Лабораторная работа 2

Построение функциональной модели телекоммуникационной системы с помощью пакета PragmaDev Studio

Цель: Изучить принцип построения функциональной модели системы и алгоритм ее реализации **с помощью пакета PragmaDev Studio.**

Задание:

1. С помощью раздела 2 в методических указаниях выполнить демонстрационный пример, который реализуете в проекте, сделанном в лабораторной работе 1 с помощью пакета PragmaDev Studio. Созданный проект сохранить для использования при выполнении заданий лабораторной работы 3.
2. Выполнить индивидуальное задание – используя графические средства языка SDL, построить модель конечного автомата по известной матрице состояний и сигналов этого автомата. Матрицу выбираете по варианту, соответствующему последней цифре пароля (если 10, то вариант 0).
3. С помощью пакета PragmaDev Studio создать проект, в котором реализовать построенную модель как диаграмму процесса и провести ее синтаксическую проверку.
4. Результаты выполнения задания оформить в виде отчета.

1. Методические указания к выполнению лабораторной работы

Под функциональным описанием системы будем понимать описание действий, выполняемых отдельными компонентами системы, включая их взаимодействие между собой посредством выдачи и получения дискретных порций информации. С точки зрения языка SDL основным функциональным компонентом, определяющим поведение системы, является процесс. В свою очередь, за этим термином стоит понятие автомата с конечным числом состояний (Finite State Machine – FSM).

Автомат с конечным числом состояний – это математический объект, который обладает следующими свойствами:

- имеется множество дискретных состояний, и в любой момент времени автомат пребывает в одном из этих состояний;
- на входе автомата возникают некоторые сигналы (говорят также «наступают события);
- при поступлении входного сигнала автомат мгновенно переходит в другое состояние (в частном случае оно может совпадать с текущим) и одновременно выдает некоторый выходной сигнал;
- для каждого состояния и для каждого входного сигнала однозначно известно новое состояние, в которое перейдет автомат, и ответный сигнал, который появится на выходе.

В отличие от обычного автомата (FSM), обобщенный (расширенный) автомат (Extended Finite State Machine – EFSM) имеет целый ряд особенностей:

1) Переход из одного состояния в другое происходит не мгновенно, а занимает некоторый промежуток времени.
2) Как следствие, на входе процесса образуется очередь сигналов, поэтому важным фактором становится дисциплина (правила) выборки сигналов из очереди.
3) Во время перехода из одного состояния в другое процесс может производить вычисления, а также выполнять целый ряд других действий: работа с таймерами, выдача выходных сигналов, порождение других процессов, вызов процедур и др.
4) Совершая переход, процесс может заниматься проверкой некоторых условий и, в зависимости от результата проверки, изменять дальнейший порядок действий, т.е. направление перехода в другое состояние.

Именно эти особенности приближают довольно абстрактное понятие FSM к реальным вычислительным процессам в системах управления сложными распределенными объектами, компоненты которых взаимодействуют с помощью дискретных сигналов.

Функционирование процесса удобнее всего описывать с помощью графических средств. Такое описание (в языке SDL его называют диаграммой) во многом напоминает обычный граф, в котором вершины соответствуют состояниям процесса, а линиями (ребрами) обозначаются возможные переходы между состояния. Кроме того, применяются и дополнительные графические элементы для разных видов действий, выполняемых в фазе перехода (табл. 1).

Таблица 1

Графический символ	Название символа	
	Пакет PragmaDev Studio	Rec. ITU-T Z.101
	Process	Process
	Process start	Start
	State	State
	Message input	Input
	Message output	Output
	Task block	Task
	Dynamic process instance creation	Create
	Decision	Decision

( )	Decision answer / branch	Answer

Поступление определенного сигнала – это основная причина, по которой процесс начинает переход в другое состояние. Сигналы извлекаются из входной очереди с помощью действия, которое называется Input (табл. 1).

Помимо обычных состояний, диаграмма процесса должна включать в себя стартовое (начальное) состояние. Это состояние (Process start в табл. 1) указывает, с какого пункта начинается функционирование процесса сразу после его возникновения. Другими словами, любой экземпляр процесса сразу после его создания (порождения) совершает самый первый переход, который всегда начинается из стартового состояния.

Описание поведения процесса pLocal

Продолжаем рассматривать систему, описанную в лабораторной работе 1 (п.1 и п.2).

Конечный автомат, который описывает поведение процесса pLocal, имеет следующий набор состояний: 1) Idle; 2) GettingID; 3) Connecting; 4) Connected; 5) Disconnecting. Рассмотрим действия автомата, связанные с каждым из этих состояний.

Состояние Idle. Сразу после возникновения (инициализации) процесса pLocal осуществляется переход из стартового (начального) состояния в состояние Idle. В этом состоянии могут происходить следующие события:

1) Поступление сигнала sCall в оконечное устройство, которым управляет процесс pLocal, т.е. абонент, который имеет право использовать это устройство для предоставления услуг связи, делает исходящий вызов.
2) Поступление сигнала sCnxReq, что соответствует входящему вызову от другого абонента.

При поступлении любого из этих сигналов процесс pLocal должен принять входной сигнал (действие Input в табл. 1) и перейти к дальнейшим действиям по его обработке.

Рассматривая случай приема сигнала sCall, следует учитывать, что указанный сигнал приносит с собой списочный номер вызываемого абонента (будем называть его абонент B). Поэтому в результате приема рассматриваемого сигнала значение поступившего номера получит переменная calledNumber, которая относится к внутренним переменным процесса pLocal. С помощью сигнала sGetId полученное значение отправляется процессу pCentral, а процесс pLocal переходит в фазу ожидания ответного сообщения (состояние GettingID).

Если воспользоваться линейной (текстовой) версией языка SDL, то описание этих действий будет иметь следующий вид:

```
INPUT sCall(calledNumber)
OUTPUT sGetId(calledNumber)
NEXTSTATE GettingID
```

В случае приема сигнала sCnxReq процесс pLocal должен выдать ответный сигнал sCnxConf (согласие на участие в соединении со стороны абонента B) другому экземпляру процесса pLocal, от которого поступил сигнал sCnxReq, и после этого перейти в состояние Connected, что соответствует фазе сеанса связи между абонентами.

Отсюда можем получить следующий фрагмент описания действий процесса pLocal при использовании текстовой версии языка SDL:

```
INPUT sCnxReq
TASK remotePId := SENDER
OUTPUT sCnxConf TO remotePId
NEXTSTATE Connected
```

Здесь при организации отправки сигнала sCnxConf используется внутренняя системная переменная SENDER, которая в момент приема сигнала sCnxReq получает значение уникального внутреннего идентификатора (PId) отправителя поступившего сигнала. Именно отправитель сигнала sCnxReq должен теперь получить ответный сигнал sCnxConf, поэтому значение переменной SENDER сначала присваивается внутренней переменной remotePId, а затем переменная remotePId используется в операторе OUTPUT как синтаксический элемент, который в явном виде указывает, какому экземпляру процесса pLocal должен быть доставлен отправляемый сигнал (явная адресация).

В графической форме все описанные действия процесса pLocal, которые относятся к состоянию Idle, можно представить в виде следующей диаграммы:

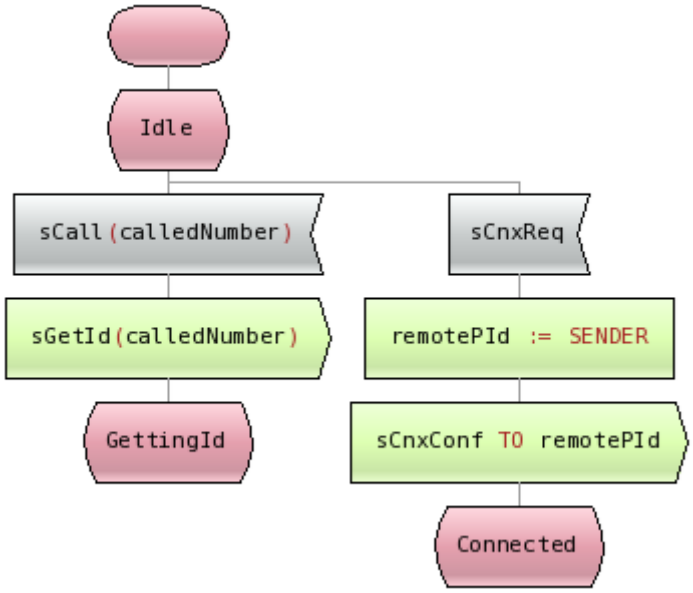


Рис. 1.1

Состояние GettingID. Действия процесса pLocal в этом состоянии определяются входными сигналами, которые поступают от процесса pCentral:

1) Сигнал sId приносит с собой значение уникального внутреннего идентификатора (PId) того экземпляра процесса pLocal, который управляет оконечным устройством абонента B. В результате приема рассматриваемого сигнала полученное значение PId присваивается внутренней переменной remotePId, и затем эта переменная используется в операторе OUTPUT для явной адресации конкретного экземпляра процесса pLocal, которому необходимо отправить уведомление о входящем вызове (сигнал sCnxReq). Выполнив это действие, процесс-отправитель переходит в состояние Connecting для ожидания дальнейших событий, связанных с установлением соединения.
2) Если ранее в состоянии Idle процессу pCentral был отправлен неверный списочный номер абонента B, то в ответ поступит сигнал sError. В этом случае абоненту A, который инициировал вызов, посылается сигнал sBusy, и процесс pLocal возвращается в исходное состояние Idle.

Отсюда получаем следующую диаграмму, которая в графической форме отображает рассмотренные действия процесса pLocal:

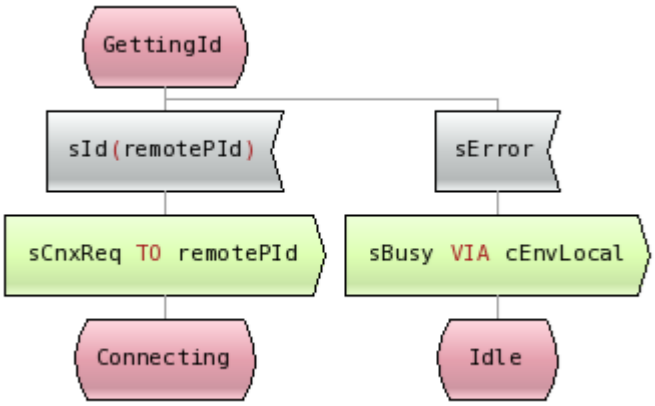


Рис. 1.2

Следует отметить, что здесь при отправке сигнала sBusy для явной адресации получателя используется название канала (cEnvLocal), который соединяет процесс pLocal с окружающей средой. Такой способ адресации необходим для того, чтобы в рассматриваемой ситуации исключить посылку сигнала sBusy другим процессам, функционирующим в системе.

Состояние Connecting. В реальных системах связи на этапе установления соединения с вызываемым абонентом существует довольно много вариантов развития событий. Чтобы чрезмерно не усложнять создаваемую модель, которая носит исключительно учебный характер, рассмотрим всего два варианта:

1) Поступление сигнала sCnxConf, что свидетельствует о свободном состоянии вызываемого абонента и его согласии принять участие в сеансе связи. После приема этого сигнала процесс pLocal с помощью сигнала sCallConf извещает абонента A, который инициировал вызов, об успешном завершении этапа установления соединения, и переходит в состояние Connected.
2) Поступление сигнала sBusy от вызываемого абонента, который находится в занятом состоянии. Реакция процесса pLocal на это событие – пересылка сигнала sBusy абоненту A, который был инициатором вызова, и возвращение в исходное состояние Idle.

Рассмотренные действия процесса pLocal можно отобразить в виде следующей диаграммы:

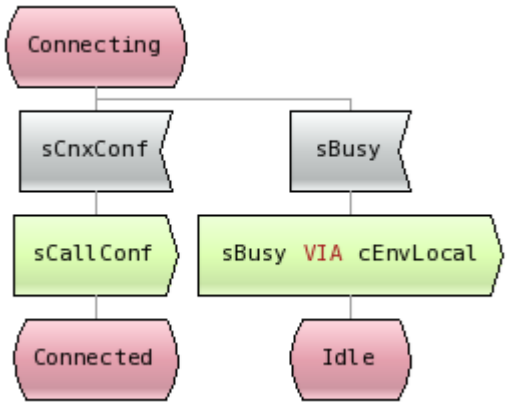


Рис. 1.3

Состояние Connected. Будем предполагать, что в этом состоянии процесса pLocal возможны следующие события, связанные с действиями участников сеанса связи (рис. 1.4):

1) Поступление сигнала sCnxReq, что означает новый входящий вызов. Поскольку состояние Connected соответствует занятому состоянию абонента, процесс pLocal посылает ответный сигнал sBusy, используя явную адресацию с помощью переменной SENDER, и не изменяет своё состояние.
2) Получение от противоположной стороны запроса на разъединение (сигнал sDisReq). В этом случае отправляется ответный сигнал sDisConf, подтверждающий разъединение, и процесс pLocal возвращается в исходное состояние Idle.
3) Абонент A, который инициировал вызов, дает отбой, что сопровождается поступлением сигнала sHangUp. После приема этого сигнала процесс pLocal посылает запрос на разъединение абоненту B (сигнал sDisReq), используя для явной адресации внутреннюю переменную remotePId, и переходит в состояние Disconnecting.

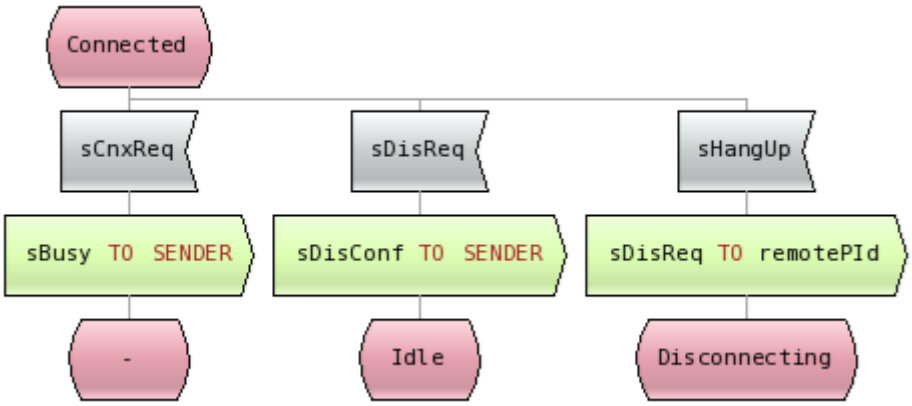


Рис. 1.4

Следует отметить, что на рис. 1.4 присутствует графический символ состояния, в котором вместо названия состояния указан дефис. По правилам языка SDL так обозначается возврат процесса в состояние, из которого начинался переход.

Состояние Disconnecting. В этом состоянии может произойти единственное событие (рис. 1.5) – поступление сигнала sDisConf, что означает подтверждение разъединения со стороны абонента B. После приема этого сигнала посылается подтверждение отбоя абоненту A (сигнал sHangUpConf) и процесс pLocal переходит в исходное состояние Idle.

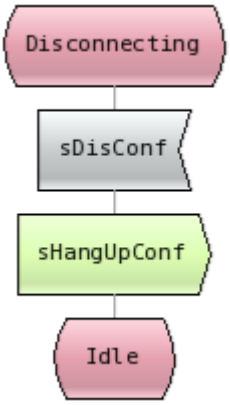


Рис. 1.5

Описание поведения процесса pCentral

Процесс pCentral является единственным программным элементом, который возникает при физическом запуске рассматриваемой телекоммуникационной системы. Следовательно, первоочередная задача этого процесса состоит в динамическом порождении остальных процессов, которые будут обеспечивать функции по обработке поступающих вызовов.

Алгоритм решения этой задачи представлен на рис. 1.6. Здесь показано, что во время перехода из стартового состояния сначала выполняются некоторые подготовительные действия:

1) формируется вспомогательный массив SN, в который записываются списочные номера абонентов, пользующихся услугами связи;
2) массив pLocals заполняется специальным значением null[1].

Далее организуется обычный цикл с параметром. Роль параметра цикла (в данном случае счетчика) играет целочисленная переменная i, значения которой последовательно изменяются от 1 до NUM_PHONE. Тело цикла включает в себя следующие основные действия:

1) Порождение экземпляра процесса pLocal (действие Create из табл. 1). При выполнении этого действия используется функциональное описание процесса pLocal, рассмотренное в предыдущем разделе. Особенность операции Create заключается в том, что порождаемый экземпляр процесса получает уникальное значение персонального идентификатора, причем это значение присваивается системной переменной offspring.
2) Запись полученного значения переменной offspring в массив pLocals. При выборе элемента массива, участвующего в записи, используется очередной элемент массив SN и этим обеспечивается соответствие между персональными идентификаторами экземпляров процесса pLocal и списочными номерами абонентов.

После выхода из цикла происходит отправка сигнала sReady во внешнюю среду, чтобы сообщить о завершении инициализации системы, а затем процесс pCentral переходит в состояние Idle (рис. 1.6).

[1] В дальнейшем значения индекса для обращения к отдельным элементам массива pLocals будут соответствовать списочным номерам абонентов, и по значению null в этом массиве легко идентифицировать несуществующие номера.

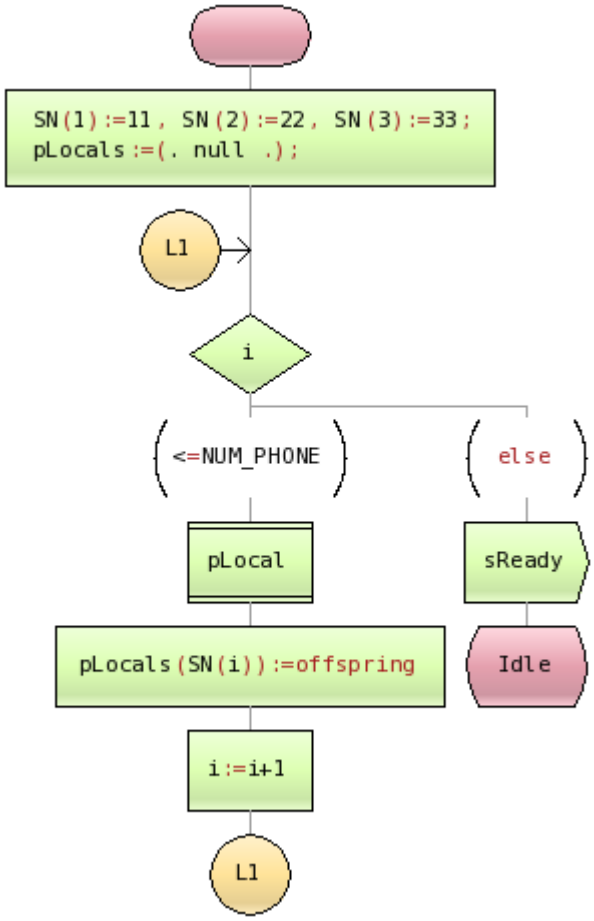


Рис. 1.6

В состоянии Idle процесс pCentral ожидает запросы, которые поступают от разных экземпляров процесса pLocal в виде сигнала sGetId. Этот сигнал содержит в себе значение списочного номера вызываемого абонента B, и задача процесса pCentral состоит в том, чтобы определить значение персонального идентификатора соответствующего экземпляра процесса pLocal, который выполняет функции управления оконечным устройством указанного абонента B.

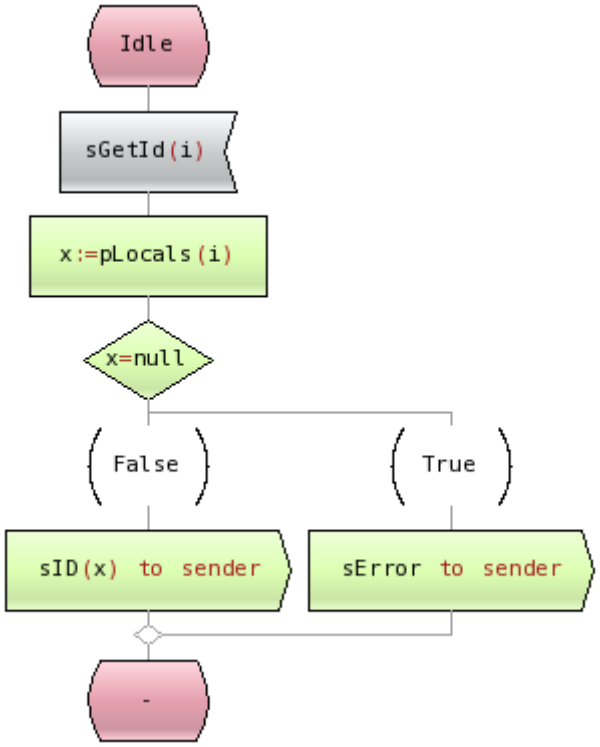


Рис. 1.7

Как следует из рис. 1.7, при выполнении действия по приему сигнала sGetId внутренняя переменная *i* получает значение параметра этого сигнала, а затем переменная *i* используется как индекс при обращении к массиву pLocals. Если значение *x*, которое будет извлечено из массива, действительно является персональным идентификатором одного из экземпляров процесса pLocal (т.е. отличается от *null*), то это значение становится параметром ответного сигнала sId. Если же значение *i* указывает на элемент массива, в котором отсутствует значение PID (т.е. в сообщении sGetId получен несуществующий списочный номер), то процесс pCentral выдает сигнал sError. Следует отметить, что явная адресация при отправке этих сигналов осуществляется с помощью системной переменной SENDER, которая в момент приема сигнала sGetId получает значение уникального внутреннего идентификатора процесса-отправителя.

Выполнив рассмотренные действия по обработке сигнала sGetId, процесс pCentral возвращается в состояние Idle, т.е. переходит в фазу ожидания очередного запроса.

2. Выполнение лабораторной работы

2.1. Построение функциональной модели процесса pLocal

1) Через окно Менеджера проекта открыть структурную модель системы, построенную при выполнении лабораторной работы №1.
2) Перейти в раздел Architecture, щелкнуть правой кнопкой мыши на графическом элементе для процесса pLocal и в контекстном меню выбрать пункт Open definition.
3) В диалоговом окне «Create def. element?» нажать кнопку OK, что приведет к открытию следующего окна – «Add child element». В этом окне нужно оставить все предварительно установленные опции (рис. 2.1) и просто нажать кнопку OK.

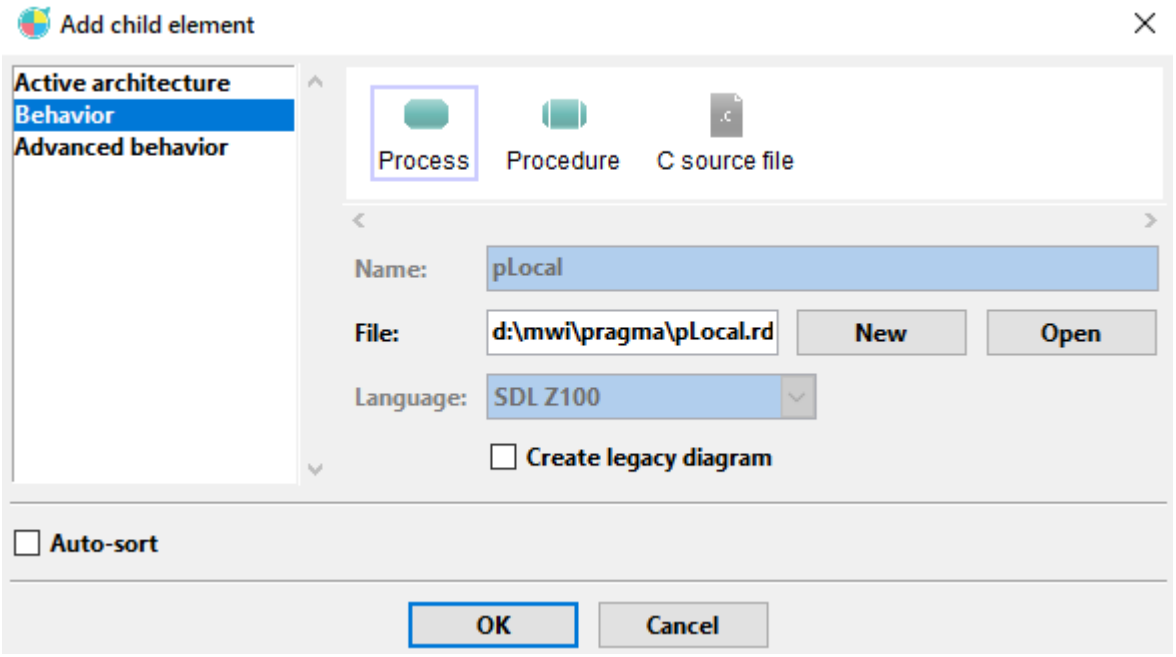


Рис. 2.1

4) Используя размещенную слева панель графических элементов, заполнить открывшееся окно «Behavioral Diagrams» диаграммами, которые формируют функциональную модель процесса pLocal (рисунки 1.1-1.5). При этом нужно учитывать, что в редакторе действует следующий порядок добавления к диаграмме нового графического элемента: сначала на создаваемой диаграмме выделяется элемент, к которому нужно присоединить очередной графический элемент, а затем на панели графических элементов редактора выбирается элемент соответствующего типа.

5) Добавить графический элемент (рис. 2.2) с объявлениями внутренних переменных, которые необходимы для функционирования процесса pLocal.

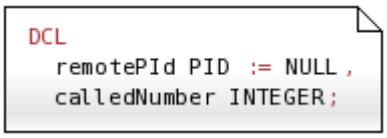


Рис. 2.2

6) Сохраните построенную модель процесса pLocal.
7) Чтобы проверить соответствие этой модели правилам языка SDL, нужно раскрыть список команд для пункта Diagram в главном меню редактора и выбрать там команду «Check syntax/semantics». При отсутствии ошибок в левом нижнем углу окна редактора появится сообщение «Everything's fine…».

2.2. Построение функциональной модели процесса pCentral

1) Через окно Менеджера проекта снова открыть структурную модель системы, построенную при выполнении лабораторной работы №1.
2) Перейти в раздел Architecture, щелкнуть правой кнопкой мыши на графическом элементе для процесса pCentral и в контекстном меню выбрать пункт Open definition.
3) Нажатием кнопки OK в диалоговом окне «Create def. element?» утвердительно ответить на вопрос о необходимости создания отсутствующего определения процесса pCentral. В результате откроется следующее окно – «Add child element». Здесь также (по аналогии с рис. 2.1) будут присутствовать все предварительно установленные опции, и остаётся только нажать кнопку OK.
4) С помощью панели графических элементов в левой части открывшегося окна «Behavioral Diagrams» заполнить это окно диаграммами, из которых состоит функциональная модель процесса pCentral (рисунки 1.6 и 1.7).
5) Добавить графический элемент (рис. 2.3) с объявлениями новых типов, которые требуются для построения функциональной модели процесса pCentral.

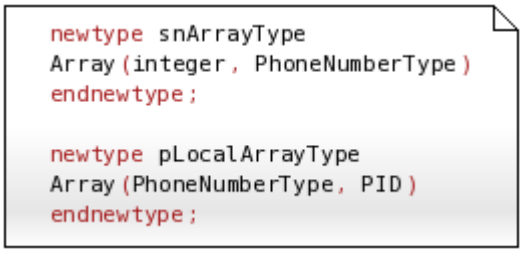


Рис. 2.3

Эти типы относятся к структуре данных, которая называется «массив» (Array). В первом случае массив состоит из элементов, которые имеют тип PhoneNumberType, и индексация этих элементов осуществляется с помощью целых чисел (тип integer). Во втором случае элементы массива имеют тип PID (т.е. это личные идентификаторы процессов), а для индексации этих элементов должны применяться значения типа PhoneNumberType.

6) Добавить графический элемент (рис. 2.4) с объявлениями внутренних переменных, которые необходимы для функционирования процесса pCentral.

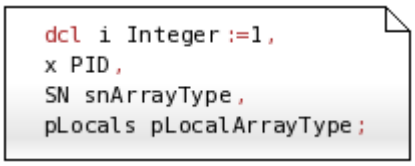


Рис. 2.4

7) Сохраните построенную модель процесса pCentral.
8) Чтобы проверить соответствие этой модели правилам языка SDL, нужно раскрыть список команд для пункта Diagram в главном меню редактора и выбрать там команду «Check syntax/semantics». При отсутствии ошибок в левом нижнем углу окна редактора появится сообщение «Everything's fine…».

3. Индивидуальное задание

Используя графические средства языка SDL, построить модель конечного автомата по известной матрице состояний и сигналов этого автомата. С помощью пакета PragmaDev Studio реализовать построенную модель как диаграмму процесса и провести ее синтаксическую проверку. Результаты выполнения задания оформить в виде отчета.

Порядок выполнения индивидуального задания

Как известно, абстрактной моделью процесса в языке SDL является конечный автомат (Finite State Machine – FSM). Поведение конечного автомата можно описать с помощью матрицы состояний и сигналов. В этой матрице столбцы соответствуют состояниям $\{W_1, W_2, \ldots, W_n\}$

рассматриваемого автомата, а строки – входным сигналам $\{S_1, S_2, \ldots, S_m\}$. Если некоторый элемент матрицы на пересечении i-го столбца ($1 \le i \le n$) и j-й строки ($1 \le j \le m$) не является пустым и содержит запись вида W_k / Z_p, то это означает, что при поступлении входного сигнала S_j автомат, находящийся в состоянии W_i, перейдет в состояние W_k и сформирует выходной сигнал Z_p. Следует также учитывать ситуацию, когда во время перехода в новое состояние автомат не формирует никаких выходных сигналов. Этой ситуации соответствует слово *none* в знаменателе дроби.

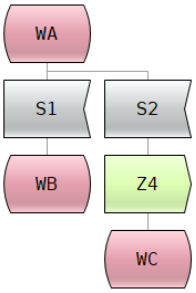
Для примера рассмотрим следующую матрицу состояний и сигналов некоторого автомата:

		Состояния автомата		
		WA	**WB**	**WC**
Входные сигналы	**S1**	WB / none		
	S2	WC / Z4		
	S3			WA / Z2, Z3
	S4		WA / none	
	S5			WB / Z1

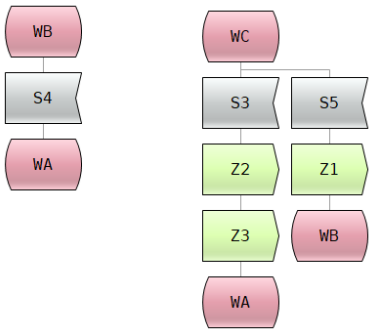
В этой матрице элемент на пересечении строки **S4** и столбца **WB** говорит о том, что если автомат находится в состоянии **WB** и поступает входной сигнал **S4**, то происходит переход в состояние **WA,** причем выходные сигналы отсутствуют. Если взять другой элемент, который располагается на пересечении строки **S3** и столбца **WC**, то он имеет следующий смысл: входной сигнал **S3**, поступающий в состоянии **WC,** вынуждает автомат перейти в состояние **WA**, а также оправить два выходных сигнала – **Z2** и **Z3.**

Чтобы осуществить переход от матрицы состояний и сигналов к модели автомата на языке SDL, требуется последовательно рассмотреть в этой матрице отдельные столбцы. Каждый из них определяет поведение автомата при условии, что автомат находится в состоянии, которое идентифицирует выбранный столбец.

В частности, если автомат находится в состоянии **WA**, то он должен реагировать на входные сигналы **S1** и **S2.** В первом случае реакция сводится к тому, что автомат переходит в состояние **WB** без выдачи каких-либо выходных сигналов, а во втором случае должен произойти переход в состояние **WC** и это сопровождается формированием выходного сигнала **Z4.** Если воспользоваться средствами языка SDL, то указанные действия можно описать с помощью следующей диаграммы:



Применяя аналогичный ход рассуждений к другим столбцам в матрице состояний и сигналов, несложно получить:



Следует обратить внимание, что описание поведения автомата в виде диаграммы, которая состоит из нескольких несвязанных частей, никак не противоречит правилам языка SDL. Хотя визуально линии потока между фрагментами диаграммы отсутствуют, на логическом уровне связи остаются, поскольку в разных фрагментах фигурируют состояния с одинаковыми названиями. Если же требуется более цельное представление, то следует попытаться совместить (т.е. наложить друг на друга) графические символы для одного и того же состояния.

Варианты исходных данных

Вариант 1

	Disconnected	**WaitForCC**	**Connected**	**WaitForAK**
ICONreq	WaitForAK / CR			
CC		Connected / ICONconf		
DR		Disconnected / IDISind	Disconnected / IDISind	Disconnected / IDISind
IDATreq			WaitForAK / DT	
AK				Connected / none

Вариант 2

	Disconnected	**Connecting**	**Connected**	**Sending**
ICONreq	Connecting / CR			
CC		Connected / ICONconf		
DR		Disconnected / IDISind	Disconnected / IDISind	Disconnected / IDISind
IDATreq			Sending / DT	
AK				Connected / none
noAK				Sending / DT

Вариант 3

	Idle	**Ringing**	**Session**	**Disconnecting**
INVITE	Ringing / sRinging, s100, s180			
sAnswer		Session / s200		
sDcon			Disconnecting /BYE	
BYE			Idle / s200	
s200				Idle / none

Вариант 4

	Idle	**Connecting**	**Session**	**Disconnecting**
sCall	Connecting / INVITE			

s200		Session / ACK		
sDcon			Disconnecting /BYE	
BYE			Idle / s200	
s200				Idle / none

Вариант 5

	Idle	**Ringing**	**Session**	**Disconnecting**
IAM	Ringing / sRinging, ACM			
sAnswer		Session / ANM		
sDcon			Disconnecting /REL	
REL			Idle / RLC	
RLC				Idle / none

Вариант 6

	Idle	**Connecting**	**Session**	**Disconnecting**
sCall	Connecting / IAM			
ANM		Session / none		
sDcon			Disconnecting / REL	
REL			Idle / RLC	
RLC				Idle / none

Вариант 7

	Unterbrochen	**Warten**	**Verbunden**	**Sendend**
ICONreq	Warten / CR			
DR	Unterbrochen / IDISind	Unterbrochen / IDISind	Unterbrochen / IDISind	Unterbrochen / IDISind
CC		Verbunden / ICONconf		
IDATreq			Sendend / DT	
AK				Verbunden / none

Вариант 8

	S1	**S2**	**S3**	**S4**
x1	S2 / z5	S1 / z1	S1 / z1	
x2	S3 / z3		S4 / z2	S3 / z3
x3		S4 / z4		S2 / z1

Вариант 9

	Disconnected	**Connecting**	**Connected**
ICONreq	Connecting / ICONind		
ICONresp		Connected / ICONconf	
IDATreq			Connected / IDATind
IDISreq		Disconnected / IDISind	Disconnected / IDISind

Вариант 10

	Disconnected	**Connecting**	**Connected**
CR	Connecting / ICONind		

ICONresp		Connected / CC	
IDISreq		Disconnected / DR	Disconnected / DR
DT			Connected / AK, IDATind

4. Требования к отчету по лабораторной работе

1) В отчете кратко описать процесс построения функциональной модели телекоммуникационной системы согласно рекомендациям из раздела 2 методических указаний.

2) Привести решение индивидуального задания из раздела 3 по аналогии с примером выполнения индивидуального задания.

3) К отчету приложить архивные файлы с проектами, которые были получены при работе с пакетом PragmaDev Studio.