства и неравенства*

В каждом типе неявно задаются операторы равенства и неравенства. Так, если частичное определение типа вводит сорт S, то в нем имеются неявные определения операторов:

```
"=": S, S –> Булевский;
"/=": S, S –> Булевский;
```

(Примечание.–Кавычки указывают, что = и/= используются как инфиксные операторы.)

Оператор равенства имеет ожидаемые свойства:

```
a = a,
a = b => b = a,
a = b AND b = c => a = c,
a = b => a = = b,
a = b => op(a) = op(b) для операторов op.
```

Вышеприведенные свойства выписаны без использования синтаксиса SDL и не должны задаваться в аксиомах или равенствах, поскольку они неявно подразумеваются. Булевское значение, принимаемое этим оператором, есть Истина, если термы в левой и правой частях лежат в одном классе эквивалентности, а в противном случае его значением будет Ложь. Если же явно не специфицируется, что значением будет Истина или Ложь, то спецификация является неполной.

Семантика оператора неравенства лучше всего может быть объяснена с помощью SDL-равенства:

```
FOR ALL a, b IN S
(     a/= b = = Не (a = b)).
```

Между равенством и эквивалентностью нет никакой разницы. Два эквивалентных терма обозначают одно и то же значение, и оператор равенства между ними принимает значение Истина.

D.6.1.5.3 *Ошибка*

В предыдущих примерах ощущалась необходимость специфицировать случаи, когда применение операторов к некоторым значениям рассматривается как ошибка. В SDL имеется средство для такой формальной спецификации: Ошибка. Ошибка должна использоваться для выражения того, что

> "применение данного оператора к данному значению недопустимо, и если это произошло, дальнейшее поведение системы не определено."

В конкретном синтаксисе это обозначается термом Ошибка!, который не может быть использован как аргумент оператора.

Когда Ошибка является результатом применения оператора, к которому применен еще один оператор, то результатом внешнего оператора также будет Ошибка (размножение-ошибок). В условном терме вычисляется THEN-часть, либо ELSE-часть, поэтому одна из них может быть Ошибкой, которая не будет вычисляться (поскольку вычисляется другая альтернатива).

Примеры:

a) В примере деления значений Вещественного сорта многоточие может быть заполнено:

```
FOR ALL x, z IN Вещественный
(     (x/z)*z == IF z/= 0
                 THEN   x
                 ELSE   Ошибка!
                 FI
)
```

Для большей ясности можно добавить:

```
FOR ALL x IN Вещественный
(     x/0 == Ошибка!)
```

b) В примере с оператором Факт можно специфицировать, что применение этого оператора к целым отрицательным числам приводит к Ошибке. Так можно избежать появления новых значений Целого сорта, Факт(-1), Факт(-2), ... Хорошее определение оператора Факт может быть следующим:

```
n < 0    ==> Факт(n) == Ошибка!;
             Факт(0) == 1;
n > 0    ==> Факт(n) == Факт(n-1)* n;
```

Эти три строки гораздо яснее, чем приведенное ниже равенство в духе программирования. Вообще условный терм должен использоваться, если рассматриваются два дополняющих друг друга случая; гнездовое включение условных термов неудобочитаемо, в чем можно убедиться из примера:

```
Факт(n)  ==  IF n > 0
             THEN Факт(n-1)* n
             ELSE    IF n = 0
                     THEN 1
                     ELSE Ошибка!
                     FI
             FI
```

D.6.2 *Генераторы и наследование*

В этом разделе рассматриваются две конструкции, которые могут использоваться для специфицирования типов, имеющих в описании общие части. Генератор специфицирует не тип, а схему, превращающуюся в тип, если формальные сорта, операторы, литералы и константы заменяются на фактические.

Наследование предлагает возможность породить новый тип, исходя из уже существующего типа. Литералы и операторы могут быть переименованы, и кроме того, могут быть специфицированы дополнительные литералы, операторы и равенства.

D.6.2.1 *Генераторы*

Определение генератора определяет схему с входящими в нее в качестве параметров формальными именами сортов, литералов, констант и операторов. Генераторы предназначены для определения типов, являющихся 'вариациями на тему', таких как множества элементов, строки элементов, файлы записей, справочные таблицы, массивы.

Это будет объяснено на примере, в котором могут быть предусмотрены вариации. Предположим, возникла потребность в типе, подобном математическому понятию множества целых чисел. Нижеследующий текст является частью определения этого типа Множество_целых.

```
NEWTYPE Множество_целых
    LITERALS пустое_множество_целых
    OPERATORS
        Добавить: Множество_целых,Целые->Множество_целых;
        Изъять: Множество_целых,Целый->Множество_целых;
        Содержится: Множество_целых,Целый->Булевский
    AXIOMS
    /*1*/ Изъять(пустое_множество_целых,int)
                            == пустое_множество_целых;
    /*2*/ Содержится(set,int1), = ложь ==>
          Изъять(Добавить(set, int1), int2)
                            == IF int1 = int2
                               THEN set
                               ELSE Добавить (Изъять(set, int2), int1)
                               FI;
    /*3*/ Содержится(пустое_множество_целых, int)
                            == Ложь;
    /*4*/ Содержится(Добавить(set, int1), int2)
                            == int1 = int2 OR Содержится(set, int2);
    /*5*/ Добавить (Добавить(set, int1), int2)
                            == IF int1 = int2
                               THEN Добавить(set, int1)
                               ELSE Добавить (Добавить(set, int2), int1)
                               FI
ENDNEWTYPE Множество_целых;
```

РИСУНОК D-6.2.1

Новый тип Множество_целых

Во всех равенствах квантификация неявная. Первое равенство выражает, что изъятие элемента из пустого множества приводит в результате к пустому множеству. Второе равенство показывает, что изъятие элемента после того, как он был добавлен, приводит к исходному множеству при условии, что множество не содержало этого элемента, а в противном случае добавление и изъятие можно переставить местами. Третье равенство выражает, что пустое множество не содержит элементов. Четвертое равенство означает, что элемент будет содержаться во множестве, если этот элемент был добавлен к множеству последним, либо если он принадлежал множеству до того как к тому был добавлен последний элемент. Последнее равенство выражает, что порядок добавления элементов к множеству не имеет существенного значения.

В примере на рис D-6.2.1 Множество_целых является лишь примером типа, описывающего множества. И если в одной спецификации потребуются одновременно Множество_PId, Множество_абонентов и Множество_имен_коммутаторов, будет неудивительно, если все они будут содержать операторы: Добавить, Изъять и Содержится и литерал, обозначающий пустое множество. Равенства, заданные для этих операторов, можно легко обобщить для других множеств.

В этом случае окажется полезной концепция генератора, согласно которой общий текст задается один раз, а затем многократно используется. На рис. D-6.2.2 приведен генератор. (Отметим, что имена формальных сортов вводятся ключевым словом TYPE. Это делается лишь по историческим причинам.)

```
GENERATOR Множество (TYPE Элемент, LITERAL пустое_множество)
   LITERALS пустое_множество
   OPERATORS
      Добавить: Множество, Элемент->Множество;
      Изъять: Множество, Элемент->Множество;
      Содержится: Множество, Элемент->Булевский
   AXIOMS
   /*1*/ Изъять (пустое_множество, itm)
                        == пустое_множество;
   /*2*/ Содержится (st, itm1 ) = ложь ==>
         Изъять(Добавить(st, itm2), itm2)
                        == IF itm1=itm2
                           THEN st
                           ELSE Добавить (Изъять (st, itm2), itm1)
                           FI;
   /*3*/ Содежится (пустое_множество, itm)
                        == Ложь;
   /*4*/ Содержится (Добавить(st,itm1), itm2)
                        == itm1 = itm2 OR Содержится(st, itm2);
   /*5*/ Добавить(st, itm1), itm2)
                        == IF itm1 = itm2
                           THEN Добавить (st, itm1)
                           ELSE Добавить (Добавить(st, itm2), itm1)
                           FI
ENDGENERATOR Множество;
```

РИСУНОК D-6.2.2

Генератор Множество

Вместо Целых используется формальный тип Элемент. Для того чтобы в разных типах литерал, обозначающий пустое множество, мог иметь различные имена, этот литерал также сделан формальным параметром.

Используя этот генератор, тип Множество_целых может быть построен следующим образом:

```
NEWTYPE Множество_целых Множество (Целый, пустое_множество_целых)
ENDNEWTYPE Множество_целых;
```

Сравнивая рис. D-6.2.1 и рис. D-6.2.2, можно увидеть, что

a) GENERATOR и ENDGENERATOR заменяются соответственно на NEWTYPE и ENDNEWTYPE,

b) формальные параметры генератора (то есть текст в скобках после имени генератора) опускаются,

c) всюду в генераторе Множество, Элемент и пустое_множество заменяются соответственно на Множество_целых, Целый и пустое_множество_целых.

Так что определенное таким образом Множество_целых ничем не отличается от приведенного на рис. D-6.2.1, однако...

— если требуется определить множество PId-значений, тип может быть создан таким образом:

```
NEWTYPE Множество_PId Множество (PId, пустое_ множество_pid)
ENDNEWTYPE Множество_PId;
```

— если нужно определить множество абонентов, где абоненты представляются типом, который вводит сорт Абонент, множество абонентов может быть создано таким образом:

```
NEWTYPE Множество_абонент Множество (Абонент, пустое_множество_абонент)
ENDNEWTYPE Множество_абонент;
```

Это не только экономит бумагу, но и облегчает жизнь, поскольку понятие множества придется продумывать всего один раз, либо эту работу можно поручить квалифицированному специалисту по абстрактным типам данных.

Пример:

В данном примере приведен генератор, использующий формальный сорт, оператор, литерал и константу. Он описывает кортеж элементов с максимальной длиной макс_длина. Сорт содержит один литерал для обозначения пустого кортежа и операторы для вставки элементов в кортеж, их удаления, сцепления кортежей, выделения подкортежа и определения длины кортежа. Последний из этих операторов сделан формальным с тем, чтобы его можно было переименовывать.

```
GENERATOR Кортеж (TYPE Элемент, OPERATOR Длина, LITERAL Пустой,
                                  CONSTANT макс_длина)
    LITERAL Пустой
    OPERATORS
        Длина: Кортеж                    -> Целый;
        Вставить: Кортеж, Элемент, Целый -> Кортеж;
        Удалить: Кортеж, Целый, Целый    -> Кортеж;
        "//": Кортеж, Кортеж             -> Кортеж;
        Выделить: Кортеж, Целый, Целый   -> Кортеж;
        /* и прочие операторы, относящиеся к кортежам элементов*/
    AXIOMS
        /* Равенства для вышеприведенных операторов,*
        * в том числе следующие два (или им эквивалентные)*/
        Длина (г) = макс_длина ==> Вставить (r, itm, int) == > Ошибка!;
        Длина (г1) + Длина (г2) > макс_длина == > г1//г2 = = Ошибка!
ENDGENERATOR Кортеж;
```

Отметим, что формальные оператор Длина и литерал Пустой вновь заданы в теле генератора еще раз, поскольку при реализации они будут переименованы. Для оператора сорта его аргументов и результата даны только в теле генератора. Генератор Кортеж может быть использован для создания строк, страниц и книг следующим образом:

```
NEWTYPE Строка Кортеж (Знаковый, Ширина, Пустая_строка, 80)
ENDNEWTYPE Строка;

NEWTYPE Страница Кортеж (Строка, Длина, Пустая_страница, 66)
ENDNEWTYPE Строка;

NEWTYPE Книга Кортеж (Страница, Число_страниц, Пустая_книга, 10000)
ENDNEWTYPE Книга;
```

D.6.2.2 *Наследование*

Наследование – это средство, позволяющее получить все значения так называемого родительского сорта, некоторые или все операторы родительского типа и все равенства родительского типа. Как для литералов, так и для операторов имеется возможность их переименовывать. Вообще это является хорошей практикой, поскольку в данном случае читатель может вывести из контекста, что он имеет дело с другим типом, даже если литералы остались теми же.

Если оператор не наследуется, он систематически переименовывается, получая имя, недоступное пользователю. То обстоятельство, что эти операторы, тем не менее, присутствуют, означает, что все равенства родительского типа все же присутствуют (с переименованными операторами). Этим гарантируется, что все родительские значения будут унаследованы.

Наряду с возможностью не допускать использование оператора (не наследуя его) обеспечивается также возможность добавлять новые операторы. Литералы, операторы и равенства могут задаваться после ключевого слова ADDING так же, как и в обычном типе. Следует, однако, быть очень осторожным с новыми литералами и с взаимным воздействием унаследованных и добавленных операторов.

При добавлении новых литералов должен быть определен результат применения унаследованных операторов к этим новым литералам (посредством равенств). При добавлении операторов следует помнить об операторах, переименованных в недоступные имена, и ассоциированных с ними равенствах. Равенства, определяющие добавленные операторы, должны быть совместимы с равенствами, содержащими как унаследованные, так и не унаследованные операторы.

После этого списка предупреждений рассмотрим некоторые примеры.

а) Допустим, новый тип цвет задан полностью и доступен. Настоящий тип основан на том, что берутся и смешиваются световые лучи основных цветов. Пришлось бы долго думать, писать и/или переписывать, чтобы определить нечто подобное для того, как берутся и смешиваются краски.

У этой задачи есть красивое решение, состоящее в превращении нового типа цвет в генератор всего лишь двумя заменами:

1) первая строка
```
NEWTYPE Цвет
```
заменяется на
```
GENERATOR Цвет (TYPE Основной_цвет)
```

2) ключевое слово ENDNEWTYPE заменяется на ENDGENERATOR.

Теперь при реализации генератор может быть переименован. Предположим, прежний сорт Основной_цвет назван Основной_света, а сорт Основной_краски определен как

```
NEWTYPE Основной_краски
    LITERALS Красный, Желтый, Синий
ENDNEWTYPE Основной_краски;
```

Теперь уже очень легко определить два подобных типа, один для света, а другой для красок:

```
NEWTYPE Цвета_света Цвет(Основной_света) ENDNEWTYPE;
NEWTYPE Цвета_краски Цвет(Основной_краски) ENDNEWTYPE;
```

До сих пор проблем не возникало, однако как же теперь отличить терм Взять(Красный) типа Цвета_света от имеющего тот же синтаксис терма типа Цвета_краски? Если есть необходимость отличать их друг от друга, может помочь наследование. Вместо Цвета_света и Цвета_красок посредством наследования вводятся типы Свет и Палитра, в которых переименован оператор Взять:

```
NEWTYPE Свет
    INHERITS Цвета_света
    OPERATORS (Луч=Взять, Смешать, Содержит)
    ADDING
        LITERALS Белый
        AXIOMS
            Белый==Смешать(Красный, Смешать(Желтый, Луч (Синий)))
ENDNEWTYPE Свет;
```

Теперь новый тип Свет имеет литералы из типа Цвета_света плюс литерал Белый. У типа Цвета_света нет своих литералов (поскольку в нем используются литералы типа Основной_света), так что Белый – это единственный литерал типа Свет. Операторы и равенства типа Свет те же, что и в Цвета_света, за исключением того, что имя оператора Взять заменено на Луч и добавлено равенство для литерала Белый. Добавленная аксиома устанавливает, что этот добавленный литерал становится элементом множества термов, отвечающих случаю смешения всех трех основных цветов.

Новый тип Палитра имеет литералы из типа Цвета_краски и оператор Взять заменен в нем на оператор Краска:

```
NEWTYPE Палитра
    INHERITS Цвета_краски
    OPERATORS (Краска=Взять, Смешать, Содержит)
ENDNEWTYPE Палитра;
```

b) Предположим, потребовалось дополнить тип множества целых (сорт Множество_целых), введенный в предыдущем разделе, оператором, который находит наименьший элемент множества. В первую очередь нужно выяснить, можно ли ввести этот оператор в определение генератора, с тем чтобы его использовать также для множеств элементов другой природы. Если такое возможно, это значит, что в сорте Элемент определены > и <. Для произвольного сорта (например, для PId) это не так, и поэтому лучше определить новый тип с сортом Новое_множество_целых, задав в нем оператор Мин.

```
NEWTYPE Новое_множество_целых
    INHERITS Множество_целых
    OPERATORS ALL
    ADDING
        OPERATORS
            Мин:Новое_множество_целых -> Целый;
        AXIOMS
            Мин(Пустое_множество_целых)== Ошибка!;
            Мин (Добавить(Пустое_множество_целых, x))==x;
            Мин(Добавить(Добавить(nis, x), y))==
                                IF y < Мин(Добавить (nis, x))
                                THEN y
                                ELSE Мин(Добавить(nis, x))
                                FI
ENDNEWTYPE Новое_множество_целых;
```

Поскольку добавление после реализации генератора довольно общее, текст, начинающийся с ADDING и кончающийся перед ENDNEWTYPE, может быть задан в реализации генератора. Пример приводится в § 5.4.1.12 Рекомендации.

D.6.3 *Точки зрения на равенства*

При введении нового типа данных существенно, чтобы было задано достаточное количество равенств. Ниже приводятся три точки зрения на равенства, которые полезны при конструировании равенств.

D.6.3.1 *Общие требования*

Вне зависимости от того, как задаются равенства, должны соблюдаться следующие требования:

a) Каждый оператор появляется во множестве равенств хотя бы один раз (за исключением патологических случаев).

b) Все истинные предложения должны выводиться из равенств. Либо они задаются в виде аксиом, либо могут быть выведены подстановкой эквивалентных термов в равенства.

c) Не должна обнаруживаться никакая несовместимость, то есть требуется, чтобы из равенств нельзя было вывести, что Истина = Ложь.

На рис. D-6.3.1 приведена процедура нахождения равенств на неформальном SDL.

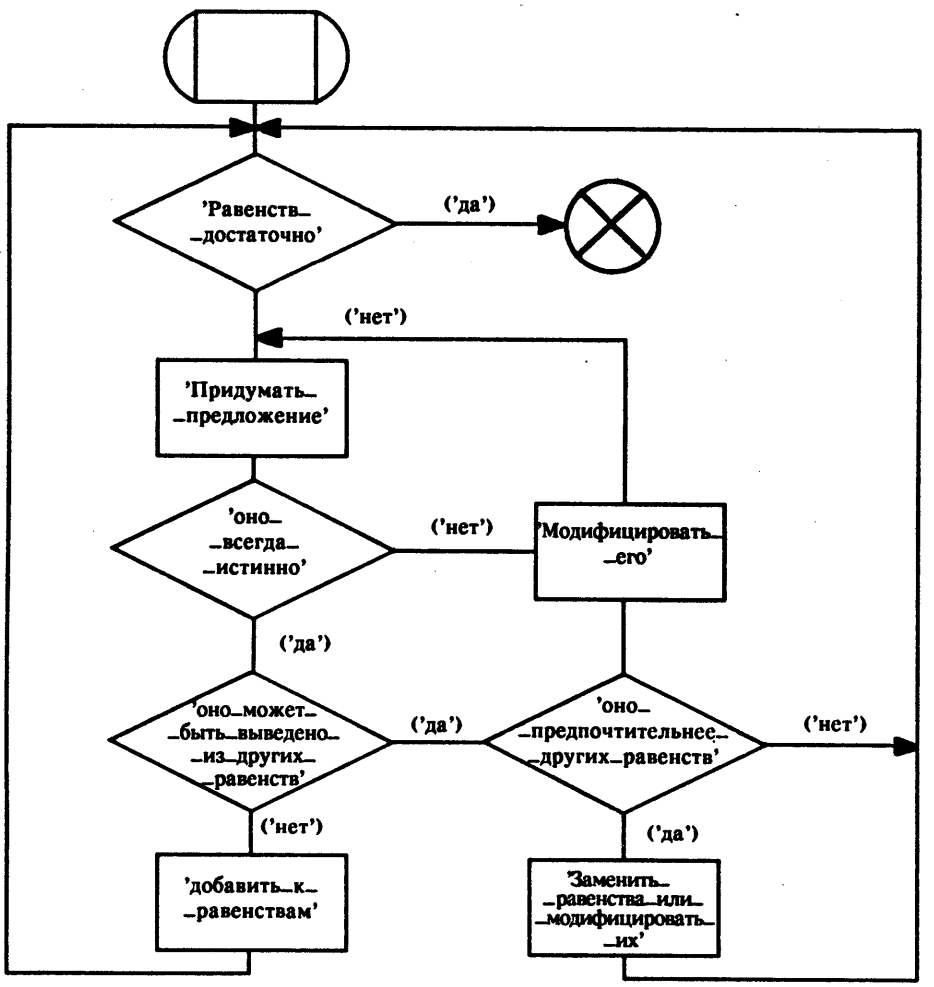


РИСУНОК D-6.3.1

Процедура нахождения равенств на неформальном SDL

D.6.3.2 *Применение функций к конструкторам*

Как правило, во множестве операторов выделяются подмножества операторов, называемых 'конструкторы' и 'функции'. Конструкторы используются для генерации всех значений (классов эквивалентности) сорта. При таком подходе литералы рассматриваются как операторы без аргументов.

Примеры:

a) В булевском типе конструкторами будут литералы.

b) В натуральном типе конструкторами будут литерал 0 и оператор Следующий; каждое натуральное число может быть сконструировано с использованием только литерала 0 и оператора Следующий.

c) Генератор для множеств имеет в качестве конструкторов литерал пустое_множество и оператор Добавить; любое множество может быть построено с использованием только литерала пустое_множество и добавить.

d) Целый тип может быть построен из литералов 0 и 1 и операторов + и унарный минус.

Следует отметить, что иногда имеется несколько возможностей выбрать множество конструкторов. В оставшейся части данного раздела любой выбор считается приемлемым, но лучшими вообще являются маленькие множества.

Затем рассматриваются одна за другой все функции. Для каждого аргумента функции составляется список всех возможных термов, состоящих из конструкторов. Во избежание трудностей с бесконечным числом термов следует использовать квантификацию.

Примеры:

a) Для натуральных чисел этот список может быть сокращен до следующего:

0
Следующий (n), где n – произвольное натуральное число.

b) Для множеств возможным списком является:

пустое_множество
Добавить (s, i), где s – произвольное множество, а i – произвольный элемент.

Если для терма в левой части равенства, правая часть которого содержит Добавить (s, i), существенно различать случаи, когда s пусто и не пусто, то список можно переписать следующим образом:

пустое_множество
Добавить (пустое_множество, i), где i – любой элемент,
Добавить (Добавить (s, i), j), где s – любое множество, а i, j – любые элементы.

После того как такой список создан, левые части равенств получаются путем применения каждой функции к любой комбинации аргументов из списка. Идентификаторам значений в разных аргументах даются разные имена. Такая же процедура, как и приведенная выше для функций, может быть проведена для конструкторов; в этом случае она дает соотношения между термами, в которых конструкторы используются в различном порядке.

Примеры:

a) В случае оператора умножения для натуральных чисел с сигнатурой

" * ": Натуральный, Натуральный -> Натуральный

эта процедура дает левые части следующих (неполных) равенств. Пользователь должен задать правые части.

```
                0 * 0                 ==  ...;
                0 * Следующий (n)     ==  ...;
Следующий (n)   * 0                   ==  ...;
Следующий (n)   * Следующий (m)       ==  ...;
```

b) Для операторов Содержится и Удалить в генераторе Множество (§ D.6.2.1) этот подход уже был использован.

c) Для сорта Цвет конструкторами являются Взять и Смешать.
Для аргументов

Взять (p), где p – произвольный основной цвет,
Смешать (p, c), где p – произвольный основной цвет, а c – произвольный (смешанный цвет)

должен быть определен оператор, подобный оператору Содержит из § D.6.1.4.2.

Поскольку в § D.6.1.4.2 было обещано задать для этого оператора более изящные равенства, они приводятся полностью:

```
Содержит (Взять (p), q) == p = q;
Содержит (Смешать (p, c), q) == (p = q) OR Содержит (c, q);
```

Такая процедура порождения равенств может привести к более чем необходимому числу равенств, но она очень надежна. Так, в примере с умножением натуральных чисел скорее всего будет установлено свойство коммутативности умножения и поэтому понадобится только последнее (или второе) равенство из трех первых.

Процедура, описанная в настоящем разделе, может быть использована в комбинации с процедурой из предыдущего раздела, где она будет полезна при выполнении работы "Придумать_предложение".

D.6.3.3 *Спецификация тест-множества*

По-другому равенства можно рассматривать с точки зрения практической реализации. Если операторы практически реализованы как функции в языке программирования, равенства описывают, как эти функции должны тестироваться.

Нужно вычислять выражения в левой и правой частях равенства и проверять, будут ли они эквивалентны. К трудностям может привести конструкция FOR ALL. Часто преодолеть трудности помогает очень прагматический подход:

```
Вместо                                  FOR ALL i IN Целый
при тестировании можно использовать     FOR ALL i IN {-10, -1, 0, 1, 10}
```

и в большинстве случаев этого недостаточно.

При выполнении работы "Придумать_предложение" из процедуры § D.6.3.1 полезно рассматривать равенства как требования при практической реализации.

D.6.4 *Особые возможности*

В этом разделе описываются особые возможности SDL, которые редко используются, но иногда намного облегчают жизнь.

D.6.4.1 *Скрытые операторы*

Иногда множество равенств может быть упрощено или сделано более удобочитаемым за счет введения дополнительного оператора, который, однако, не должен использоваться в процессах. Это означает, что оператор видим внутри определения типа, но невидим извне.

Такой результат может быть достигнут введением 'скрытого типа', то есть типа, который не будет использоваться пользователем. Из этого скрытого типа пользователь может наследовать все операторы, к которым ему разрешен доступ, а использоваться будет наследуемый тип. Корректность использования можно проконтролировать проверкой всех объявлений переменных (не должно быть ни одной переменной сорта, введенного скрытым типом).

Особая возможность скрытых операторов означает, что то же самое может быть достигнуто путем ограничения видимости этих операторов только на область равенств. Это делается добавлением к оператору восклицательного знака.

Пример:

Правильным способом задания одноэлементного множества в генераторе Множество является

Добавить (пустое_множество, x)

и все пользователи должны следовать этому способу. При спецификации в равенствах может использоваться специальный оператор, например:

Создать_множество! : Элемент - > Множество;

определенный равенством:

Создать_множество! (itm) == Добавить (пустое_множество, itm);

которое может использоваться в частичном определении типа, но не в теле SDL-процесса.

D.6.4.2 *Упорядочение*

Если требуется специфицировать упорядочение элементов сорта, это обычно означает, что должны быть заданы четыре оператора (<, < =, >, > =) и стандартные математические свойства (транзитивность и пр.). Если литералов много, то должно быть задано и много равенств. Например, таким образом определен предварительно определенный тип данных Знаковый.

SDL обеспечивает особую возможность справиться с этими пространными, неудобочитаемыми и утомительными определениями типов – сокращение ORDERING.

ORDERING задается в списке операторов, предпочтительно в начале или в конце этого списка. Тем самым вводятся операторы упорядочения и стандартные равенства. Если ORDERING специфицировано, то литералы, если таковые имеются, должны задаваться в порядке возрастания.

Пример:

```
NEWTYPE Четная_десятичная_цифра
    LITERALS 0, 2, 4, 6, 8
    OPERATORS
        ORDERING
ENDNEWTYPE Четная_десятичная_цифра;
```

Таким образом неявно задано упорядочение $0 < 2 < 4 < 6 < 8$.

В § D.6.2.2 (Наследование) дано предупреждение о том, что следует быть осторожным при добавлении литералов к наследуемому сорту. Мы имеем теперь прекрасную возможность проиллюстрировать это.

Предположим, потребовалось задать следующее расширение сорта Четная_десятичная_цифра:

```
NEWTYPE Десятичная_цифра
    INHERITS Четная_десятичная_цифра
    OPERATORS ALL
    ADDING
        LITERALS 1, 3, 5, 7, 9
        AXIOMS
            0 < 1;    1 < 2;
            2 < 3;    3 < 4;
            4 < 5;    5 < 6;
            6 < 7;    7 < 8;
            8 < 9;
ENDNEWTYPE Десятичная_цифра;
```

Заданные здесь аксиомы не могут быть опущены. Без этих аксиом было бы только так называемое частичное упорядочение:

$0 < 2 < 4 < 6 < 8$
и $1 < 3 < 5 < 7 < 9$.

Вместе с вышеприведенными аксиомами получится полное упорядочение:

$0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$.

Если же вместо множества вышеприведенных аксиом была бы задана единственная аксиома "$9 < 0$", получилось бы другое полное упорядочение:

$1 < 3 < 5 < 7 < 9 < 0 < 2 < 4 < 6 < 8$.

D.6.4.3 *Сорта с полями*

Как показано в § 5.4.1.10 Рекомендации, можно определить структурные сорта, не прибегая к специальным конструкциям. Но, поскольку структурные сорта являются и общими, и полезными, то оправданно введение в язык дополнительных конструкций.

Структурные сорта используются, когда значения объектов формируются путем объединения значений нескольких сортов. Каждое значение из такого объединения характеризуется именем, называемым именем поля. Сорт поля фиксирован.

Примеры:

```
NEWTYPE Абонент
    STRUCT  номера Ключ_номера;
            имя    Ключ_имени;
            админ  Административный;
ENDNEWTYPE Абонент;

NEWTYPE Ключ_имени
    STRUCT  имя,
            улица  Знакостроковый;
            номер  Целый;
            город  Знакостроковый;
ENDNEWTYPE Ключ_имени;
```

Вместе со структурным сортом неявно определяются некоторые операторы:

a) оператор конструктора, "(." до, и ".)" после значений полей;

b) операторы избрания поля, переменная структурного сорта, за которой либо стоит ! и имя поля, либо имя поля в скобках. Переменную, за которой стоит !, не следует путать со скрытым оператором (§ D.6.4.1).

На рис. D-6.4.1 приведен пример

```
PROCESS Некоторый_процесс;
    DCL имя,ул,где  Знакостроковый,
        ном         Целый,
        ким         Ключ_имени;
    /* . . . Текст, в котором переменным имя и ул присваиваются значения */
    TASK ким := (.имя,ул,5,'Лондон'.);
    /* . . . Текст, в котором нет присваиваний переменной ким */
     TASK где := ким!город;
    /* Теперь имеет место где='Лондон' */
ENDPROCESS Некоторый_процесс;
```

РИСУНОК D-6.4.1

Пример использования неявных операторов структурного сорта

D.6.4.4 *Индексированные сорта*

Индексированный сорт – это сорт, в типе которого определено имя оператора Извлечь! Из предварительно определенных типов данных таким типом будет генератор Массив. Массив является одним из наиболее общих примеров индексированного типа.

Для скрытого оператора Извлечь! имеется специальный конкретный синтаксис, который используется вне определений типа.

Можно было бы решить, что тип Индекс в предварительно определенном генераторе Массив должен быть 'простым', таким как Целый, Натуральный или Знаковый. Однако нет причин, по которым структура, такая как Ключ_имени, не могла бы использоваться как Индекс.

Пример:

```
NEWTYPE База_данных_абонент
    Массив (Ключ_имени, Абонент)
ENDNEWTYPE База_данных_абонент;
```

Сорта Ключ_имени и Абонент определены в предыдущем разделе. Предположим, имеется процедура Счет с одним параметром сорта Абонент и эта процедура определена в процессе, имеющем также переменную Бд_аб сорта База_данных_абонент. В этом процессе может возникнуть следующий вызов:

```
CALL Счет (Бд_аб((.'Р.М.', 'Даунингстрит', 10, 'Лондон'.)));
```

D.6.4.5 *Значения переменных, принимаемые по умолчанию*

Как объяснено в разделе об объявлении переменных (§ D.3.10.1), можно присвоить значение переменной непосредственно после объявления. Однако в некоторых типах имеется значение, которое (почти) всегда является первоначальным значением переменной. Существует особая возможность, позволяющая избежать необходимости писать первоначальное значение во всяком объявлении: клаузула DEFAULT.

В качестве примера рассмотрим множества. Весьма вероятно, что почти все переменные, независимо от того, о каких множествах идет речь, будут инициализированы значением пустое_множество.

Нотация в конце списка равенств:

```
DEFAULT пустое_множество
```

обеспечивает инициализацию любой переменной в любой реализации генератора значением пустое_множество, за исключением случая явной инициализации (см. § D.3.10.1).

Если не очевидно, что первоначальные значения всех переменных сорта должны быть одинаковыми, клаузулу DEFAULT использовать не следует. Иначе будет трудно избежать сюрпризов.

D.6.4.6 *Активные операторы*

Пользователи, знакомые с Рекомендацией Z.104 по SDL от 1984 года, могут спросить, что же произошло с так называемыми активными операторами. Эта особая возможность была исключена, поскольку

a) в ней нет необходимости, ибо обычные операторы вместе с процедурами и/или макрокомандами обеспечивают ту же силу выражения,

b) она делала неудобочитаемыми равенства,

c) большинство пользователей испытывали затруднения в правильном ее использовании,

d) она не вписывается в модель абстрактного типа данных, основанного на начальных алгебрах, которые выбраны в качестве математического обоснования этой части SDL.

D.7 *Дополнительные указания по разработке графических и текстовых материалов*

Для каждой концепции SDL в Рекомендации специфицируется основной способ изображения концепции. Например, концепция работы изображается в SDL/GR с помощью прямоугольника, а в SDL/PR –с помощью ключевого слова TASK.

Руководство пользователя дополняет Рекомендацию спецификацией правил того, как рисовать диаграммы и писать тексты. Эти правила применимы ко всем концепциям, с тем чтобы подчеркнуть, что считается подходящим, ошибочным или неудачным рисованием/написанием.

D.7.1 *Указания по SDL/GR*

D.7.1.1 *Общие соображения*

Общими указаниями для разработки диаграмм являются следующие:

– направление чтения должно быть сверху вниз и слева направо;

– диаграммы не должны нести слишком много информации на одном и том же уровне. Часто бывает удобно расчленять большие диаграммы на поддиаграммы, охватывая таким способом различные части или аспекты, например, используя механизм ссылок.

D.7.1.2 *Входные и выходные линии*

Входные линии в символы и выходные линии из символов должны рисоваться вертикально (в соответствии с чтением сверху вниз), и только в тех случаях, когда это практически не выгодно, можно использовать горизонтальные входные и выходные линии.

Исключениями из этого общего правила являются:

– Символы принятия решения, имеющие две или три выходные линии, рисуют обычно с двумя горизонтальными выходами (плюс один вертикальный).

– Вызовы макрокоманд используют горизонтальные и вертикальные соединения.

– Как входные, так и выходные коннекторы имеют обычно горизонтальные входные и выходные линии.

Диагональные линии могут использоваться только в исключительных случаях (например, для каналов и маршрутов сигналов).

Вертикальные входные и выходные линии должны делить символы на две части, имеющие в горизонтальном направлении одинаковую длину.

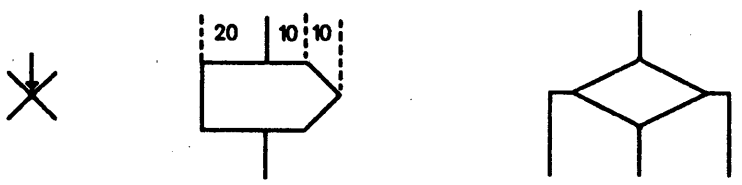


РИСУНОК D-7.1.1

Правильно нарисованные входы и выходы

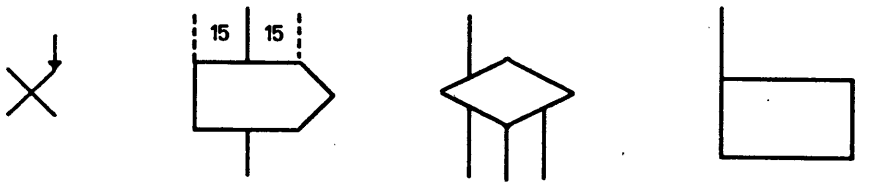


РИСУНОК D-7.1.2

Неправильно нарисованные входы и выходы

D.7.1.3 *Символы*

a) Символы должны быть нарисованы так, чтобы их горизонтальная и вертикальная оси совпадали с осями листа бумаги.

b) Вертикальное зеркальное отображение символов допускается только для входов, выходов, расширений текста и комментариев (см. рис. D-7.1.3).

c) Общим отношением высоты к длине для всех символов в графе, а также символов ссылок должно быть 1:2.

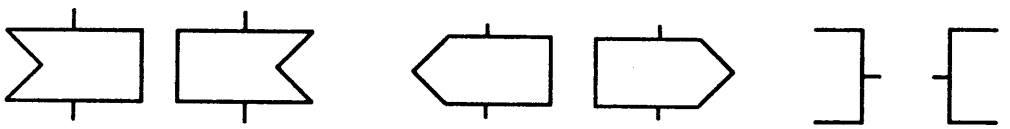


РИСУНОК D-7.1.3

Четыре символа, допускающие зеркальное отображение

D.7.1.4 *Лекало*

Лекало для символов SDL/GR изображено на третьей стороне обложки настоящего выпуска. Схематическое изображение этого лекала приведено на рис. D-7.1.4.

На этом рисунке изображены непосредственно символы Ввода, Вывода, Принятия Решения, Альтернативы, Процесса, Старта, Работы, Состояния, Сохранения, Ссылки на Сервис, Коннектора и Стопа; изображения приводятся в трех размерах: 20 x 40 мм, $20/\sqrt{2}$ x $40/\sqrt{2}$ мм и 10 x 20 мм. Внутренние соотношения указаны только для самого большого размера.

Символы для Вызова Процедуры, Вызова Макрокоманды и Создания могут быть получены из символа Работы путем рисования дополнительной(ых) горизонтальной(ых) или вертикальной(ых) линии(ий), указанных на рисунке.

Символ Старта Процедуры может быть получен из символа Старта Процесса путем рисования указанных дополнительных вертикальных линий.

Символ Возврата является комбинацией символов Коннектора и Стопа.

Символы Комментария, Расширения Текста и Списка Сигналов рисуют, используя символ Работы.

Символы Приоритетного Ввода и Приоритетного Вывода могут быть получены из символов Ввода и Вывода пририсовыванием указанных дополнительных линий.

Символ Ссылки на Процедуру может быть получен из символа Ссылки на Процесс путем рисования двух дополнительных вертикальных линий, как они указаны на рисунке.

Символы Разрешающего Условия и Непрерывного Сигнала могут быть нарисованы путем использования символа Стопа.

Каналы, Маршруты Сигналов и линия, ведущая к символу Расширения Текста, являются сплошными линиями.

Линия, ведущая к символу Комментария, является пунктирной линией с отношением 1:1.

Символ Текста рисуется с помощью символа Работы, в котором загибается правый верхний угол, как это указано на рисунке.

Все рекомендуемые символы определены в Рекомендации. Обзор рекомендуемых символов можно найти в Приложении C – Сводное описание графического синтаксиса. На рисунке указаны три предпочтительных размера. Если используются другие размеры, отношения должны быть теми же (то есть 1:2). Приведенные размеры, то есть длины 40 мм, 28 мм и 20 мм, позволяют фотоуменьшение от формата бумаги A3 к формату A4 с совместимыми размерами символов (так как 40 мм/$\sqrt{2}$ = 28 мм и 28 мм/$\sqrt{2}$ = 20 мм).

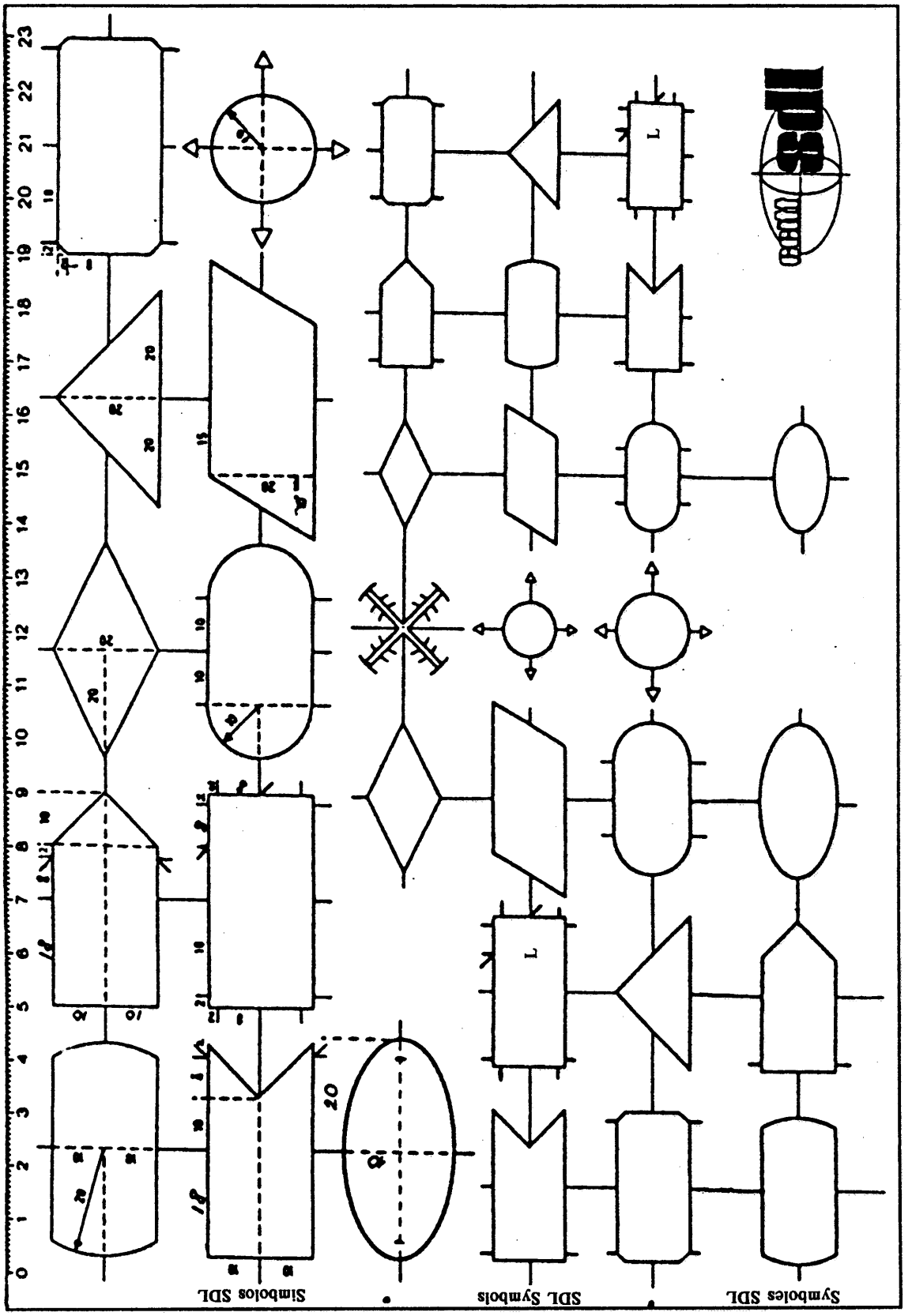


РИСУНОК D-7.1.4

Схематическое изображение лекала

D.7.2 *Указания по SDL/PR*

Общими правилами разработки текстового SDL являются:

– Направление чтения должно быть сверху вниз и слева направо.

– Текст должен быть разбит на части, охватывающие различные аспекты.

– Комментарии, касающиеся уровня предложений, должны начинаться в той же колонке.

– Строки должны иметь отступы, которые могут следовать обычной иерархии концепций SDL, как это показано на рис. D-7.1.5.

```
SYSTEM A;
    SIGNAL S1,S2;
    BLOCK B;
        PROCESS P;
            START;
            NEXSTATE ST1;
            STATE F:
                INPUT G: . . .
                INPUT F: . . .
                . . .
            ENDSTATE F;
        ENDPROCESS P;
    ENDBLOCK B;
ENDSYSTEM A;
```

РИСУНОК D-7.1.5

Пример отступов в текстовом SDL

D.8 *Документация*

D.8.1 *Введение*

ISO определяет документ как "ограниченное и связанное количество информации, записанное на носителе в форме, допускающей чтение". Поэтому документ должен рассматриваться как строго ограниченная логическая единица. Документы используются для передачи всей относящейся к системе информации, специфицированной с помощью SDL.

Если для хранения документа в качестве физического носителя используется бумага, то очень часто термин "документ" неправильно применяется к листам бумаги, а не к их логическому содержимому. По мере возрастающего использования магнитных носителей термину возвращается его исходное значение.

В настоящей главе рассматривается скорее логическая организация документов, а не их физическая организация. Она оставлена на усмотрение пользователя. Некоторая сходность требований к логической и физической организации документов означает, что из нижеследующего текста пользователь может извлечь какие-то полезные советы насчет физической организации документов.

Разбивая информацию на подходящее число документов, можно сделать систему более понятной и легче управляемой.

В языке нет рекомендаций относительно того, какие документы нужны или какой должна быть структура документов. Однако, чтобы пользователю было легче обращаться с документами, ниже приводится несколько предложений.

D.8.2 *Типы изображения системы*

Результатом спецификации системы с помощью SDL является комплект определений на SDL/GR и/или SDL/PR.

Эти определения могут быть либо вложенными, либо образовывать цепочку в зависимости от типа изображения системы, которое может быть либо иерархическим, либо плоскостным. На рис. D-8.2.1 и D-8.2.2 система изображена на SDL/GR в двух альтернативных вариантах. Эти два рисунка не являются полными спецификациями системы, так как для простоты на них изображены только вводы и выводы, но отсутствуют возможные определения сигналов и данных.

Конечно, при спецификации системы вполне допускается сочетание этих двух типов изображения. Если определения, как на рис. D-8.2.1, образуют цепочку, то на них имеются 'ссылки'; этот механизм допустим для определений как на SDL/PR, так и на SDL/GR.

Если смотреть на спецификацию системы как на комплект определений, то документ можно рассматривать как контейнер для этих определений.

Если система невелика и иерархическая, как на рис. D-8.2.2, то достаточно одного единственного документа. Если же, как на рис. D-8.2.1, используется плоскостное представление, то может потребоваться большее число документов; например по одному документу на каждое определение.

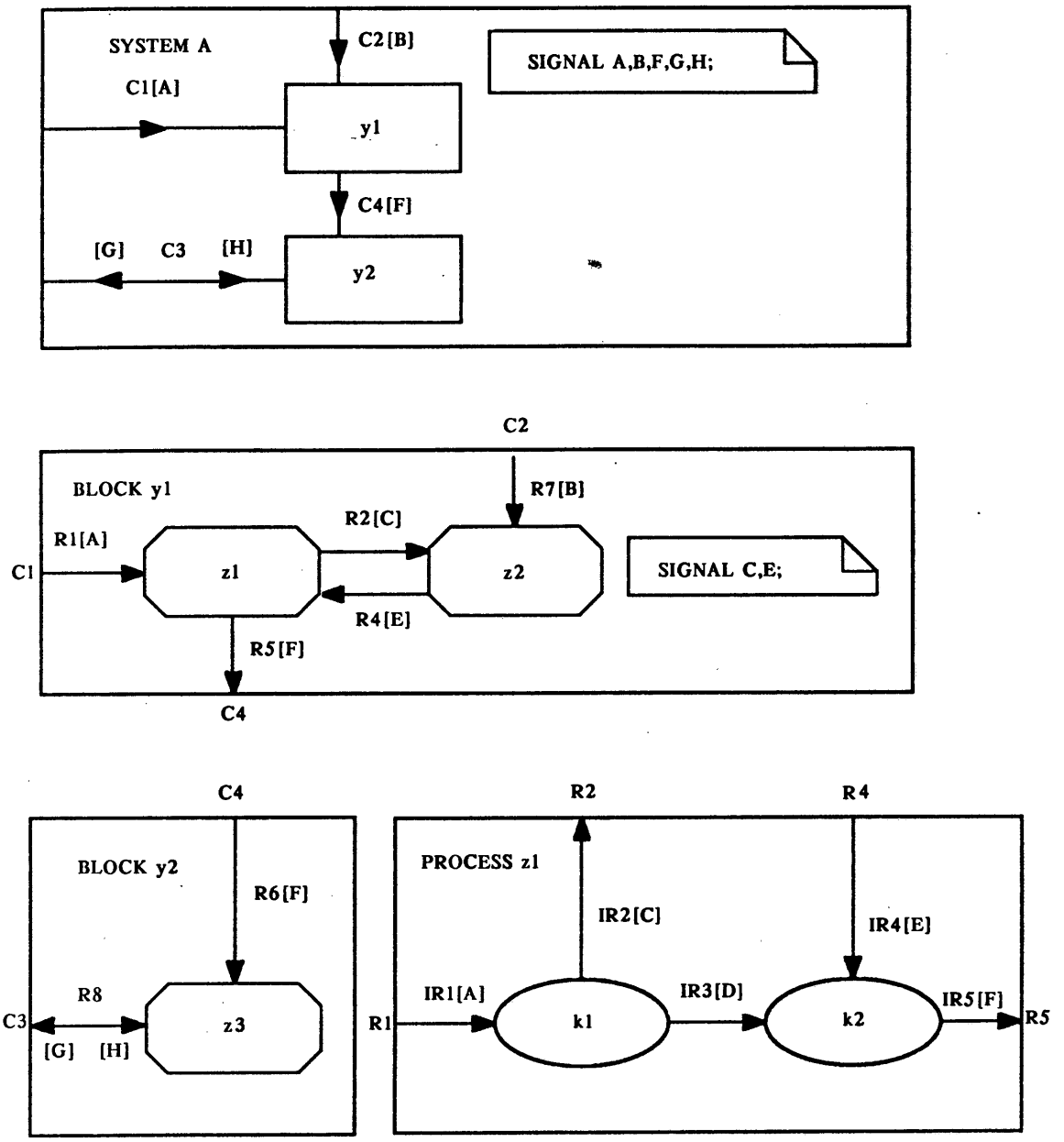


РИСУНОК D-8.2.1a

Пример цепочки диаграмм (на которые имеются ссылки) при плоскостном представлении системы

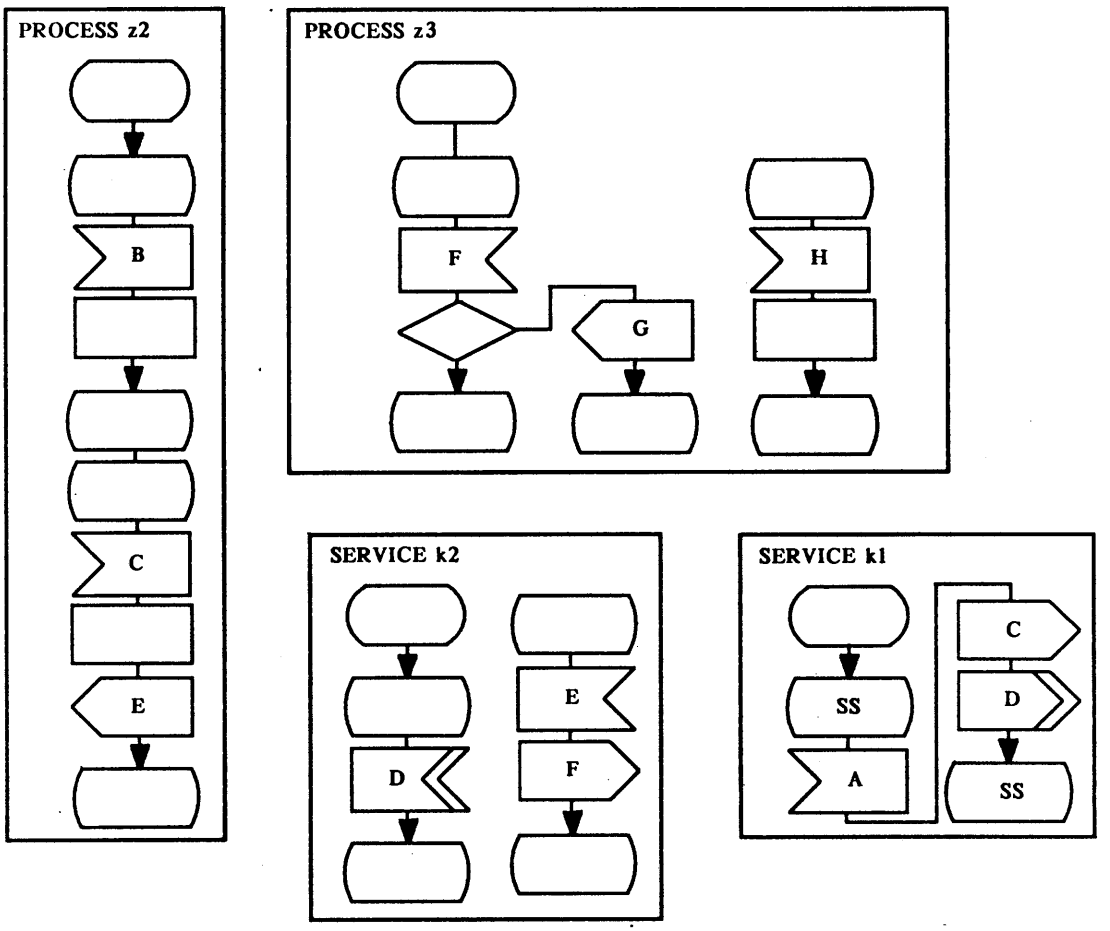


РИСУНОК D-8.2.1b

**Пример цепочки диаграмм (на которые имеются ссылки)
при плоскостном представлении системы**

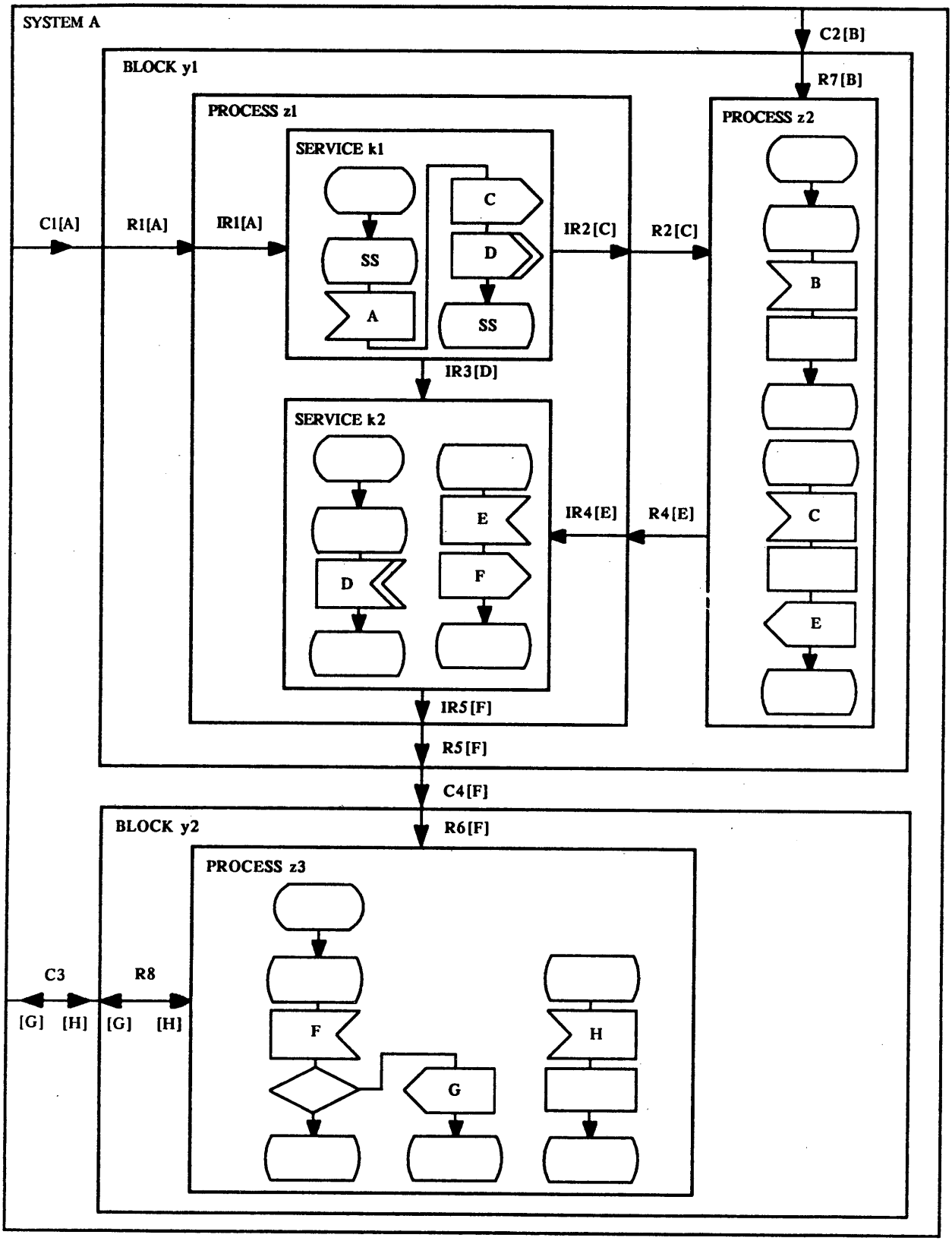


РИСУНОК D-8.2.2

Пример вложенных определений при иерархическом представлении системы

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.8

При выборе типа изображения системы требуется рассмотреть вопрос о желательном типе документа. Чтобы получить по одному документу на каждое определение, надо выбрать плоскостное изображение. Если желательно иметь один единственный документ, образующий всю спецификацию системы, то следует остановиться на иерархическом представлении.

Наиболее естественным будет, по-видимому, сочетание этих методов. К такому сочетанию применимы следующие правила:

1) определение не должно быть разделено по нескольким документам;

2) если желательно иметь определение в виде отдельного документа, то требуется ссылка на него, но он не должен быть вложенным в другое определение;

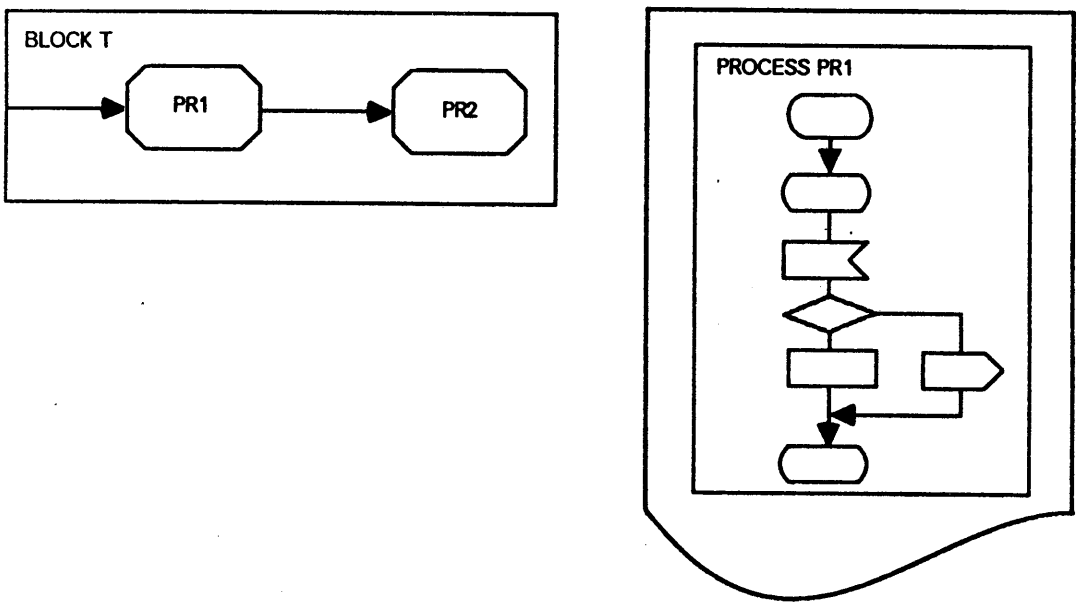


РИСУНОК D-8.2.3

Определение, на которое имеется ссылка, может быть помещено на отдельном документе

3) если для разбиения диаграммы на несколько листов диаграммы используется концепция логических листов, то листы диаграмм должны совпадать с физическими листами документа (см. рис. D-8.2.4);

4) если диаграмма занимает более одного листа, то на нее должна быть сделана ссылка, но она не должна быть погружена в другую диаграмму.

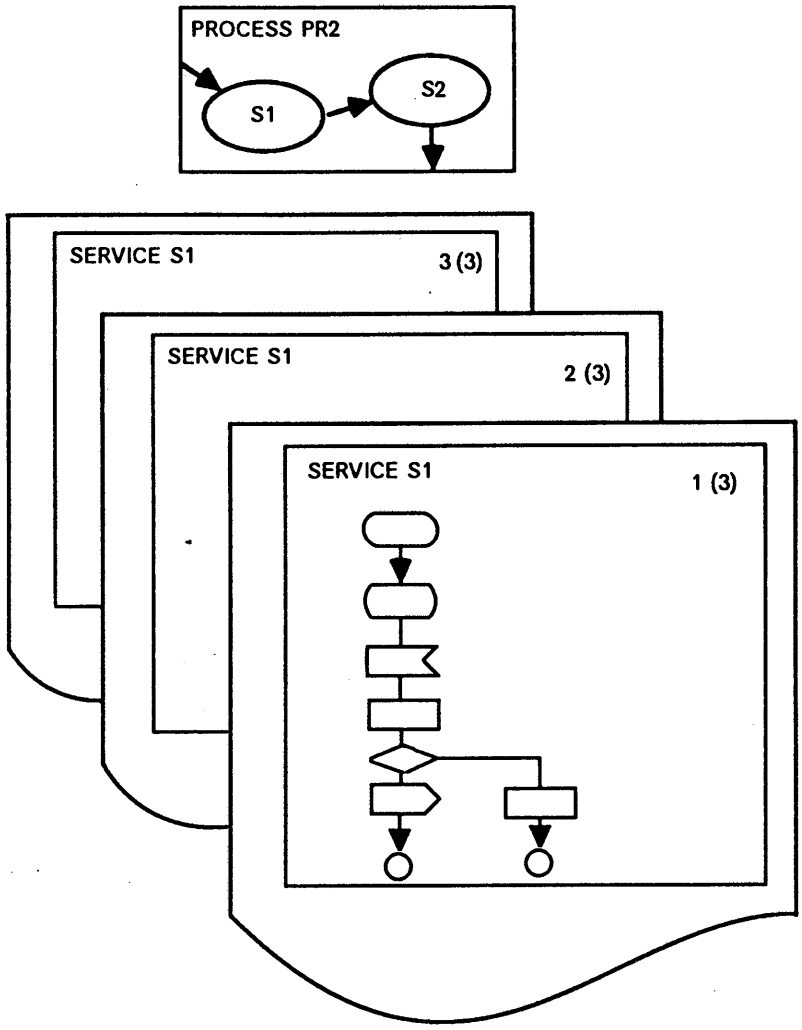


РИСУНОК D-8.2.4

Диаграмма сервиса, на которую имеется ссылка, разложенная на три листа

D.8.3 *Структура документа*

Комплект документов, охватывающий все определение системы, может быть структурирован. Структурирование документа в случае, когда документ содержит ссылку на поддокумент, может быть связано с любым элементом системы, такими как система, блок и процесс (см. рис. D-8.3.1).

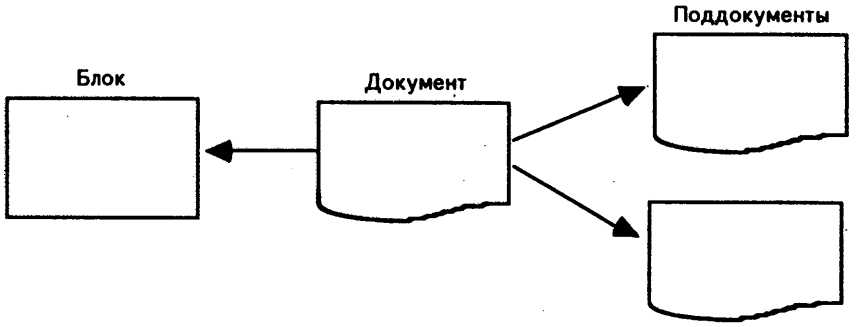


РИСУНОК D-8.3.1

Структура документов

D.8.4 *Механизм ссылок*

Механизм ссылок в языке, в котором для ссылок между концепциями используются имена концепций, может быть использован и для ссылок между документами. Такой подход является естественным, когда документ совпадает с определением.

D.8.5 *Классификация документов*

Документы можно классифицировать в соответствии с типом содержащихся в них определений.

При такой классификации определение по меньшей мере процесса или сервиса, описывающее динамическое поведение системы, должно быть выделено в отдельный документ. Эти документы могут, кроме того, содержать определения переменных.

Возможная структура документа, описывающего систему, приведена на рис. D-8.5.1.

В этом примере определения каналов и маршрутов сигналов включены в документы, содержащие определения системы, блока и процесса. Определения сигналов, определения данных и определения списков сигналов размещены по отдельным документам; кроме того, предполагается, что все определения данных находятся на уровне системы.

Определения процедур, определения макрокоманд и определения сервисов образуют поддокументы по отношению к документу, описывающему процесс.

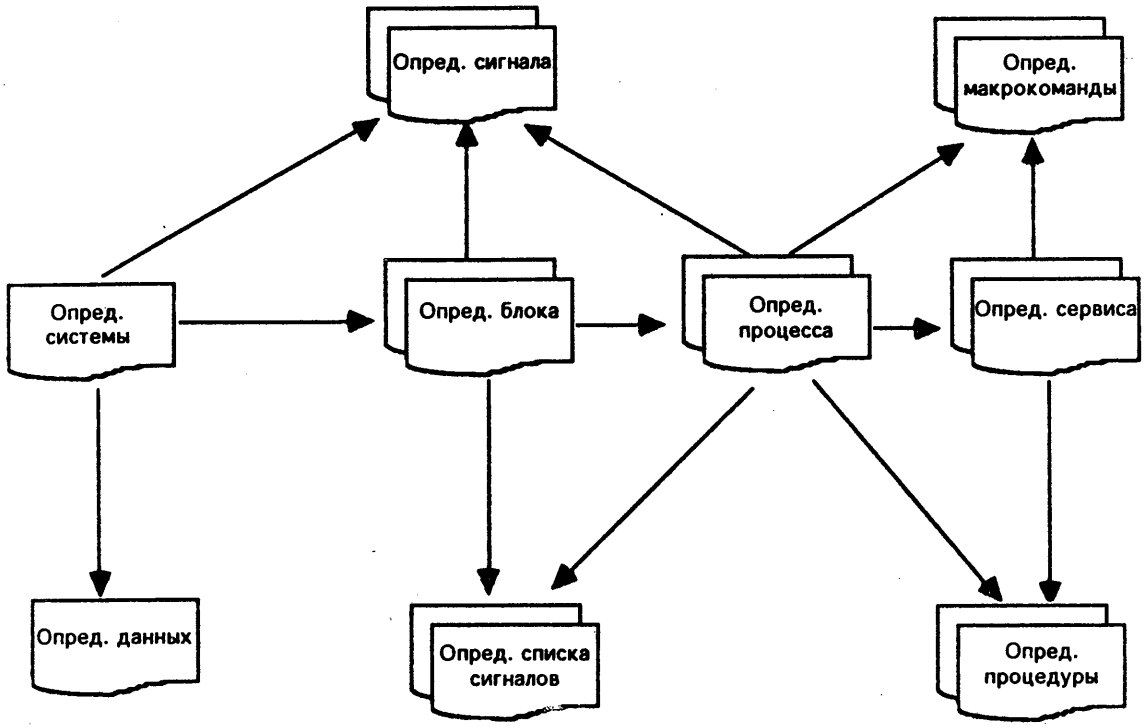


РИСУНОК D-8.5.1

Возможная структура документов системы

Если несколько сервисов совместно образуют некоторую функцию системы, то эти сервисы могут быть описаны в общем документе.

Различные определения сервисов могут быть помещены один за другим в одном документе, но, кроме того, их можно расположить рядом друг с другом на одной странице документа. Последняя планировка документа дает хорошее представление о взаимодействии между сервисами. Рис. D-8.5.2 является примером страницы документа, описывающего сервисы.

Для больших систем должна приводиться "таблица содержимого" системы, указывающая, где найти состояния, вводы, выводы и т.д. Кроме того, таблица содержимого должна содержать все концепции, то есть где найти определения и где они используются. Примерами являются система, блоки, каналы, сигналы, процессы, сервисы, макрокоманды, процедуры.

Такие таблицы содержимого системы могут сами образовывать отдельный документ.

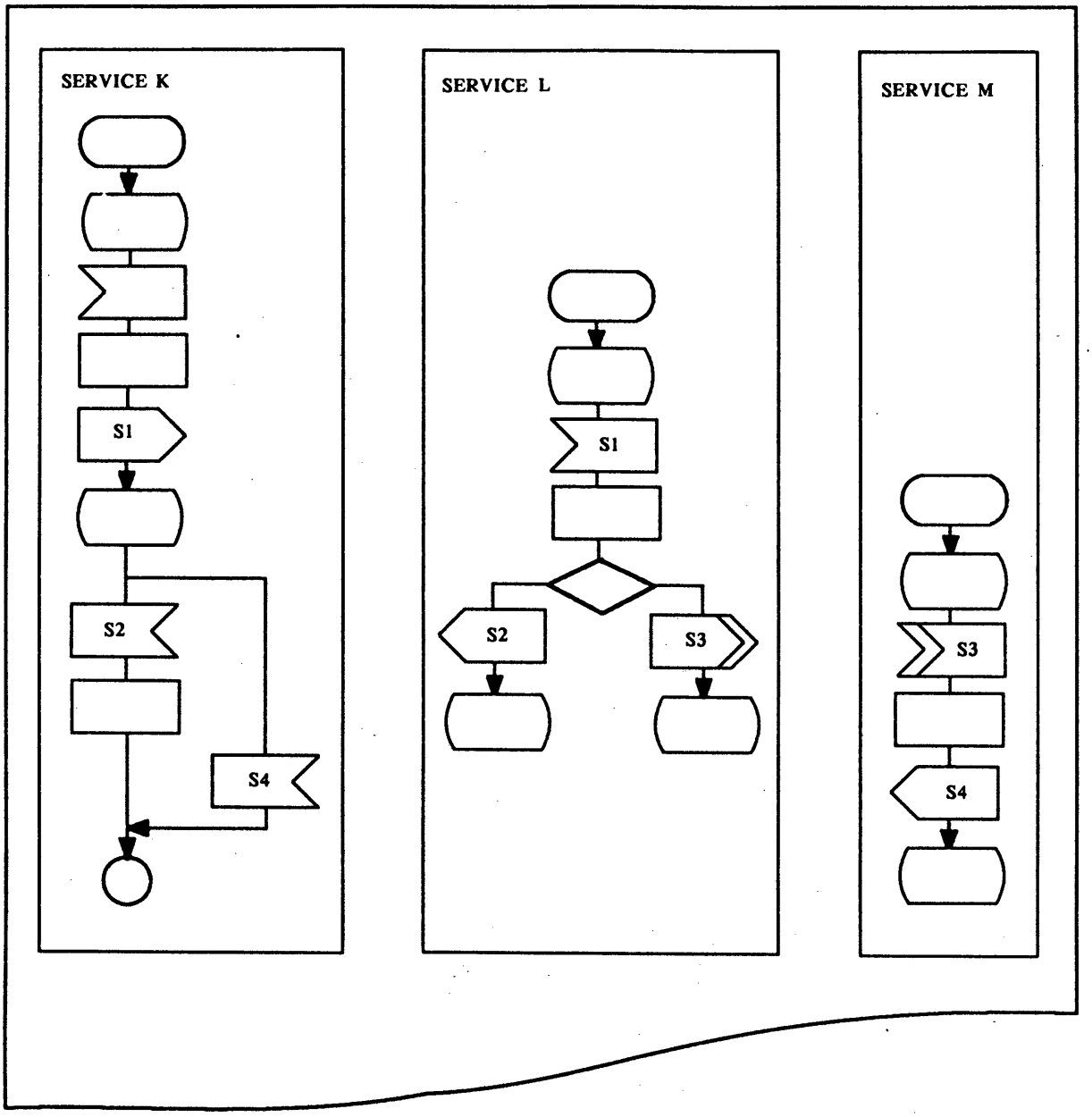


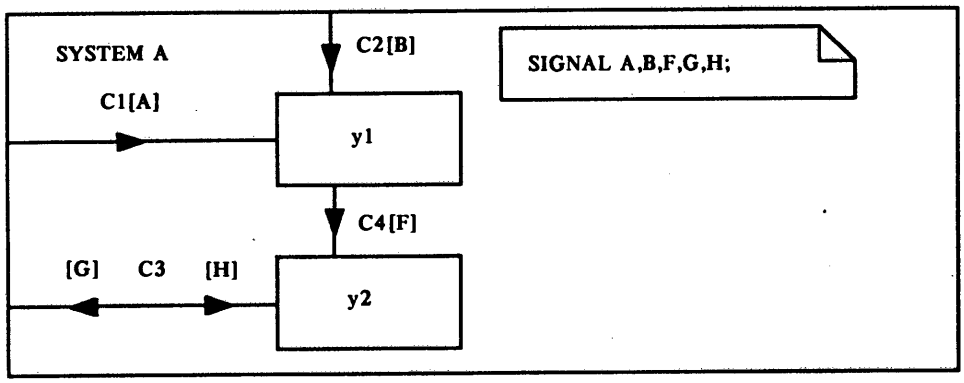
РИСУНОК D-8.5.2

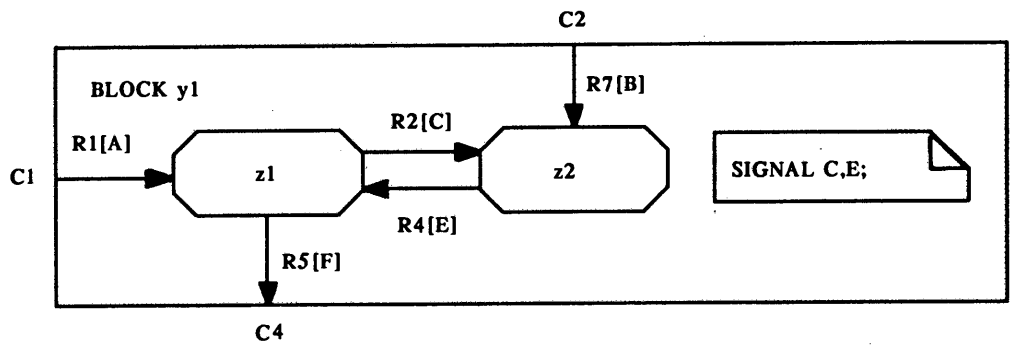
Одна страница документа о сервисах

D.8.6 *Сочетание SDL/GR и SDL/PR*

При спецификации системы допустимо совместное использование SDL/GR и SDL/PR. Концепции системы, блока, процесса, сервиса, процедуры, макрокоманды, канала и т.д. могут быть специфицированы как на SDL/GR, так и на SDL/PR. Переход от SDL/PR к SDL/GR или от SDL/GR к SDL/PR осуществляется с помощью механизма ссылок, имеющегося в языке. Концепция, на которую есть ссылка в SDL/PR, может быть специфицирована на SDL/GR, и концепция, на которую есть ссылка в SDL/GR, может быть специфицирована на SDL/PR.

Рис. D-8.6.1 является "полной" спецификацией системы, использующей сочетание SDL/GR и SDL/PR. Это та же самая система, что и приведенная на рис. D-8.2.1 и рис. D-8.2.2. Каждое определение в этом примере может быть помещено в отдельном документе.






```
BLOCK y2;
   SIGNALROUTE R8FROM ENV TO z3 WITH H;
                FROM z3 TO ENV WITH G;
   SIGNALROUTE R6FROM ENV TO z3 WITH F;
   CONNECT C3 AND R8;
   CONNECT C4 AND R6;
   PROCESS z3 REFERENCED;
ENDBLOCK y2;
```

```
PROCESS z1;
   SIGNALROUTE IR1 FROM ENV TO k1 WITH A;
   SIGNALROUTE IR2 FROM k1 TO ENV WITH C;
   SIGNALROUTE IR3 FROM k1 TO k2 WITH D;
   SIGNALROUTE IR4 FROM ENV TO k2 WITH E;
   SIGNALROUTE IR5 FROM k2 TO ENV WITH F;
   CONNECT R1 AND IR1;
   CONNECT R2 AND IR2;
   CONNECT R4 AND IR4;
   CONNECT R5 AND IR5;
   SERVICE k1;
      START;
      STATE SS;
         INPUT A;
         OUTPUT C;
         PRIORITY OUTPUT D;
         NEXSTATE SS;
      ENDSERVICE k1;
      SERVICE k2 REFERENCED;
   ENDPROCESS z1;
```

РИСУНОК D-8.6.1a

Спецификация системы, содержащая возможное сочетание SDL/GR и SDL/PR

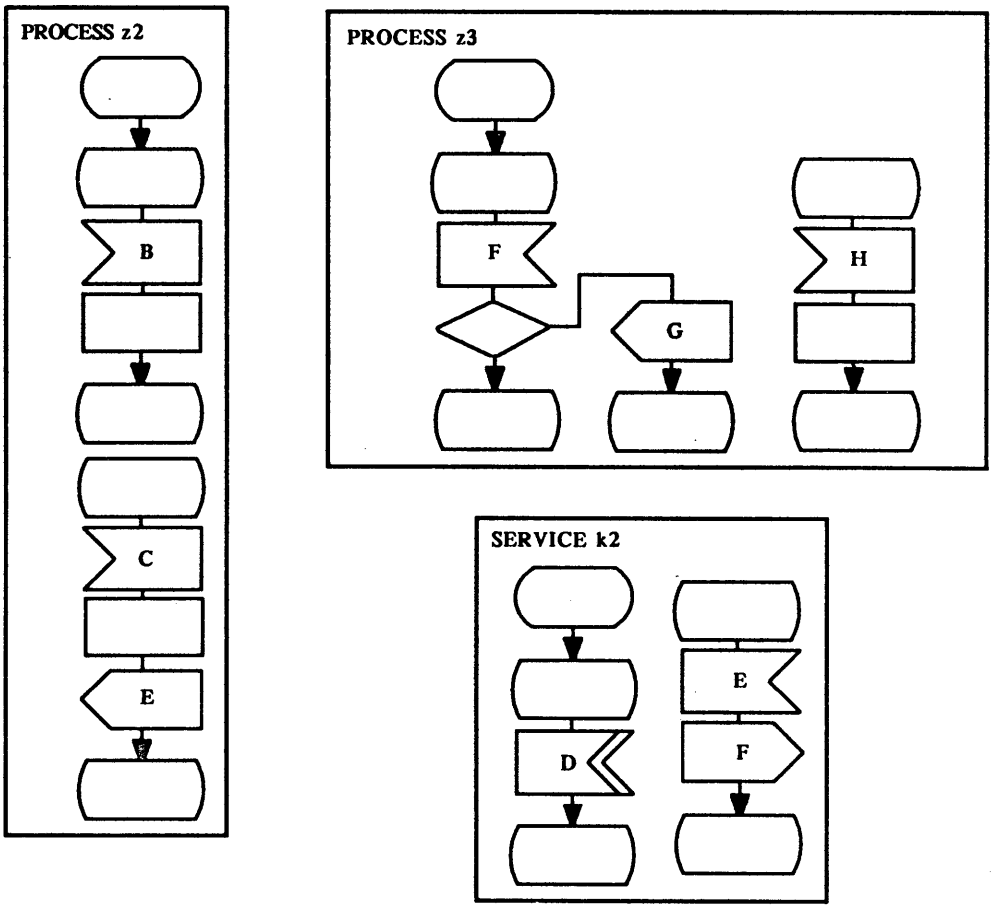


РИСУНОК D-8.6.1b

Спецификация системы, содержащая возможное сочетание SDL/GR и SDL/PR

D.9 *Отображения*

Здесь описываются некоторые аспекты отображения между SDL и CHILL (§ D.9.1) и отображения между SDL/GR и SDL/PR (§ D.9.2).

D.9.1 *Отображения между SDL и CHILL*

Ниже иллюстрируются некоторые возможные способы отображения SDL на CHILL. Приводимое описание является очень кратким и не должно рассматриваться ни как исчерпывающее, ни как утверждающее, что на практике должен быть выбран один из этих способов.

Рассмотрение отображения не должно ограничиваться имеющимся в распоряжении компилятором с CHILL и предполагаемой машиной, оно должно учитывать и более общие случаи. Отображение является очень трудной интеллектуальной работой и только благодаря опыту проектировщики/программисты могут выбрать конкретную структуру CHILL-программы, с помощью которой можно реализовать конкретное SDL-представление. Это касается также представления функций, реализуемого CHILL-программами. Взаимооднозначное отображение (если оно возможно) не обязательно является наилучшим способом использования SDL для представления функций, реализованных с помощью CHILL.

При таком подходе общая структура CHILL-программы, полученной из SDL-диаграммы, выглядит так, как это отображено на рис. D-9.1.1.

Некоторые примеры отображения между конструкциями двух языков приведены на рисунках D-9.1.2 – D-9.1.5. Они охватывают нижеследующие конструкции:

- состояние и получение/сохранение сигналов; выбор следующего состояния;
- вывод;
- присоединение;
- принятие решения.

Объявляемый модуль содержит как определения, так и объявления всех сигналов, используемых в преобразованной SDL-диаграмме, и всех переменных, ассоциируемых с этими сигналами. Все эти переменные принадлежат модулю, представляющему функциональный блок SDL-диаграммы.

```
Declaring: MODULE
        /*  Модуль CHILL, содержащий сигналы и ассоциированные переменные,
            использованные в SDL - диаграмме                                  */

        GRANT
        /*  присвоение сигналов                                               */

        SIGNALS
        /*  определение сигналов                                              */

        SYNMODE (OR NEWMODE)
        /*  определение типа                                                  */

END Declaring;

Functional_Block: MODULE
        /*  модуль, содержащий процедурную часть SDL - диаграммы              */

        SEIZE

        /*  охват всех сигналов и переменных, которые могут быть получены этим
            функциональным блоком или посланы из него                         */

        /*  определение и декларация данных; такие данные являются глобальными для
            всех процессов, принадлежащих данному модулю                      */

        Process name:PROCESS ( );

            /*  определение и декларация локальных данных                     */
            nextstate:=...;
            join:=none;
            DO FOR EVER;
                state_loop: CASE nextstate OF
                /*      цикл по переменной nexstate, указывающей SDL - состояние */

                    (state_label1): RECEIVE CASE
                                        (signal name1):
                                        . . .
                                        (signal namen):
                ESAC state_loop;
                DO WHILE join/=none;
                    CASE join OF
                        (join - lab1): join: = none;
                        . . .
                        (join -labm): join: = none;
                    ESAC;
                OD;
            OD;
        END process - name;
END Functional - Block;
```

РИСУНОК D-9.1.1

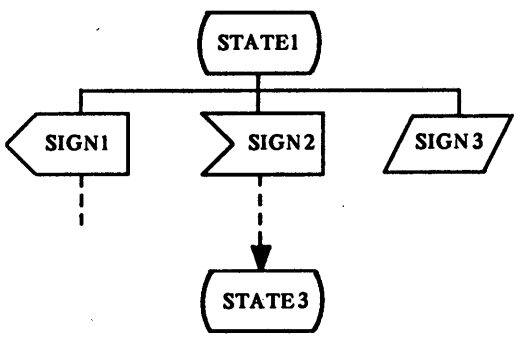
Общая структура CHILL-программы, полученной из SDL-диаграммы

Модуль, являющийся функциональным блоком, представляет поведение (процедурную часть) SDL-процессов.

При такой схеме отображения каждый SDL-процесс представляется в виде бесконечного цикла; переменная, названная "nextstate" (следующее состояние), задает состояние, подлежащее рассмотрению, а переменная, названная "join" (присоединение), задает возможные точки присоединения, определяющие общие наборы предложений.

Выбор с помощью конструкции "case" языка CHILL осуществляется, исходя из значения переменной "nextstate"; каждая статья конструкции "case" идентифицирует SDL-состояние. В каждой статье делается выбор среди возможных входных сигналов. Каждый входной сигнал определяет набор действий, подлежащих выполнению ("ветвь перехода").

Каждая ветвь перехода завершается присвоением либо переменной "nextstate", тем самым непосредственно определяя следующее состояние, либо присвоением значения переменной "join". Последующий цикл выбора по текущему значению переменной "join" делает возможным завершение, в смысле SDL, каждого перехода; и под конец переменной "nextstate" присваивается значение.



a) SDL

```
STATE1:
  RECEIVE CASE
    (SIGN1): ...
             ...
    (SIGN2): ...
             NEXTSTATE:=STATE3;
  ELSE GETOUT (LIST1)
  ESAC STATE1;
```

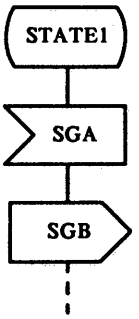
b) CHILL

РИСУНОК D-9.1.2

Пример отображения концепций STATE/INPUT/SAVE/NEXTSTATE

Одной из основных трудностей сопоставления SDL и CHILL является различная семантика приема сигнала; действительно, в то время как CHILL не воспринимает (и, следовательно, не уничтожает) ни одного сигнала помимо ожидаемых (устойчивые сигналы), SDL-процесс воспринимает все поступившие сигналы и уничтожает сигналы, не ожидавшиеся в этом состоянии. Семантическое несоответствие было разрешено введением встроенной подпрограммы GETOUT в качестве альтернативы (ветвь ELSE) в CHILL-конструкции RECEIVE CASE, как это показано на рис. D-9.1.2. Встроенная CHILL-подпрограмма GETOUT, которой известен (с помощью параметров) список входных и сохраняемых сигналов, уничтожает прочие сигналы, доступные процессу, когда эта подпрограмма вызывается.

После выполнения подпрограммы GETOUT механизм выбора состояния устанавливается на повторение цикла для этого состояния, пока не будет выбран входной сигнал (или пока он не прибудет, если его еще нет).



a) SDL

```
STATE1:
  RECEIVE CASE
    (SGA): pi := get_instance_value();
           send SGB to pi;
           nextstate := ... ;
    ELSE nextstate := state1;
  ESAC STATE1;
```

b) CHILL

РИСУНОК D-9.1.3

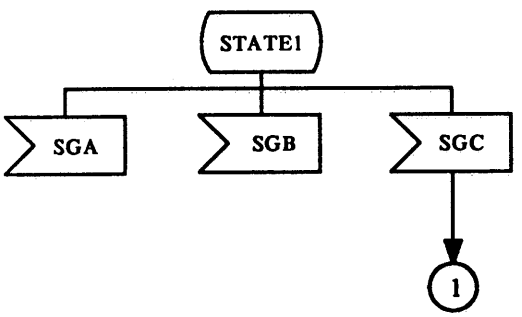
Пример отображения концепции OUTPUT

Например, после обнаружения входного сигнала SGA (на рис. D-9.1.3) осуществляется выбор экземпляра приемного процесса и посылается сигнал SGB.

Может оказаться, что перед посылкой сигнала SGB требуется заполнить какие-то информационные поля, которые будут пересланы вместе с сигналом. Это может быть выполнено либо непосредственно перед самой посылкой сигнала, либо задолго до того.

Когда на диаграмме встречается точка присоединения (см. рис. D-9.1.4), то переменной "JOIN" присваивается надлежащее значение. Как объяснялось на рис. D-9.1.1, выполняется цикл по значению переменной "join" с целью определить подлежащее анализу следующее состояние. С точки зрения языков программирования точка присоединения может рассматриваться как конструкция "goto"; если собрать все точки присоединения вместе, с тем чтобы их можно было проверить, то полный скелет программы может быть сформирован без использования "goto", что облегчит чтение программы.

Как показано на рис. D-9.1.5, принятие решения в языке SDL может быть непосредственно переведено в конструкцию "case" языка CHILL.



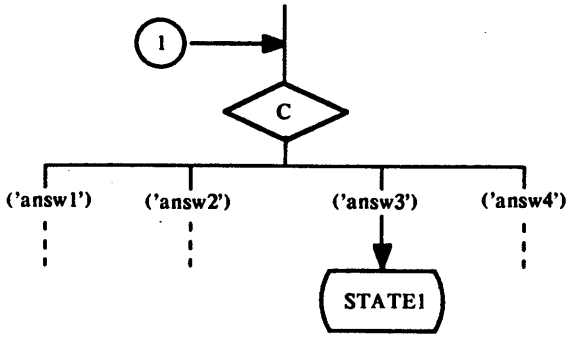
a) SDL

```
STATE1:
  RECEIVE CASE
    (S1 in m): case m.id of
                 (SGA): ... ;
                 (SGB): ... ;
                 (SGC): ... ; JOIN:=1;
                 ELSE nextstate ; := state1;
               esac;
  ESAC STATE1;
```

b) CHILL

РИСУНОК D-9.1.4

Пример отображения присоединения



a) SDL

```
...
1: CASE C OF
   ('answ1'): ... ;
   ('answ2'): ... ;
   ('answ3'): nextstate :=state1;
   ('answ4'): ... ;
   ESAC 1;
```

b) CHILL

РИСУНОК D-9.1.5

Пример отображения концепции DECISION

D.9.2 *Отображения между GR и PR*

С учетом ограничений на макрокоманды, отмеченных в § D.5.1, GR-формы всегда могут быть отображены на PR и наоборот.

Примеры эквивалентных спецификаций в обеих формах приводятся на протяжении всего этого текста.

D.10 *Примеры приложений*

D.10.1 *Введение*

Параграф D.10 содержит некоторые примеры использования SDL. Примеры взяты из области применения телекоммуникации; при этом была сделана попытка привести реальные примеры и охватить как можно больше концепций SDL.

Приведенные примеры имеют целью иллюстрацию использования SDL, но не являются международной спецификацией.

D.10.2 *Концепция сервисов*

Нижеследующая система является иллюстрацией концепции сервисов. В системе для этого примера особый интерес представляют три "функции системы". Этими функциями являются: функция управления трафиком, функция сопровождения и функция сбоев. На нижеследующем рисунке D-10.2.1 можно увидеть, как функции системы компонуются из подчиненных сервисов пяти различных процессов в пяти блоках.

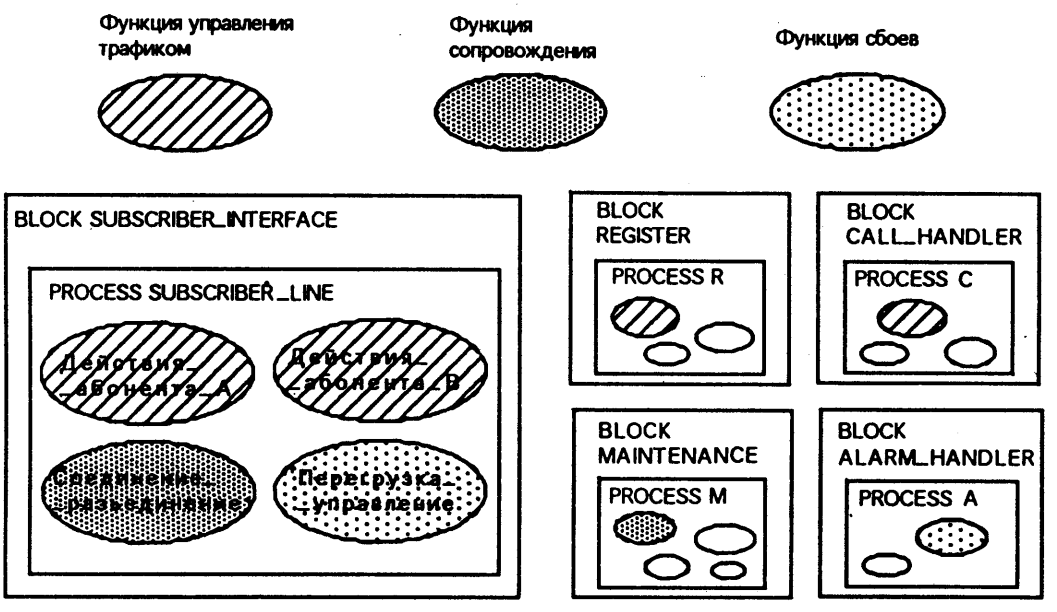


РИСУНОК D-10.2.1

Компоновка функций

Функция управления трафиком содержит установку и окончание телефонного вызова.

Этот пример не является полной документацией функций уровня системы; это всего лишь описание четырех сервисов процесса 'SUBSCRIBER_LINE'.

Нижеследующие три рисунка являются диаграммой системы (рис. D-10.2.2), диаграммой блока 'SUBSCRIBER_INTERFACE' (рис. D-10.2.3) и диаграммой процесса 'SUBSCRIBER_LINE' (рис. D-10.2.4). На этих диаграммах можно увидеть требуемые каналы, маршруты сигналов и сигналы.

SYSTEM Концепция_сервисов

```
/* Настоящая система является примером, иллюстрирующим концепцию сервисом SDL.
Для иллюстрации выбран блок SUBSCRIBER_INTERFACE. Диаграмма системы
изображает взаимодействующие блоки и каналы, требующиеся для сервисов в блоке
SUBSCRIBER_INTERFACE*/

SIGNAL A_OFF_HOOK,A_ON_HOOK,DIALLED_DIGIT,B_ON_HOOK,B_OFF_HOOK,
  DIAL_TONE,CONG_TONE,DIAL_TONE_OFF,CONG_TONE_OFF,RING_TONE_TO_A,
  RING_TONE_A_OFF,RING_SIG_TO_B,RING_SIG_B_OFF,CONNECT_DIG_REC,DIGIT,
  DISCONN_DIG_REC,CONNECTION,CONGESTION,FETCH_NEXT_DIGIT,
  CONNECT_CALL_HANDLER,A_OFF,A_ON,LINE_CONNECTED,LINE_DISCONNECTED,
  SEND_RING_TONE,B_ANSWER,DISC_A,RING_SIG,DISC_B,SEND_ALARM,
  CEASE_ALARM,CONN_REQ_ACK1,CONN_REQ_ACK2,DISC_REQ_ACK1,
  DISC_REQ_ACK2,CONN_REQ,DISC_REQ,B_ON,B_OFF;

SIGNALLIST L1A = A_OFF_HOOK,A_ON_HOOK,DIALLED_DIGIT;
SIGNALLIST L1B = B_ON_HOOK,B_OFF_HOOK;
SIGNALLIST L2A = DIAL_TONE,CONG_TONE,DIAL_TONE_OFF,CONG_TONE_OFF,
                 RING_TONE_TO_A,RING_TONE_A_OFF;
SIGNALLIST L2B = RING_SIG_TO_B,RING_SIG_B_OFF;
SIGNALLIST LAL = SEND_ALARM,CEASE_ALARM;
SIGNALLIST L1MAIN = CONN_REQ,DISC_REQ;
SIGNALLIST L2MAIN = CONN_REQ_ACK1,CONN_REQ_ACK2,DISC_REQ_ACK1,
                    DISC_REQ_ACK2;
SIGNALLIST L1REG = CONNECT_DIG_REC,DIGIT,DISCONN_DIG_REC;
SIGNALLIST L2REG = CONNECTION,CONGESTION,FETCH_NEXT_DIGIT;
SIGNALLIST L1CALL = SEND_RING_TONE,B_ANSWER,DISC_A;
SIGNALLIST L2CALL = A_OFF,A_ON;
SIGNALLIST L3CALL = RING_SIG,DISC_B;
SIGNALLIST L4CALL = LINE_CONNECTED,LINE_DISCONNECTED,B_ON,B_OFF;
```

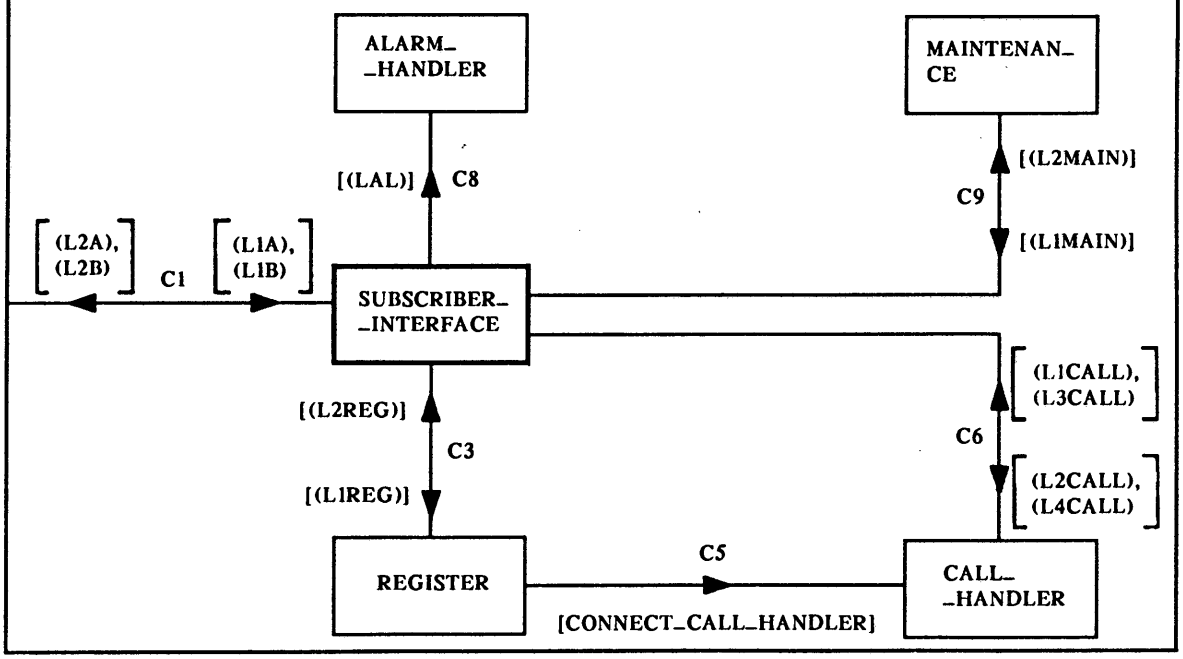


РИСУНОК D-10.2.2

Диаграмма системы

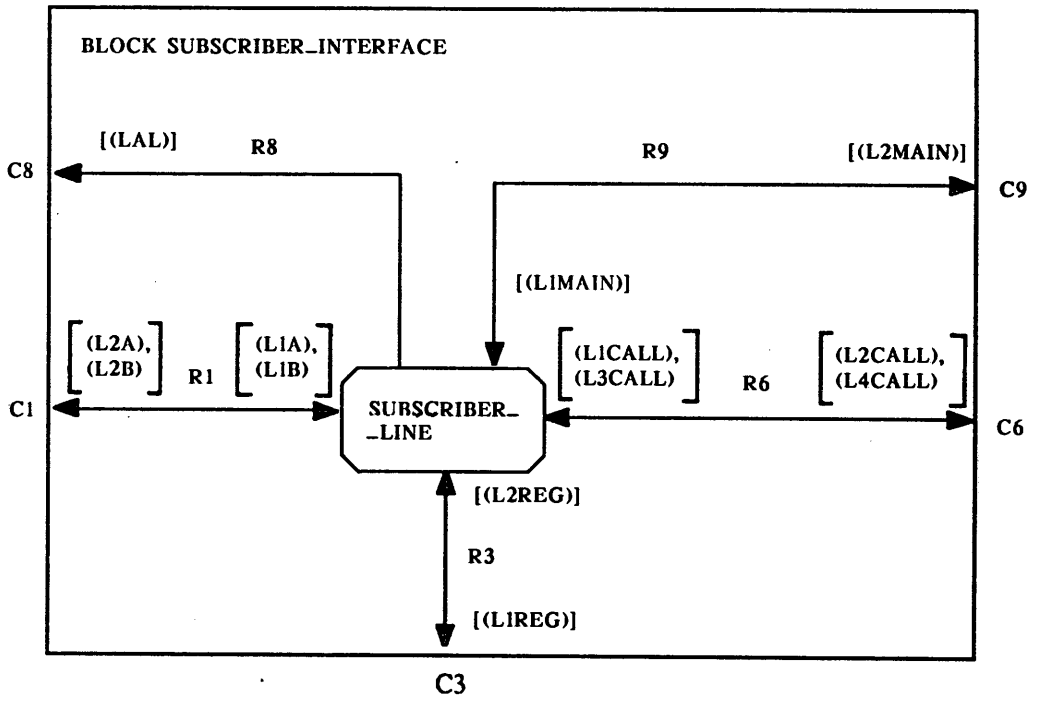


РИСУНОК D-10.2.3

Диаграмма блока

Как видно из диаграммы процесса (рис. D-10.2.4), сервисы взаимодействуют с помощью приоритетных сигналов, передаваемых по маршрутам сигналов IR01, IR02, IR03 и IR04. Но, кроме того, сервисы взаимодействуют и влияют друг на друга посредством 'глобальной' переменной 'Connected', объявленной в процессе.

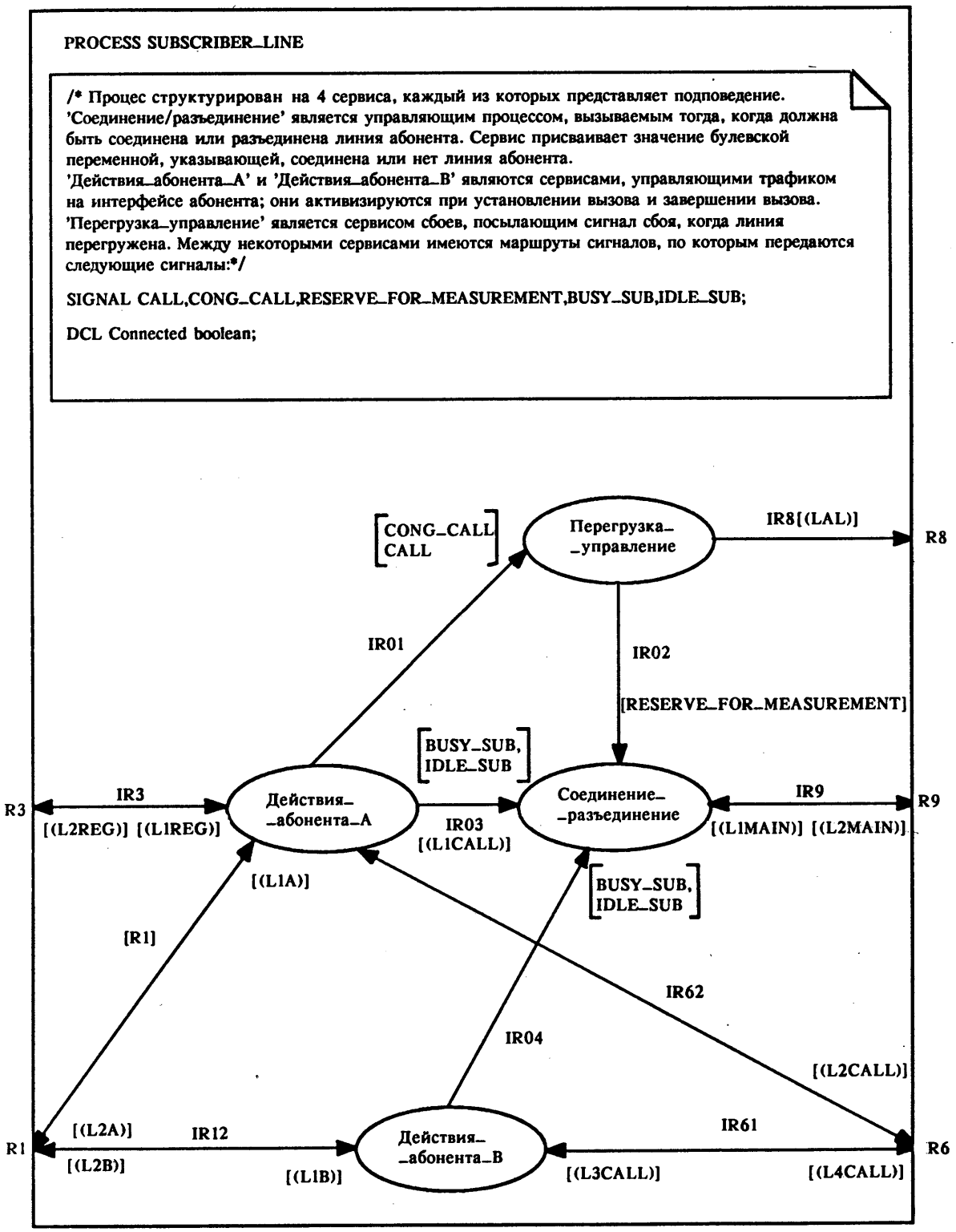


РИСУНОК D-10.2.4

Диаграмма процесса

Для лучшего понимания сервисов и взаимодействия между блоками в пример включено несколько временны́х схем.

Первые две временны́е схемы представляют нормальный случай взаимодействия блоков в течение вызова. Взаимодействие при перегрузке регистра показано на третьей схеме. Для упрощения схем предполагается, что между посылкой и получением сигнала нет никакой задержки.

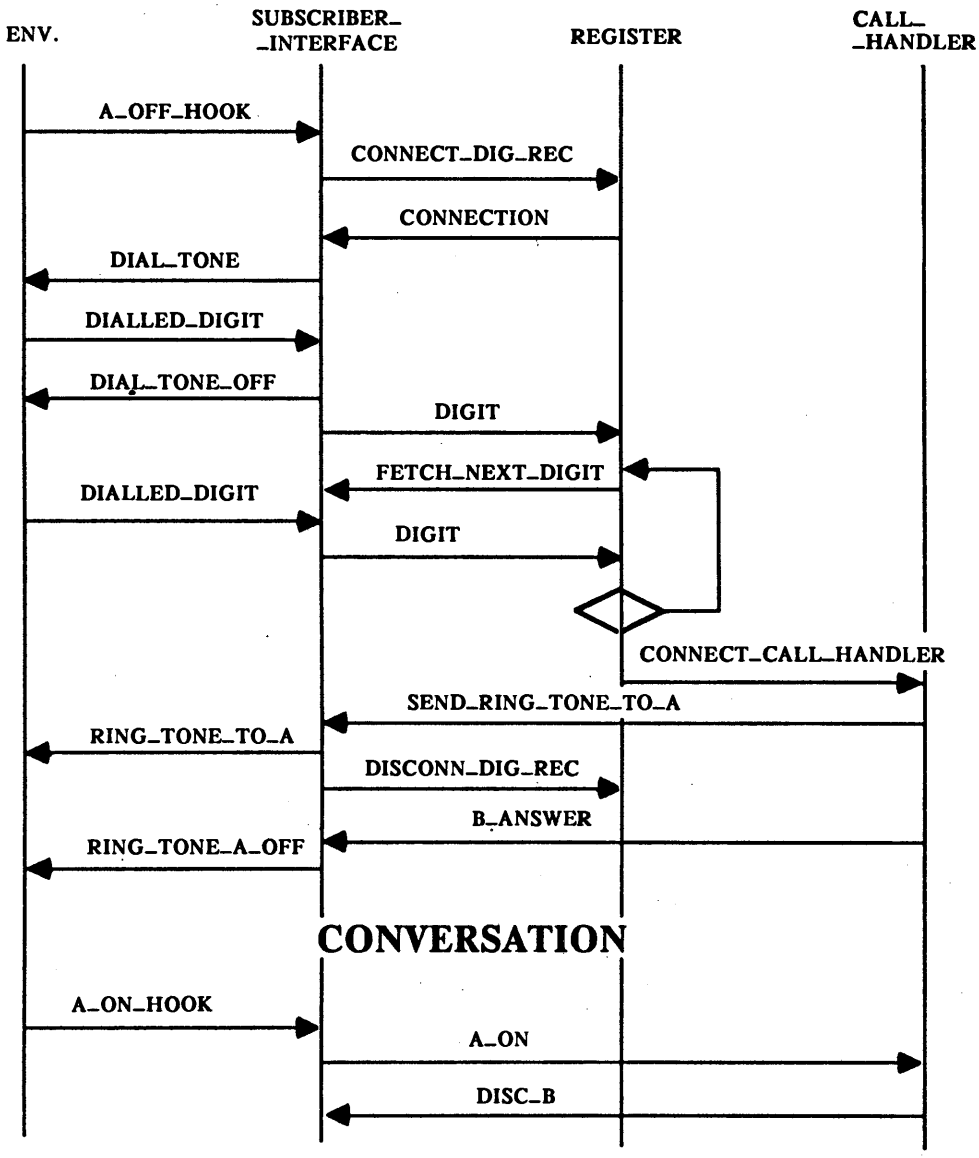
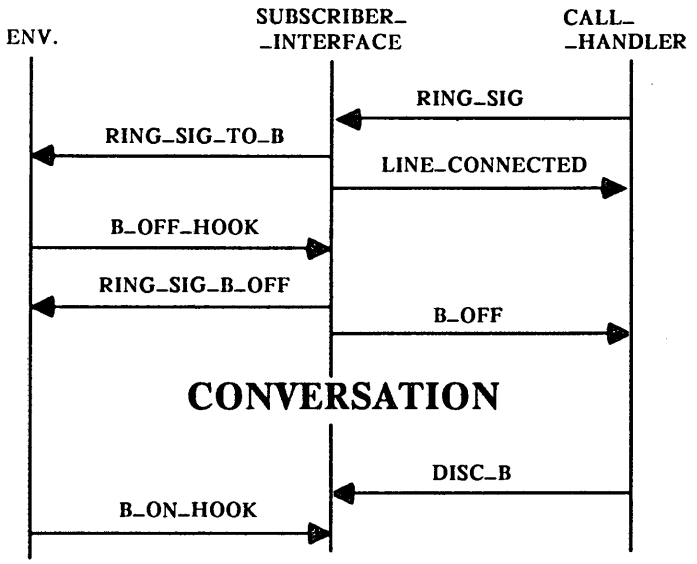


РИСУНОК D-10.2.5

Временна́я схема. Нормальный случай взаимодействия блоков. Необходимые сигналы для стороны А-абонент



Примечание. — Временна́я схема. Взаимодействие блоков в нормальном случае. Необходимые сигналы для стороны B-абонент.

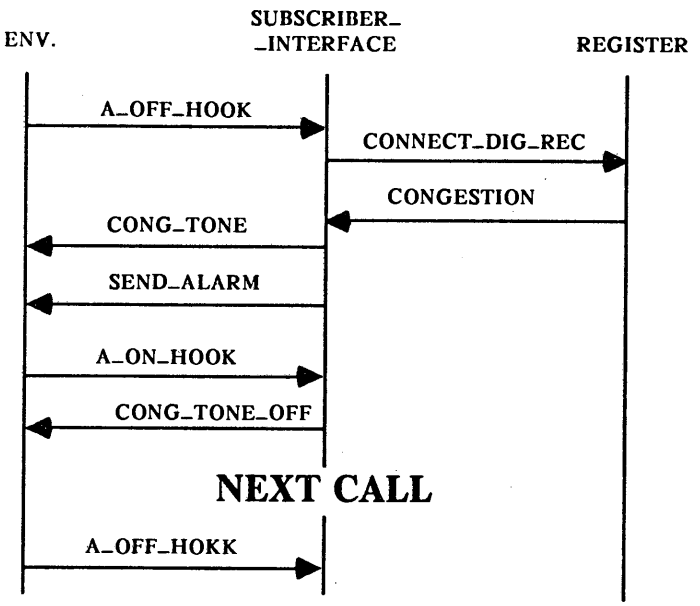
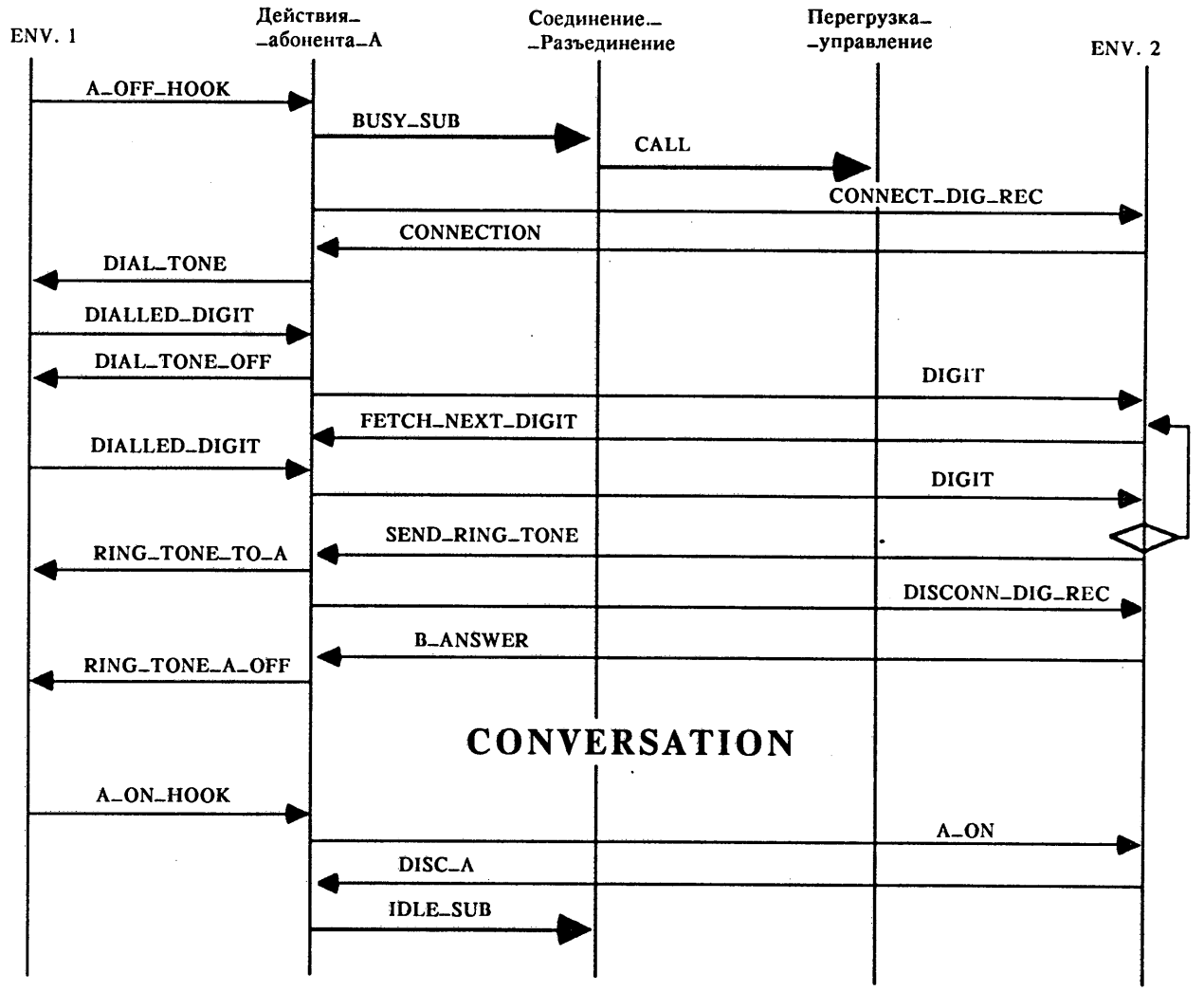


РИСУНОК D-10.2.6

Временна́я схема. Взаимодействие блоков при перегрузке регистра

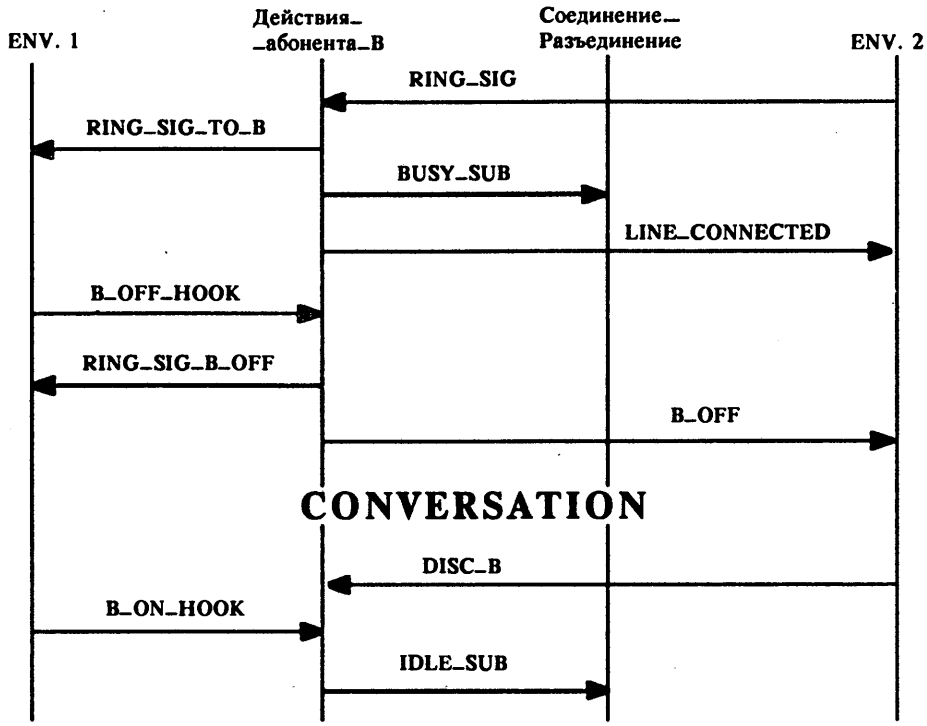
Взаимодействие между сервисами в процессе 'SUBSCRIBER_LINE' описано в нижеследующих диаграммах сервисов.



Примечание. — Приоритетные сигналы выделены более жирными линиями.

РИСУНОК D-10.2.7

Временна́я схема. Нормальный случай взаимодействия сервисов. Необходимые сигналы для стороны A-абонент



Примечание. — Приоритетные сигналы выделены более жирными линиями.

РИСУНОК D-10.2.8

Временна́я схема. Нормальный случай взаимодействия сервисов. Необходимые сигналы для стороны B-абонент

Поведение каждого из сервисов процесса 'SUBSCRIBER_LINE' описывается в четырех диаграммах сервисов (рисунки D-10.2.9–D-10.2.15).

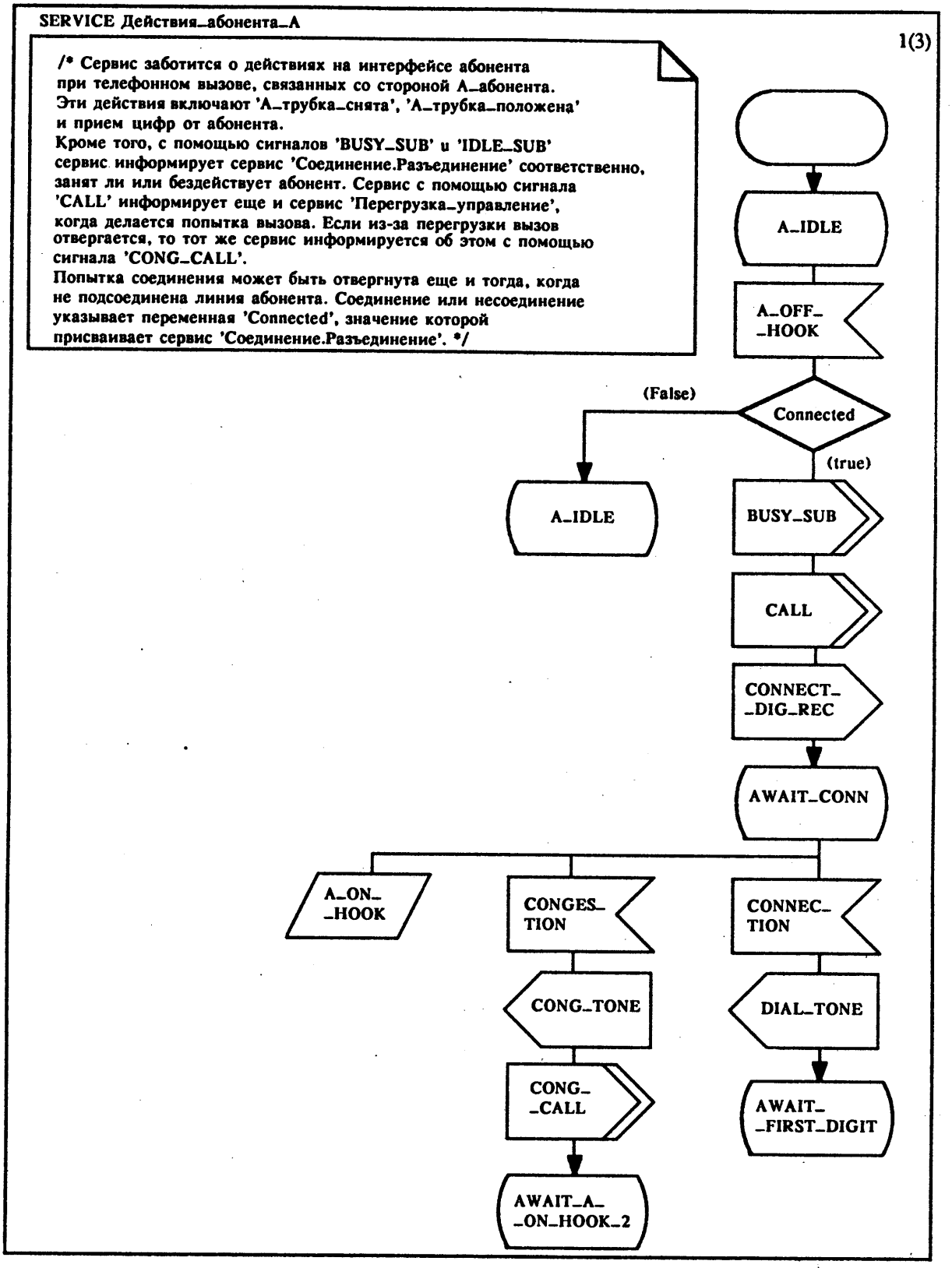


РИСУНОК D-10.2.9

Диаграмма сервиса

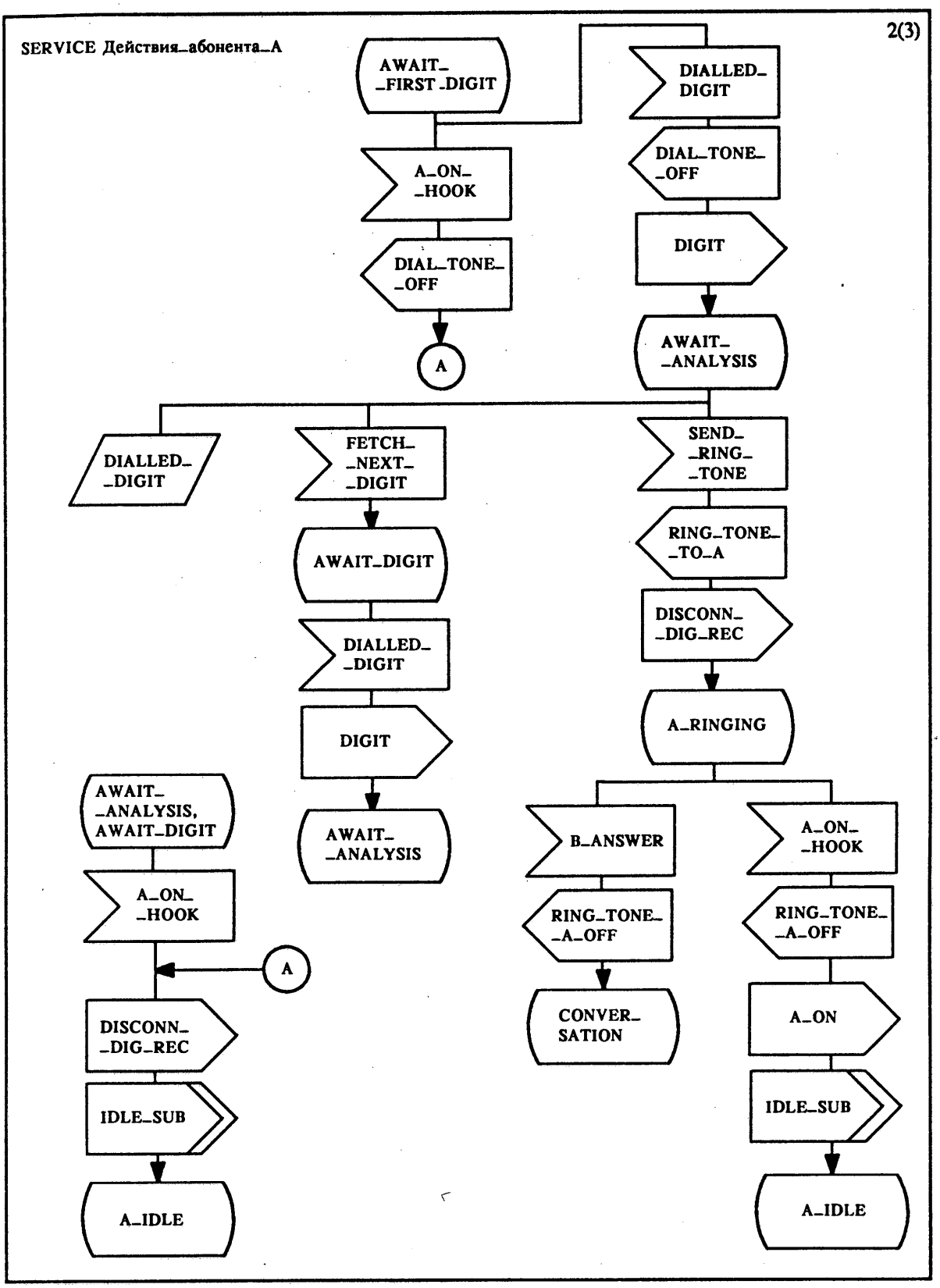


РИСУНОК D-10.2.10

Диаграмма сервиса

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.10

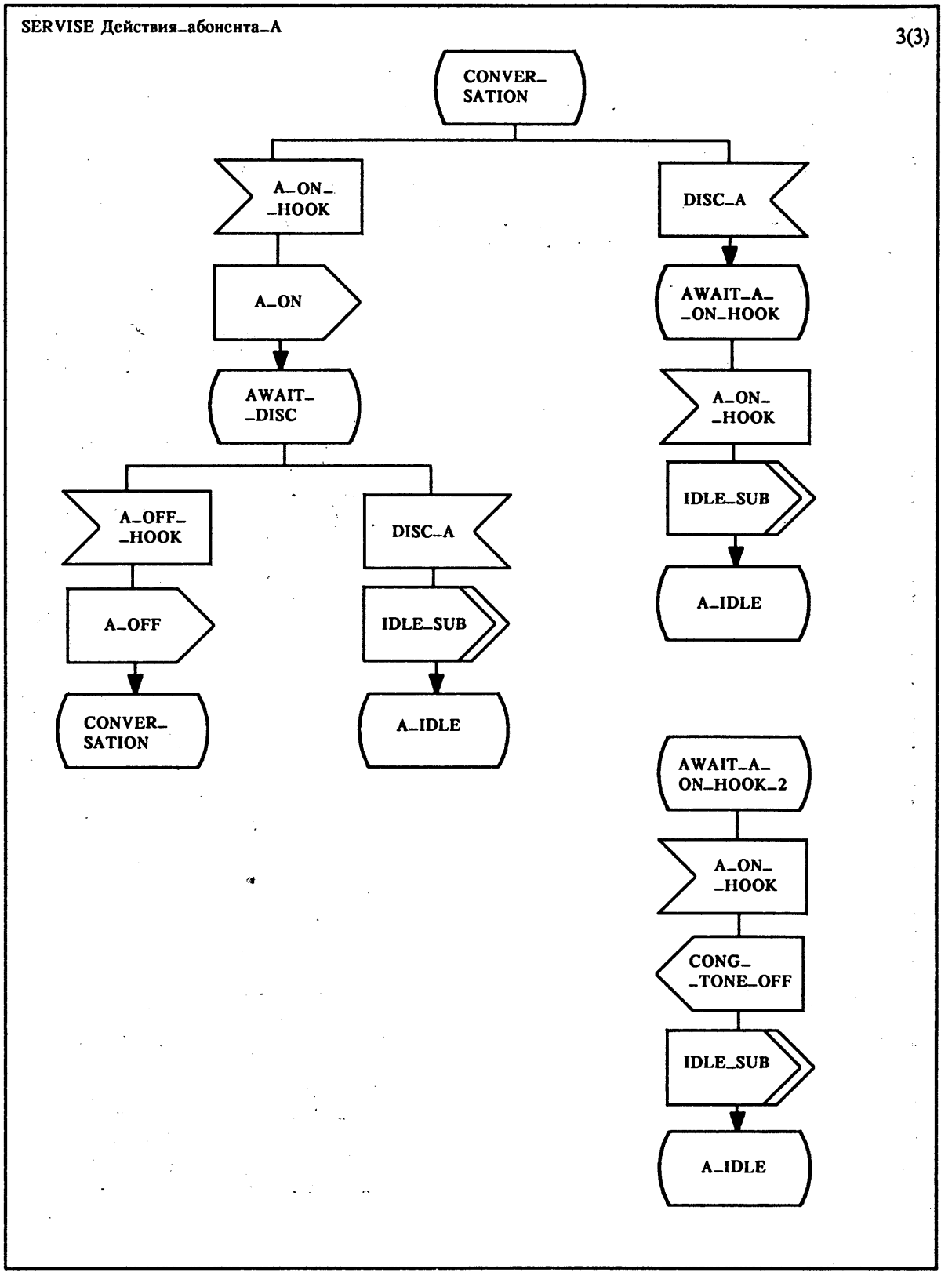


РИСУНОК D-10.2.11

Диаграмма сервиса

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.10

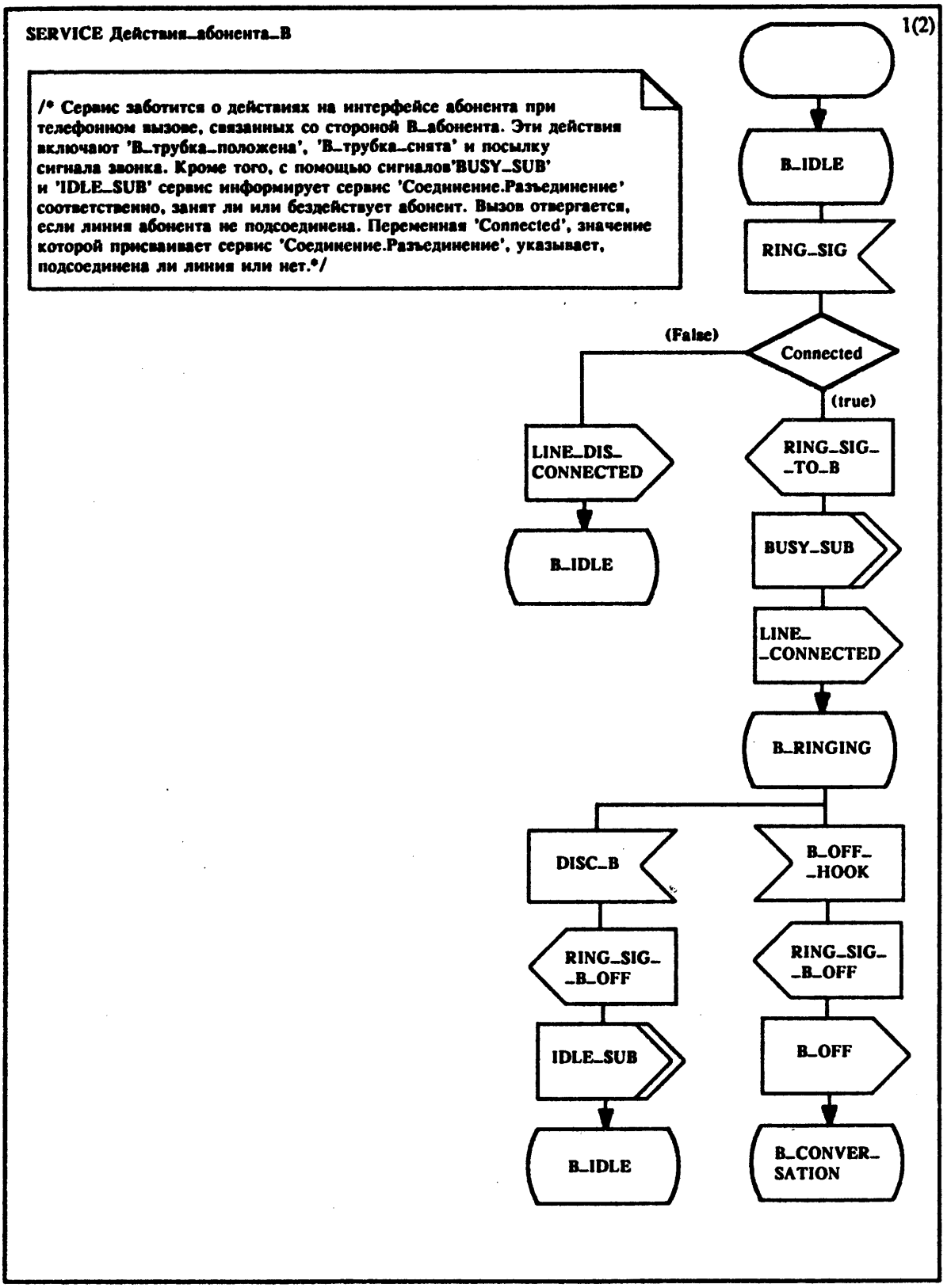


РИСУНОК D-10.2.12

Диаграмма сервиса

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.10

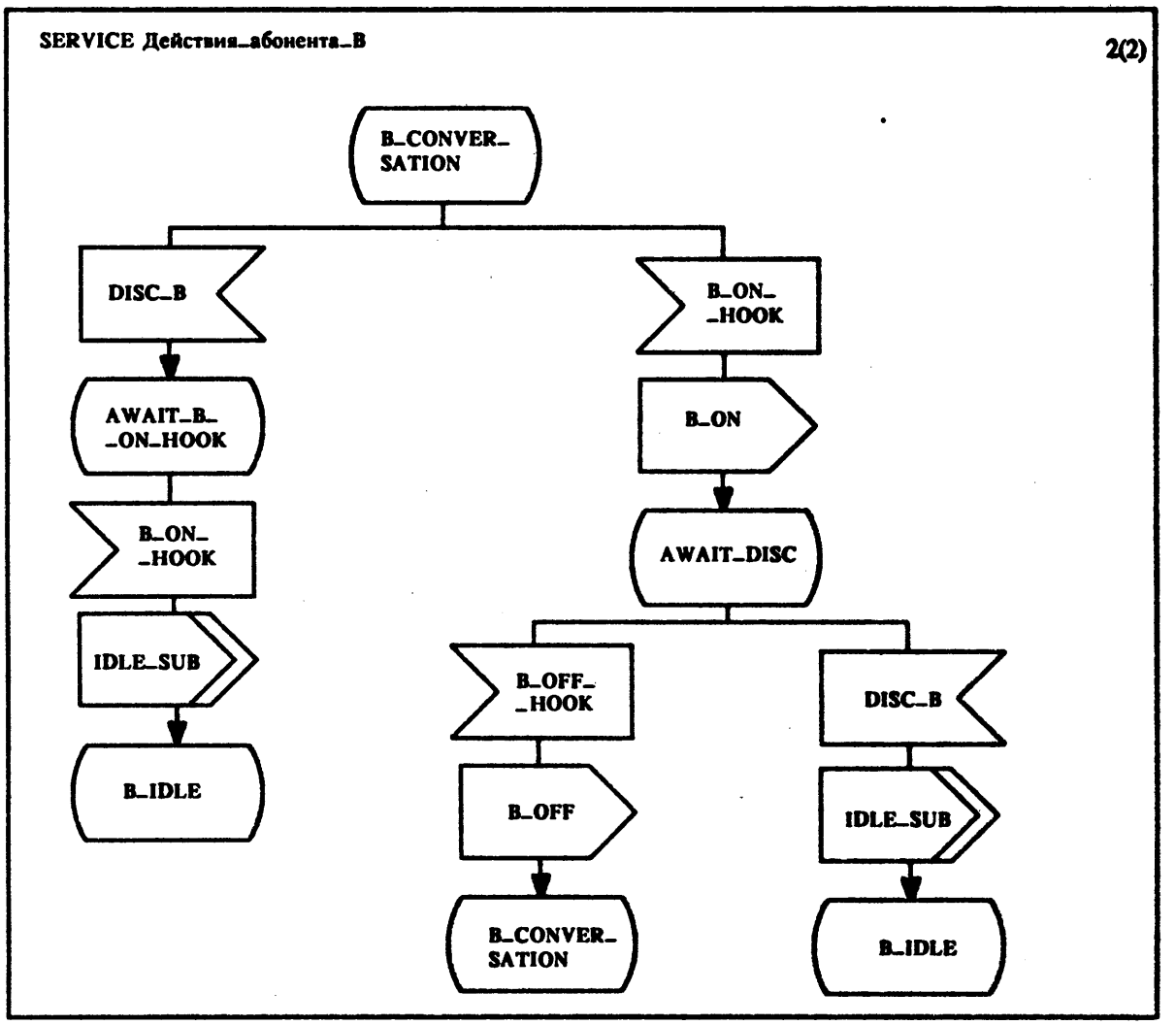


РИСУНОК D-10.2.13

Диаграмма сервиса

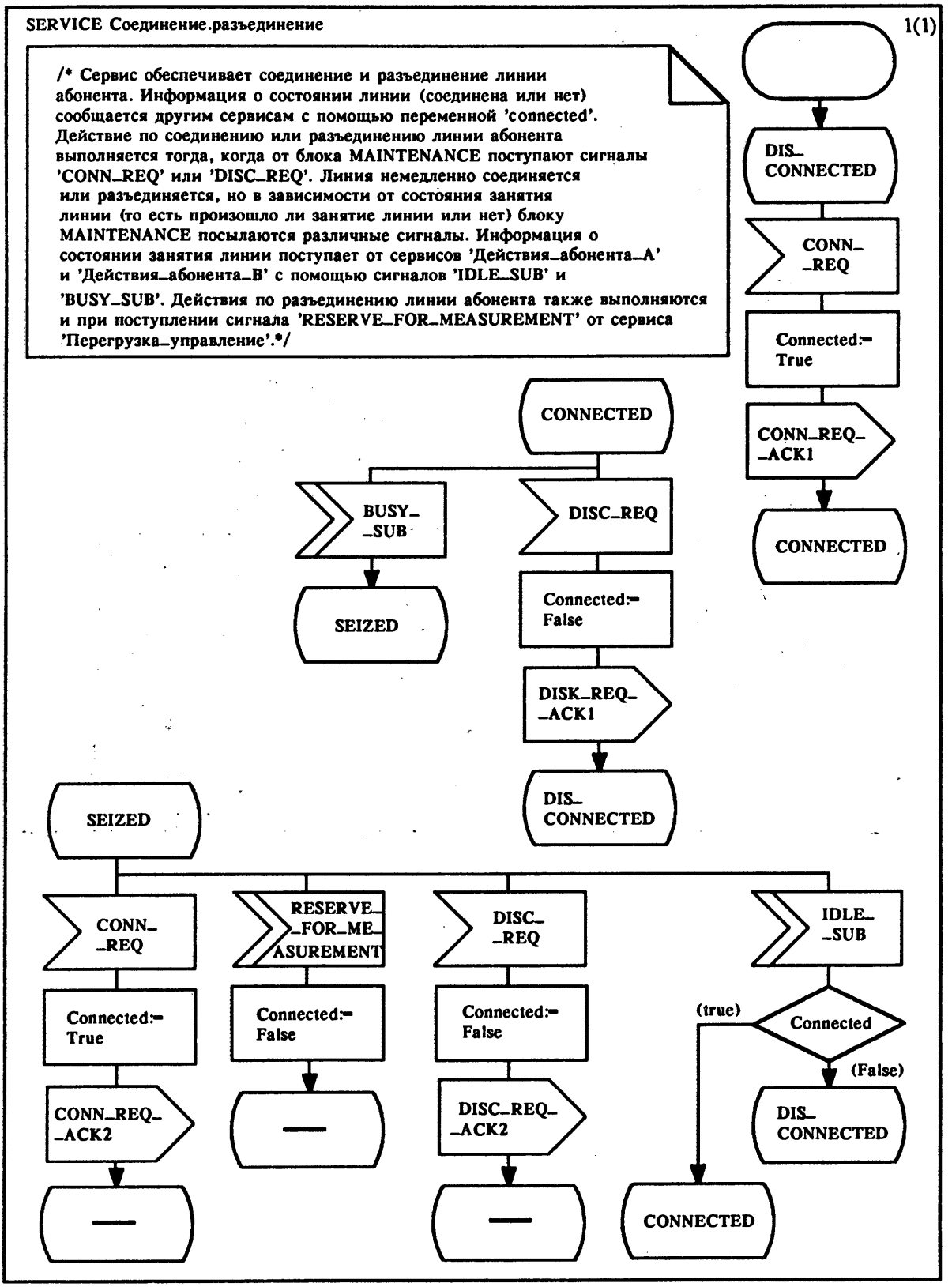


РИСУНОК D-10.2.14

Диаграмма сервиса

Рекомендация по SDL — Приложение D: Руководство пользователя — § D.10

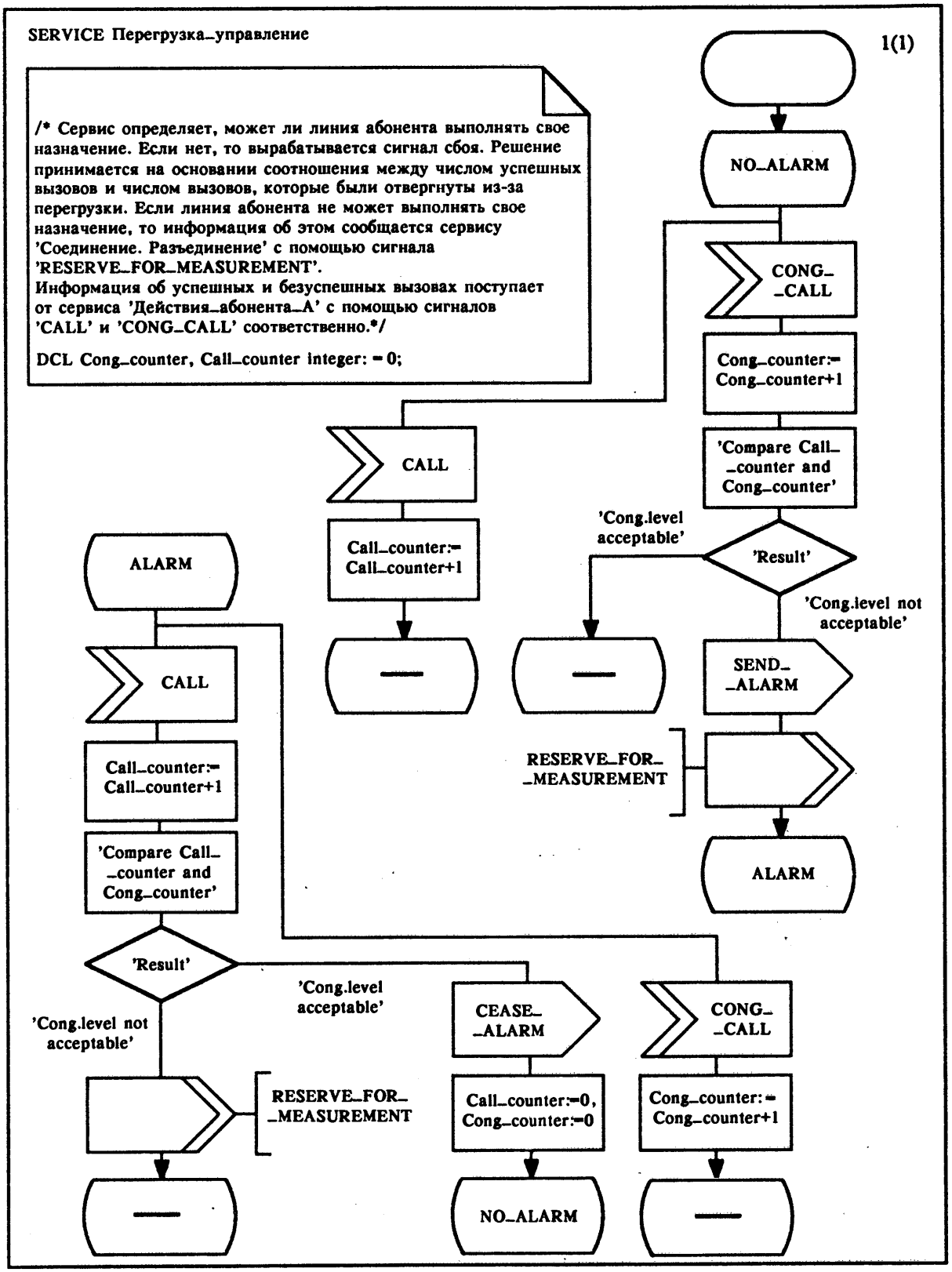


РИСУНОК D-10.2.15

Диаграмма сервиса

D.11 *Инструментарий SDL*

D.11.1 *Введение*

В настоящем параграфе обрисовывается инструментарий, который может быть полезен в SDL. Этот инструментарий может помочь при производстве документов, SDL-диаграмм (SDL/GR) или предложений (SDL/PR) и/или при проверке SDL-спецификации на действительность.

В настоящем описании нет исчерпывающего списка всего возможного инструментария. Требующийся инструментарий зависит от метода, выбранного пользователем.

В принципе SDL может использоваться без каких-либо вспомогательных средств. Однако сложность, изначально присущая современным системам, настолько велика, что очень часто SDL-спецификация оказывается весьма сложной. Поэтому для многих систем требуются автоматические средства, помогающие обращаться со спецификацией, проектированием и документацией систем. Например, сложность и стоимость ручной разработки и возможного обновления графической документации на коммутируемую АТС были бы существенно снижены благодаря использованию подходящих вспомогательных средств.

С учетом вышеприведенных соображений разработка SDL велась таким образом, чтобы можно было эффективно использовать инструментарий, облегчающий применение SDL.

D.11.2 *Классификация инструментария*

Инструментарий SDL может быть классифицирован в соответствии с видами работ, осуществляемых при разработке SDL-документации; например:

– Инструментарий для ввода: в соответствии с синтаксическими формами имеются средства для графического SDL, текстовых фраз и рисунков.

– Инструментарий для синтаксической проверки: в него входят синтаксические анализаторы для каждого из двух синтаксисов.

– Инструментарий для производства документации: после того как SDL-документы введены в машину, специальные средства могут извлечь и репродуцировать их, причем, возможно, с помощью различных периферийных устройств. Они могут использовать синтаксическую форму, отличную от использованной при вводе документов. Более того, специальные средства могут выработать новые типы документов, выведенные из тех, которые были первоначально введены.

– Инструментарий для моделирования и анализа систем: SDL-документация, описывающая систему, может быть использована для абстрактного моделирования системы. Модели могут подвергаться проверке. На них можно осуществлять поиск тупиковых ситуаций, сравнивать различные модели одной и той же системы (например, спецификацию и описание), моделировать поведение системы и т.д.

– Инструментарий для разработки машинных программ: очень детальная SDL-спецификация может использоваться для облегчения создания программного обеспечения. Могут быть разработаны средства, которые под руководством программиста выработают программы полуавтоматическим способом.

Специальный, но полезный класс инструментария представлен

– обучающим инструментарием SDL: это средство может использоваться отдельно, либо быть интегрированным с прочим инструментарием. Такая интеграция позволяет пользователям других функций получить при необходимости помощь.

Поскольку SDL используется на многих различных стадиях жизненного цикла системы, то легко обозреть использование всех видов инструментария в полной среде целого проекта.

D.11.3 *Ввод документов*

На ввод текстовой формы SDL нет никаких специальных требований, так как с точки зрения ввода SDL/PR ничем не отличается от ввода любой цепочки символов. Поэтому такой ввод может пользоваться теми же средствами (редактором текста). Однако другой синтаксис предполагает наличие средства обработки графиков.

Очевидно, что, в то время как ввод SDL/PR очень полезен, весьма существенно наличие средств поддержки ввода SDL/GR, если мы намерены пользоваться этой формой синтаксиса.

Для таких функций, как связывание двух символов, перемещение набора символов в другую область страницы или на другие страницы и сцепленное удаление (удаление символа предполагает удаление всех соединений с данным символом) необходимо наличие графического редактора. Так же, как для SDL/PR, инструментарий для ввода SDL/GR должен быть смоделирован на семантику/синтаксис SDL. Таким образом он должен выявить недействительные соединения, подталкивать пользователя к заполнению незавершенных частей и т.п.

Перед инструментарием стоит несколько трудностей, вытекающих из физических ограничений графических устройств, таких как "разрешающая способность". Практически невоможно иметь достаточный набор таких знаков, которые и легко распознаются, и вместе с тем допускают изображение на экране разумного числа символов.

Требуется рассмотреть такие решения, как прокручивание или разворачивающееся окно, но они не являются вполне удовлетворительными. Высокая разрешающая способность не должна рассматриваться как "обязательная", если пользователь копирует диаграммы, но она весьма желательна, если диаграммы разрабатываются непосредственно самим пользователем. По тем же причинам (требования к общей обозримости и достаточному числу деталей) высокая разрешающая способность желательна при выводе диаграммы на экран.

Средства для ввода SDL/PR также могут быть полезны: они могут подсказывать пользователю возможные ключевые слова SDL/PR.

Кроме того, они могут строить непосредственно форматы SDL/PR, исходя из полученных ключевых слов, автоматически вводить разделители, предоставлять пользователю SDL/PR ориентированные функциональные ключи, обеспечивать ясное размещение текста и т.д.

Реализация такого инструментария может базироваться на имеющихся редакторах цепочек знаков, которые можно будет расширить так, чтобы они охватывали вышеупомянутые свойства.

D.11.4 *Верификация документов*

После того, как документы введены в машину, следующим шагом является их верификация. Сначала они должны быть подвергнуты индивидуальной верификации, после чего связанные диаграммы должны быть объединены и вновь подвергнуты верификации вплоть до верификации всей системы.

Если ввод был осуществлен с помощью инструментария, приспособленного к нуждам SDL, то большой объем проверок отдельных документов должен быть уже выполненным.

Ошибки, вытекающие из "невозможных" операций (например, вводы и сохранения, следующие за чем угодно, кроме состояний), должны быть все выявлены и исправлены в процессе ввода. Вместе с тем обнаружение некоторых ошибок возможно только после завершения фазы ввода как в отношении одного документа, так и, естественно, в отношении несогласованности между документами.

Некоторые из правил SDL могут быть проверены автоматически. Например, требование, согласно которому каждый вывод должен иметь соответствующий ввод.

Для случая многоуровневой спецификации в некоторой степени может быть проверена согласованность между уровнями.

Формальная модель SDL может быть использована для разработки комплекта процедур верификации.

D.11.5 *Репродуцирование документов*

Документы SDL, введенные в машину, должны подлежать выборке, изображению на экране и репродуцированию. Для всех этих действий требуется инструментарий. Возможно, что потребуется выборка только части или подмножества документов. Выборка может быть SDL-ориентированной, например, "извлечь все процессы, посылающие" заданный сигнал, или процессы, в состояниях которых выполняются конкретные действия и т.п. Инструментарий для выдачи информации на экран приобретает особый интерес, когда требуется выдача информации в графическом синтаксисе. Сюда относятся все соображения, высказанные при рассмотрении ввода документов в синтаксисе SDL/GR. Репродуцирование документов зависит от типа документов, подлежащих репродуцированию, от того, как были сохранены в машине эти документы, и от характеристик периферийных устройств для вывода. Оно может зависеть и от того, как были введены документы. Пользователям может потребоваться выдача документов в синтаксической форме, отличной от той, в которой они были введены.

На репродуцирование документов влияют ограничения выводных периферийных устройств. Например, диаграмма может быть слишком велика, чтобы поместиться на выделенном ей пространстве бумаги и поэтому должна быть разбита на части. Поэтому должны быть добавлены коннекторы и введены перекрестные ссылки. Может оказаться желательным отличать "добавления", сделанные инструментарием, от исходных вводных характеристик. Желательно наличие средств, обеспечивающих гибкость в формате вывода: в понятие "гибкость" включают различные размеры символов, различные форматы вывода, вертикальное или горизонтальное расположение и т.д.

Должна всегда иметься возможность репродуцировать документ точно в той форме, в какой он был введен.

D.11.6 *Генерация документов*

Отправляясь от документов SDL, введенных пользователем и сохраненных в машине, можно автоматически получить ряд других документов, в том числе:

- списки сигналов, сгруппированных по процессам, по блокам или по системе;
- диаграммы обзора состояний, представляющих граф процесса в виде набора состояний, соединенных дугами, соответствующими переходам;
- таблицу перекрестных ссылок, сгруппированных по процессам, по блокам или по системе;
- диаграмму дерева блоков, изображающую структуру блоков и уровней;
- поведение системы как ответ на последовательность внешних воздействий;
- указатели: однажды произведенные документы должны подлежать репродуцированию; при этом сохраняют силу все вышеприведенные соображения.

SDL-документы, введенные на SDL/GR, могут быть автоматически переведены в эквивалентную SDL/PR-форму и наоборот.

Необходимо принять во внимание следующие соображения:

- SDL/GR содержит визуальную информацию, отсутствующую в SDL/PR (она не существует в SDL/PR). Например, координаты символов лишены смысла в SDL/PR;

– коннекторы, соединяющие линии потока на различных листах, могут быть исключены.

Однако обратное преобразование из SDL/PR в SDL/GR значительно сложнее и вряд ли удовлетворит всех потенциальных читателей.

В силу двумерности представления на SDL/GR некоторые метки, введенные из-за последовательной структуры SDL/PR, могут быть удалены, так как будет достаточно соединяющих линий.

Обычно преобразование генерирует только модель SDL/GR-диаграммы. Эта модель содержит всю информацию, требующуюся инструментарию для форматирования и репродуцирования диаграммы на графическом устройстве.

Следует обратить внимание на то, что два разных транслятора, преобразующих некоторый SDL/PR в SDL/GR, могут привести к двум спецификациям с различным размещением символов. Полученные таким образом обе SDL/GR-спецификации правильны при условии, что они сохраняют семантику, заложенную в исходной спецификации.

D.11.7 *Моделирование и анализ системы*

SDL-документы, независимо от того, специфицируют они систему или описывают ее, являются главным образом моделью системы.

Модель, первоначальное назначение которой – передать информацию от одного человека другому, может быть, кроме того, интерпретирована с помощью специальных средств, проверена на согласованность, полноту (этот аспект может быть не удовлетворен для спецификаций, рассчитанных на спецификацию только некоторых частей системы), а также на корректность и соответствие правилам SDL (как описано в параграфе о верификации документов).

Кроме того, можно разработать средства, использующие модель, для моделирования функционального поведения системы. При таком моделировании система может взаимодействовать с окружающей средой и пользователи могут сделать заключение о соответствии модели их ожиданиям.

Если добавлена дополнительная информация, указывающая время, затрачиваемое на выполнение каждого действия, и осуществляющая разметку доступных ресурсов (очередей, экземпляров и т.д.), то путем моделирования можно изучить еще и мощность системы.

Можно разработать инструментарий, который, отправляясь от моделей систем, способен создавать модели окружающей среды, вырабатывающие содержательные последовательности сигналов для проверки реальных систем. Путем анализа ветвей можно будет обнаруживать тупиковые ситуации в модели.

Кроме того, модель системы может использоваться в качестве онлайновой документации. Если существуют надлежащие связи между фактической системой и хранилищем документации, то можно разработать инструментарий, прослеживающий на модели события системы реального времени.

Для достижения этого должна быть обеспечена корреляция между физическими событиями, как они видны системе, и логическими событиями, с которыми имеет дело SDL-документация. Если документация организована по нескольким уровням абстракции, то пользователь может выбрать уровень, который он желает проследить. Это может оказаться очень полезным, поскольку позволяет пользователям, имеющим разные степени подготовленности и опыта, проверить функционирование системы.

Инструментарий, интерпретирующий SDL-модель, может также оказаться полезным и для выделения разницы в поведении различных моделей одной и той же системы. Кроме того, то же средство может быть использовано для сравнения различных описаний системы (систем, разработанных различными компаниями), или для сравнения спецификации системы с описанием. Таким способом можно проверить, совпадает ли описание системы с исходной спецификацией.

D.11.8 *Генерация программ*

Обеспеченность формально определенного синтакса и формального математического определения SDL делают возможным создание средств, способных отображать семантику SDL-спецификаций на семантику языков программирования. Возможно, такие средства не способны обеспечить полностью реализуемые программы, но они могут быть весьма полезны в выработке по меньшей мере структуры фактических программ.

В параграфе D.9.1 настоящего Руководства Пользователя обрисован подход к осуществлению изображения между SDL и CHILL.

D.11.9 *Обучение*

Разработан полный курс обучения языку SDL; он охватывает все аспекты языка, обеспечивая одновременно примеры и предлагая несколько рекомендаций по использованию SDL.

Курс обучения языку SDL можно пройти в Международном союзе электросвязи (ITU, General Secretariat – Sales Section, Place des Nations, CH-1211 Geneve 20 (Switzerland).

ПРИЛОЖЕНИЕ

к русскому изданию Синей книги, том X, выпуск X.2

Переводы названий блоков, сигналов, состояний, переменных и принятий решений, использованных в §§ D.5.3.2; D.5.3.3.1; D.5.5.3; D.10.2

ACKN подтверждение
A_EXCHANGE A–коммутатор
A_IDLE A – бездействует
ALARM сбой, тревога
ALARM_HENDLER обработка сбоя, обработка сигнала травли
A_OFF A–выключен
A_OFF_HOOK A–трубка снята
A_ON A–включен
A_ON_HOOK A–трубка положена
A_RINGING A–вызов
AWAIT_ANALYSIS ожидание проведения анализа
AWAIT_A_ON_HOOK ожидание, пока будет положена A–трубка
AWAIT_A_ON_HOOK_2 ожидание, пока будет положена A–трубка (2)
AWAIT_B_ON_HOOK ожидание, пока будет положена B–трубка
AWAIT_CONN ожидание соединения
AWAIT_DIGIT ожидание поступления цифры
AWAIT_DISC ожидание разъединения
AWAIT_FIRST_DIGIT ожидание поступления первой цифры
B_ANSWER B–ответ
B_CONVERSATION B–диалог
B_IDLE B–бездействие
B_NUMBER B–номер
B_OFF B–выключен
B_OFF_HOOK B–трубка снята
B_ON B–включен
B_ON_HOOK B–трубка положена
B_RINGING B–звонок вызова
BUSY_SUB абонент занят
CALL вызов
Call_counter счетчик вызовов
CALL_HANDLER обработчик вызовов
CLEAR освобождение
Compare Call_counter and Cong_counter сравнить "счетчик вызовов" и "счетчик перегрузок"
CONG_CALL перегруженный вызов
Cong-counter счетчик перегрузок
CONGESTION перегрузка
Cong level acceptable допустимый уровень перегрузки
Cong level not acceptable недопустимый уровень перегрузки
CONG_TONE тон перегрузки
CONG_TONE_OFF тон перегрузки выключен
CONNECT_CALL_HANDLER обработчик соединений вызовов
CONNECT_DIG_REC запрос цифры соединения
CONNECTED соединен
Connected соединен
CONNECTION соединение
CONN_REC запрос на соединение
CONN_REC–ACK1 подтверждение запроса на соединение (1)
CONN_REC_ACK2 подтверждение запроса на соединение (2)
CONV диалог
CONVERSATION диалог
DIALLED_DIGIT цифра, набранная по номеронабирателю
DIAL_TONE сигнал готовности станции
DIAL_TONE_OFF включение сигнала готовности станции
DIGIT цифра
DISC_A A–разъединение
DISC_B B–разъединение
DISCON_DIG_REC запрос цифры разъединения
DISCONNECTED разъединен
DISC_REC запрос на разъединение
DISC_REC_ACK1 подтверждение запроса на разъединение (1)
DISC_REC_ACK2 подтверждение запроса на разъединение (2)
ENV окружающая среда
FETCH_NEXT_DIGIT выборка следующей цифры
IDLE бездействие
IDLE_SUB абонент бездействует
LINE_CONNECTED линия соединена
LINE_DISCONNECTED линия разъединена
MAINTENANCE сопровождение
NEXT CALL следующий вызов
NO_ALARM нет сбоя
OFF_HOOK трубка снята
ON_HOOK трубка положена
REGISTER регистр
RESERVE_FOR_MEASUREMENT резервирован для измерений
Result результат
RING_SIG_B_OFF B–сигнал контроля посылки вызова выключен
RING_SIGN сигнал контроля посылки вызова
RING_SIG_TO_B сигнал контроля посылки вызова направлен к B
RING_TONE зуммер контроля посылки вызова
RING_TONE_A_OFF A–зуммер контроля посылки вызова выключен
RING_TONE_TO_A зуммер контроля посылки вызова направлен к A
SEIZE занятие (захват)
SEIZE_ALARM сбой занятия
SEIZED занятие произошло, захвачен
SEND–ALARM сообщение о сбое
SEND_RINC_TONE посылка зуммера контроля посылки вызова
SET_UP установление
SIGNAL_SEQUENCE последовательность сигналов
SUBSCRIBER_INTERFACE интерфейс абонента
SUBSCRIBER_LINE линия абонента
SUPERVISION управление

Printed in USSR • 1990 — ISBN 92-61-03764-X