ГОСТ Р ИСО/МЭК ТО 10023 –93

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ

ПЕРЕДАЧА ДАННЫХ И ОБМЕН ИНФОРМАЦИЕЙ МЕЖДУ СИСТЕМАМИ. ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ УСЛУГ ТРАНСПОРТНОГО УРОВНЯ (ГОСТ 34.960—91) НА ЯЗЫКЕ LOTOS

Издание официальное

БЗ 12—92/1177

ГОССТАНДАРТ РОССИИ
Москва

gost.e-ecolog.ru

Предисловие

1 ПОДГОТОВЛЕН И ВНЕСЕН Техническим комитетом по стандартизации ТК 22 «Информационная технология»

2 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Постановлением Госстандарта России от 29.12.93 № 293

Настоящий стандарт подготовлен на основе применения аутентичного текста международного стандарта ИСО МЭК ТО 10023—92 «Информационная технология. Передача данных и обмен информацией между системами. Формализованное описание ИСО 8072 на LOTOS»

3 ВВЕДЕН ВПЕРВЫЕ

© Издательство стандартов, 1994

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Госстандарта России

СОДЕРЖАНИЕ

1	Область применения	1
2	Нормативные ссылки	1
3	Определения	2
4	Символы и сокращения	2
5	Соглашения	2
6	Требования	3
7	Введение в формализованное описание	3
8	Типы данных на интерфейсе	4
9	Глобальные ограничения	30
10	Обеспечение транспортного соединения	32
11	Локальные ограничения для оконечного пункта ТС	33
12	Межоконечные ограничения для одного ТС	37
13	Идентификация транспортных соединений	43
14	Принятие транспортных соединений	45
15	Управление потоком при помощи обратной связи	46
16	Передача в режиме-без-установления-соединения	47
	Библиографические данные	49

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная технология

ПЕРЕДАЧА ДАННЫХ И ОБМЕН ИНФОРМАЦИЕЙ МЕЖДУ СИСТЕМАМИ. ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ УСЛУГ ТРАНСПОРТНОГО УРОВНЯ (ГОСТ 34.960—91) НА ЯЗЫКЕ LOTOS

Information Technology Telecommunications and Information Exchange Between Systems. Formal Description of 8072 in LOTOS

Дата введения 1994—07—01

1 ОБЛАСТЬ ПРИМЕНЕНИЯ

Настоящий стандарт распространяется на транспортный уровень эталонной модели взаимосвязи открытых систем (ВОС) и определяет услуги транспортного уровня ВОС, определенные в ГОСТ 34.960, при помощи метода формализованного описания LOTOS, определенного в ИСО 8807.

Примечание — Формальное определение типов данных и процессы, представленные в настоящем стандарте, могут использоваться для формализованного описания протоколов транспортного и сеансового уровней ВОС на языке LOTOS

2 НОРМАТИВНЫЕ ССЫЛКИ

Нижеперечисленные стандарты содержат положения, которые путем ссылок на них по тексту образуют положения настоящего стандарта. Все ссылки предполагают последнее издание указанных стандартов.

Национальные комитеты — члены МЭК и ИСО имеют списки международных стандартов, действующих на текущий момент.

ГОСТ 28906—91 (ИСО 7498—84, ИСО 7498—84 Доп. 1—84) Системы обработки информации. Взаимосвязь открытых систем. Базовая эталонная модель

ГОСТ 34.960—91 (ИСО 8072—86, Доп. 1—86 ИСО 8072—86) Системы обработки информации. Взаимосвязь открытых систем. Определение услуг транспортного уровня

Издание официальное

ИСО 8072—86/Поп. 1 Системы обработки информации. Взаимосвязь открытых систем. Определение услуг транспортного уровня. Техническая поправка 1*.

ИСО/ТО 8509—87 Системы обработки информации. Взаимосвязь открытых систем. Соглашения по услугам *.

ИСО 8807—89 Системы обработки информации. Взаимосвязь открытых систем. LOTOS — метод формализованного описания, основанный на упорядочении во времени наблюдаемого поведения *.

ИСО/МЭК ТО 10024—92 Информационная технология. Передача данных и обмен информацией между системами. Формализованное описание *ИСО 8073* (разделы 0, 1, 2 и 3) на языке LOTOS *.

3 ОПРЕДЕЛЕНИЯ

В настоящем стандарте используются определения, приведенные в ГОСТ 34.960.

4 СИМВОЛЫ И СОКРАЩЕНИЯ

В настоящем стандарте используются символы, определенные в разделе 6 (формальный синтаксис) и приложении А (библиотека типов данных) ИСО 8807.

В настоящем стандарте используются сокращения, содержащиеся в разделе 4 ГОСТ 34.960. Использование других символов и сокращений поясняется при первом их появлении.

5 СОГЛАШЕНИЯ

Неформальные пояснения, предшествующие формальным определениям, к которым они относятся, отделяются от последних показанной ниже верхней линией. Отделение формальных определений от последующих неформальных пояснений обозначается показанной нижней линией.

---------- *)

(* ----------

* До прямого применения данного документа в качестве государственного стандарта его распространение осуществляет секретариат ТК 22 «Информационная технология».

Примечание — Это соглашение соответствует правилам ограничения комментариев, определенным для LOTOS в ИСО 8807. Формальный текст представлен курсивом, а ключевые слова и операторы LOTOS — полужирным шрифтом. Идентификаторы из форматированного текста в неформатированном тексте набраны курсивом.

Соблюдаются соглашения, определенные в ИСО/ТО 8509, но с учетом следующего: термин «запрос» означает как запросные, так и ответные сервисные примитивы, а термин «индикация» означает сервисные примитивы как индикации, так и подтверждения.

6 ТРЕБОВАНИЯ

Настоящий стандарт отвечает требованиям, изложенным в разделе 3 ИСО 8807 (более подробную информацию см. в приложении С к указанному стандарту). Настоящий стандарт не содержит каких-либо требований к соответствию.

7 ВВЕДЕНИЕ В ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ

Вся граница услуг формально представлена в виде единственного шлюза t. Структура события в t представляет собой тройку значений типа TAddress, TCEI, TSP (см. раздел 8). Первое значение идентифицирует TSAP, в котором происходит взаимодействие. Второе значение идентифицирует TCEP внутри TSAP, в котором происходит взаимодействие. Третье значение — это выполняемый при взаимодействии примитив транспортного уровня (TSP). Конкретные значения зависят от многих аспектов услуги.

Описывается поведение поставщика услуг бесконечного вида. Спецификация не содержит параметров.

Используется стиль спецификации, ориентированный на ограничения, поскольку он наиболее подходит для определительного характера стандарта по услугам. Описание ориентировано на характеристики модальностей поведения поставщика услуг исключительно в терминах последовательности выполняемых TSP, т. е. без каких-либо предположений о внутренней структуре самого поставщика.

В основе первой декомпозиции специфицируемого поведения лежат следующие отдельные ограничения:

a) поставщик услуг может допускать множество, возможно, одновременных соединений (представленных процессом TConnections, см. раздел 9), вместе со

b) средством выбора среди одновременных соединений нужного (представленным процессом TCIdentification, см. раздел 13), но при

c) возможности внутреннего недетерминизма при установлении любого дополнительного соединения, если, по меньшей мере, одно соединение уже поддерживается (представленной процессом TCAcceptance, см. раздел 14), и

d) возможности внутреннего недетерминизма при передаче данных (представленной процессом TBackpressure, см. раздел 15); при помощи этого принимающий пользователь вызывает эффект управления потоком, сообщая об этом передающему пользователю.

Спецификация динамического поведения предъваряется спецификацией типов данных на интерфейсе (см. раздел 8). Такие определения являются общими для формализованных описаний, которые могут взаимодействовать, а именно для протоколов транспортного и сеансового уровней.

Порядок представления остальных определений обоснован желанием следовать порядку, установленному ГОСТ 34.960, который в основном связан с обеспечением единственного ТС. Описание требований, являющихся локальными для обеспечения одного ТС (представлены в разделах 10, 11 и 12), предшествуют описаниям глобальных требований, упомянутых выше в b), c) и d) соответственно.

Типы данных, определенные конструкцией library, импортируются из библиотеки типов данных LOTOS.

---*)

```
specification TransportService[t] : noexit
library Set, Element, OctetString, NatRepresentation, NaturalNumber, Boolean, FBoolean, DecNatRepr
endlib
(*
```

8 ТИПЫ ДАННЫХ НА ИНТЕРФЕЙСЕ

8.1 Общее описание

В соответствии с представлением взаимодействий на границе транспортных услуг (см. раздел 7) типы данных на интерфейсе состоят из трех основных определений, которые соответственно составляют виды TAddress (см. 8.2), TCEI (см. 8.3) и TSP (см. 8.4). Параметр TSP — качество услуги — определяется в 8.5, а остальные параметры — в 8.6. В 8.7 представлены вспомогательные определения.

8.2 Транспортный адрес

В ГОСТ 34.960 не определена структура транспортного адреса. Следующее определение использует определение GeneralIdentifier

(см. 8.7) и позволяет представлять бесконечное число транспортных адресов.

```
----------------------------------------------------------------------------*)
type TransportAddress
is GeneralIdentifier renamedby
sortnames
        TAddress for identifier
opnnames
        SomeTAddress  for SomeIdentifier
        AnotherTAddress for AnotherIdentifier
endtype (* TransportAddress *)
(* ----------------------------------------------------------------------------
```

8.3 Идентификатор оконечного пункта транспортного соединения

В ГОСТ 34.960 не определена структура идентификатора оконечного пункта транспортного соединения. Приводимое ниже первое определение позволяет представить бесконечное их число. Второе определение представляет идентификаторы оконечного пункта транспортного соединения, которые являются глобальными для всей границы транспортных услуг. Каждый из них представляет собой пару TAddress×TCEI (общее определение Pair и GeneralIdentifier см. в 8.7).

```
----------------------------------------------------------------------------*)
type TCEndpointIdentifier
is GeneralIdentifier renamedby
sortnames
        TCEI for identifier
opnnames
        SomeTCEI for SomeIdentifier
        AnotherTCEI for AnotherIdentifier
endtype (* TCEndpointIdentifier *)
type TCEIdentification
is Pair actualizedby TransportAddress, TCEndpointIdentifier using
sortnames
        TAddress for Element
        TCEI for Element2
        Bool for Fbool
        TId for Pair
opnnames
        TId for Pair
```

```
        TA for First
        TCEI for Second
endtype (* TCEIdentification *)
(*-----------------------------------------------------------------------
```

8.4 Сервисный примитив транспортного уровня (TSP)

8.4.1 *Общее описание*

Тип данных TCP представлен, начиная с базовой конструкции значений вида TCP (см. ниже). Эта конструкция является прямой формулировкой определения, приведенного в таблице 3 ГОСТ 34.960. Функции, генерирующие значения TCP, называются «конструкторами TCP». Это определение заимствует определения, связанные с параметрами TCP (см. 8.6).

Примечание — В некоторых TCP параметр UserData представляет собой OctetString, имеющий фиксированные границы, как определено в ГОСТ 34.960. По техническим соображениям это требование формально представлено процессом ограничения (см. 11.1 и 11.4), а не ограничивающим типом.

В 8.4.2 приведена классификация TCP, которая позволяет, с одной стороны, простым путем расширить базовую конструкцию дополнительными функциями (см. 8.4.3), а с другой — консервативно расширить тип данных в формализованном описании транспортного протокола.

```
-----------------------------------------------------------------------*)
type BasicTSP
is TransportAddress, TEXOption, TSQuality, OctetString, TDISReason, TsClQuality
sorts
        TSP
opns
        TCONreq, TCONind        : TAddress, TAddress, TEXOption, TQOS, OctetString
                                  —> TSP
          TCONresp, TCONconf    TAddress, TEXOption, TQOS, OctetString —> TSP
        TDTreq, TDTind          : OctetString    —> TSP
        TEXreq, TEXind          : OctetString    —> TSP
        TDISreq                 : OctetString    —> TSP
        TDISind                 : TDISReason, OctetString
                                  —> TSP
        TUDTreq, TUDTind        : TAddress, TAddress, ClQOS,
```

```
                                        OctetString —> TSP
endtype (* BasicTSP *)
(* ----------------------------------------------------------------------
```

8.4.2 *Классификация сервисных примитивов транспортного уровня*

8.4.2.1 *Базовая классификация*

Базовая классификация TCP определена при помощи TCPSubsort, состоящего из набора констант, каждая из которых задает имя TCP в соответствии с таблицей 3 ГОСТ 34.960.

Тип TCPBasicClassifiers — это функциональное расширение базовой конструкции в 8.4.1, где:

a) функция Subsort генерирует имя TCP;

b) булевы функции на TCP, названные «определителями подвида TCP», определяются в соответствии с базовой классификацией, введенной при помощи TCPSubsort.

Примечание — Вспомогательная функция h, отображающая имена TCP на натуральные числа, определяется для упрощения спецификации булевых операций равенств на именах TCP (так же, как на TCP в 8.4.3.3). Определение IsRequest, IsIndication (на именах TCP) и IsTreq, IsTind (на TCP) отражает соглашение, введенное в раздел 5

```
---------------------------------------------------------------------- *)
type TSPSubsort
is NaturalNumber
sorts
        TSPSubsort
opns
        TCONNECTrequest, TCONNECTindication, TCONNECTresponse, TCONNECTconfirm, TDATArequest, TDATAindication, TEXDATArequest, TEXDATAindication, TDISCONNrequest, TDISCONNindication, TUDATArequest, TUDATAindication
                                        : —> TSPSubsort
        h                       : TSPSubsort        —> Nat
        Even, Odd               : Nat               —> Bool
        IsRequest, IsIndication : TSPSubsort        —> Bool
        _eq_,_ne_               : TSPSubsort, TSPSubsort
                                                    —> Bool
eqns
forall
        s, s1  : TSPSubsort, n : Nat
 ofsort Nat
        h(TCONNECTrequest) = 0;
```

```
        h(TCONNECTindication) = Succ(h(TCONNECTrequest));
        h(TCONNECTresponse) = Succ(h(TCONNECTindication));
        h(TCONNECTconfirm) = Succ(h(TCONNECTresponse));
        h(TDATArequest) = Succ(h(TCONNECTconfirm));
        h(TDATAindication) = Succ(h(TDATArequest));
        h(TEXDATArequest) = Succ(h(TDATAindication));
        h(TEXDATAindication) = Succ(h(TEXDATArequest));
        h(TDISCONNrequest) = Succ(h(TEXDATAindication));
        h(TDISCONNindication) = Succ(h(TDISCONNrequest));
ofsort Bool
        Even(0) = true;
        Even(Succ(0)) = false,
        Even(Succ(Succ(n))) = Even(n);
        Odd(n) = not(Even(n));
        IsRequest(s) = Even(h(s));
        IsIndication(s) = Odd(h(s));
        s eq sl = h(s) eq h(sl);
        s ne sl = not(s eq sl);
endtype (* TSPSubsort *)
type TSPBasicClassifiers
is BasicTSP, TSPSubsort
opns
        Subsort                : TSP          —> TSPSubsort
        IsTCON, IsTCON1, IsTCON2, IsTDT, IsTEX, IsTDIS,
        IsTCONreq, IsTCONind, IsTCONresp, IsTCONconf,
        IsTDTreq, IsTDISind, IsTReq, IsTInd : TSP —> Bool
eqns
forall
        a, a1, a2 : TAddress, x : TEXOption, q : TQOS, d :
        OctetString, r : TDISReason, t : TSP, clq : CLQOS
ofsort TSPSubsort
        Subsort(TCONreq(a1, a2, x, q, d)) = TCONNECTrequest;
        Subsort(TCONind(a1, a2, x, q, d)) = TCONNECT-
        indication;
        Subsort(TCONresp(a, x, q, d)) = TCONNECTresponse;
        Subsort(TCONconf(a, x, q, d)) = TCONNECTconfirm;
        Subsort(TDTreq(d)) = TDATArequest;
        Subsort(TDTind(d)) = TDATAindication;
        Subsort(TEXreq(d)) = TEXDATArequest;
        Subsort(TEXind(d)) = TEXDATAindication;
        Subsort(TDISreq(d)) = TDISCONNrequest;
        Subsort(TDISind(r, d)) = TDISCONNindication;
```

```
        Subsort(TUDTreq(a1, a2, clq, d)) = TUDATArequest;
        Subsort(TUDTind(a1, a2, clq, d)) = TUDATAindication;
ofsort Bool
        IsTCON(t) = IsTCON1(t) or IsTCON2(t);
        IsTCON1(t) = IsTCONreq(t) or IsTCONind(t);
        IsTCON2(t) = IsTCONresp(t) or IsTCONconf(t);
        IsTDT(t) = IsTDTreq(t) or IsTDTind(t);
        IsTEX(t) = IsTEXreq(t) or IsTEXind(t);
        IsTDIS(t) = IsTDISreq(t) or IsTDISind(t);
        IsTCONreq(t) = Subsort(t) eq TCONNECTrequest;
        IsTCONind(t) = Subsort(t) eq TCONNECTindication;
        IsTCONresp(t) = Subsort(t) eq TCONNECTresponse;
        IsTCONconf(t) = Subsort(t) eq TCONNECTconfirm;
        IsTDTreq(t) = Subsort(t) eq TDATArequest;
        IsTDTind(t) = Subsort(t) eq TDATAindication;
        IsTEXreq(t) = Subsort(t) eq TEXDATArequest;
        IsTEXind(t) = Subsort(t) eq TEXDATAindication;
        IsTDISreq(t) = Subsort(t) eq TDISCONNrequest;
        IsTDISind(t) = Subsort(t) eq TDISCONNindication;
        IsTUDT(t) = IsTUDTreq(t) or IsTUDTind(t);
        IsTUDTreq(t) = Subsort(t) eq TUDATArequest;
        IsTUDTind(t) = Subsort(t) eq TUDATAindication;
        IsTReq(t) = IsRequest(Subsort(t));
        IsTInd(t) = IsIndication(Subsort(t));
endtype (*TSPBasicClassifiers *)
(*-------------------------------------------------------------------------------
```

8.4.2.2 *Вспомогательная классификация*

TDATAAtomSubsort вводит дальнейшую классификацию элементарных составляющих примитивов данных, а именно октеты данных пользователя и ограничители СБДТ. Однако в данной спецификации эти составляющие не специфицированы.

Примечание — Единственная причина представления TDATAAtomSubsort в этом описании состоит в том, что таким способом тип данных TransportServicePrimitive допускает консервативное расширение, при котором можно ввести более изящное неэлементарное представление примитивов данных. Такое расширение необходимо в формализованном описании транспортного протокола для правильной формулировки требований, связанных с сегментацией и управлением потоком.

Булева функция Terminates, определенная в TSPClassifiers, связывает элементарное выполнение примитивов данных, что характерно для определения услуг, с неэлементарным их выполнением, присутствующим в формализованном описании протокола.

```
----------*)

type TDATAAtomSubsort
is FourTuplet renamedby
sortnames
        TDATASubsort for Tuplet
opnnames
        TDATAOCTrequest for TheOne
        TDATAOCTindication for TheOther
        TEOTrequest for TheThird
        TEOTindication for TheFourth
endtype (* TDATAAtomSubsort *)
type TSPClassifiers
is TSPBasicClassifiers, TDATAAtomSubsort
opns
        Terminates   : TDTASubsort, TSP          —> Bool
eqns
forall
        d : OctetString,  s : TDTASubsort,  t : TSP
ofsort Bool
        TEOTrequest Terminates TDTreq(d) =true;
        TEOTindication Terminates TDTind(d) =true;
        TEOTrequest Terminates TDTind(d) =false;
        TEOTindication Terminates TDTreq(d) =false;
        TDATAOCTrequest Terminates t=false;
        TDATAOCTindication Terminates t=false;
        not(IsTDT(t))  => Terminates t=false;
endtype (* TSPClassifiers *)
(* ----------
```

8.4.3 *Функции сервисных примитивов транспортного уровня*

8.4.3.1 *Общее описание*

В 8.4.3.2 конструкция, представленная в 8.4.2, расширяется функциями, допускающими определение значений конкретных параметров TCP. В 8.4.3.3 в эту конструкцию добавляются булевы равенства. В 8.4.3.4 представлены дальнейшие функциональные расширения, которые полезны для представления согласования (см. 11.3.2) и недетерминизма поставщика услуг (см. 12.3.3).

8.4.3.2 *Селекторы параметров сервисных примитивов транспортного уровня*

Для сравнения или селекции значений конкретных параметров TSP определены булевы функции. Причина для такого непрямого представления — неполнота определения при помощи равенств.

Единственное исключение имеется в прямом представлении параметра данных пользователя при помощи функции UserData, так как этот параметр можно определить во всех TSP.

```
---------------------------------------------------------------------------------------------------- *)
type TSPParameterSelectors
is TSPClassifiers
opns
        _IsCalledOf_, _IsCallingOf_, _IsRespondingOf_ : TAddress,
        TSP —>Bool
        _IsTEXOptionOf_: TEXOption, TSP              —> Bool
        _IsTQOSOf_     : TQOS, TSP          —> Bool
        _IsReasonOf_   : TDISReason, TSP    —> Bool
        UserData       : TSP                —> OctetString
eqns
forall
        a, a1, a2 : TAddress, x, x1, : TEXOption, q, q1 : TQOS,
        d : OctetString, r, r1 : TDISReason, t : TSP, clq, clq1
        : ClQOS
ofsort Bool
        a IsCalledOf TCONreq(a1, a2, x, q, d) = a eq a1;
        a IsCalledOf TCONind(a1, a2, x, q, d) = a eq a1;
        a IsCalledOf TUDTreq(a1, a2, clq, d) = a eq a1;
        a IsCalledOf TUDTind(a1, a2, clq, d) = a eq a1;
        not(IsTCON1(t)) => a IsCalledOf t=false;
        a IsCallingOf TCONreq(a1, a2, x, q, d) = a eq a1;
        a IsCallingOf TCONind(a1, a2, x, q, d) = a eq a1;
        a IsCallingOf TUDTreq(a1, a2, clq, d) = a eq a1;
        a IsCallingOf TUDTind(a1, a2, clq, d) = a eq a1;
        not(IsTCON1(t)) => a IsCallingOf t=false;
        a IsRespondingOf TCONresp(a1, x, q, d) = a eq a1;
        a IsRespondingOf TCONconf(a1, x, q, d) = a eq a1;
        not(IsTCON2(t))) => a IsRespondingOf t=false;
        x IsTEXOptionOf TCONreq(a1, a2, x1, q, d) = x eq x1
        x IsTEXOptionOf TCONind(a1,  a2, x1, q, d) = x eq x1
        x IsTEXOptionOf TCONresp(a, x1, q, d) = x eq x1
        x IsTEXOptionOf TCONconf(a, x1, q, d) = x eq x1
        not(IsTCON(t)) => x IsTEXOptionOf t=false;
        q IsTQOSOf TCONreq(a1, a2, x, q1, d) = q eq q1
        q IsTQOSOf TCONind(a1, a2, x, q1, d) = q eq q1
        q IsTQOSOf TCONresp(a, x, q1, d) = q eq q1
        q IsTQOSOf TCONconf(a, x, q1, d) = q eq q1
        not(IsTCON(t)) => q IsTQOSOf t=false;
```

```
        r IsReasonOf TDISind(r1, d) = r eq r1;
        not(IsTDISind(t))  => r IsReasonOf t = false;
        clq IsClQOSOf TUDTreq(a1, a2, clq1, d) = clq eq clq1
        clq IsClQOSOf TUDTind(a1, a2, clq1, d) = clq eq clq1
ofsort OctetString
        UserData(TCONreq(a1, a2, x, q, d)) = d;
        UserData(TCONind(a1, a2, x, q, d)) = d;
        UserData(TCONresp(a, x, q, d)) = d;
        UserData(TCONconf(a, x, q, d)) = d;
        UserData(TDTreq(d)) = d;
        UserData(TDTind(d)) = d
        UserData(TEXreq(d)) = d;
        UserData(TEXind(d)) = d;
        UserData(TDISreq(d)) = d
        UserData(TDISind(r, d)) = d;
        UserData(TUDTreq(a1, a2, clq, d)) = d;
        UserData(TUDTind(a1, a2, clq, d)) = d;
endtype (* TSPParameterSelectors *)
(*- - - -
```

```
        (a eq a1) and (a2 eq a3) and (x eq x1) and (q eq q1) and
        (d eq d1);
        TCONresp(a, x, q, d) eq TCONresp(a1, x1, q1, d1) = (a eq a1)
        and (x eq x1) and (q eq q1) and (d eq d1);
        TCONconf(a, x, q, d) eq TCONconf(a1, x1, q1, d1) = (a eq a1)
        and (x eq x1) and (q eq q1) and (d eq d1);
        TDISind(r, d) eq TDISind(r1, d1) = (r eq r1) and (d eq d1);
        TUDTreq(a, a2, clq, d) eq TUDTreq(a1, a3, clq1, d1) =
        (a eq a1) and (a2 eq a3) and (clq eq clq1) and (d eq d1);
        TUDTind(a, a2, clq, d) eq TUDTind(a1, a3, clq1, d1) =
        (a eq a1) and (a2 eq a3) and (clq eq clq1) and (d eq d1);
        not(IsTCON(t) or IsTDIS(t))
        => t eq t1
        = (Subsort(t) eq Subsort(t1)) and (UserData(t) eq
        UserData(t1)) and (IsTDT(t) implies (t eqTerm t1);
        t ne t1 = not(t eq t1);
        t eqTerm t1
        = TEOTrequest Terminates t iff (TEOTrequest Tepminates
        t1) and (TEOTindication terminates t iff (TEOTindication
        Terminates t1));
endtype (* TSPEquality *)
(*-----------------------------------------------------------------------
```

8.4.3.4 *Прочие функции сервисных примитивов транспортного уровня*

Функция ProviderGeneratedInd характеризует TSP, генерируемые исключительно поставщиком услуг. Эта функция используется для описания возможного недетерминизма поставщика услуг (см. 12.3.3).

Функция IsIndicationOf связывает выполнение TSP в каждом оконечном пункте транспортного соединения с предварительным выполнением соответствующего примитива в другом пункте того же самого транспортного соединения (см. 12.3.3). Эта функция также представляет требования по согласованию в отношении возможного недетерминизма поставщика услуг (см. раздел 10 и 14.2 ИСО 8072).

```
-----------------------------------------------------------------------*)
type TransportServicePrimitive
is TSPEquality
opns
        ProviderGeneratedInd : TSP                    —> Bool
        IsIndicationOf_,_IsValidTCON2For_ : TSP, TSP —> Bool
```

```
eqns
forall
        t, t1 : TSP, a, a1, a2, a3 : TAddress, x, x1 : TEXOption,
        q, q1 : TQOS, d, d1 : OctetString, clq, clq1 : CIQOS
ofsort Bool
        PrividerGeneratedInd(t)
        = IsTDISind(i) and (Provider IsReasonOf(t) and
        (UserData(t) eq <>);
        TCONconf(a1, x1, q1, d1) IsValidTCON2For TCONreq(a,
        a2, x, q, d) = (a1 eq a) and (q1 eq q) and ((x1 eq UseTex)
        implies (x eq UseTEX));
        TCONresp(a1, x1, q1, d1) IsValidTCON2For TCONind(a,
        a2, x, q, d) = (a1 eq a) and (q1 eq q) and ((x1 eq UseTex)
        implies (x eq UseTEX));
        not((IsTCONconf(t1) and IsTCONreq(t)) or ((IsTCON-
        resp(t1) and IsTCONind(t))) => t1 IsValidTCON2For t=
        false;
        TCONind(a1, a3, x1, q1, d1) IsIndicationOf TCONreq(a, a2,
        x, q, d) = (a1 eq a) and (a3 eq a2) and (x1 eq x) and (q1 le
        q) and (d1 eq d);
        TCONconf(a1, x1, q1, d1) IsIndicationOfTCONresp(a, x, q,
        d) = (a1 eq a) and (x1 eq x) and (q1 eq q ) and (d1 eq d);
        IsTCON(t), not(IsTReq(t)) => t1 IsIndicationOf t=false;
        IsTCON(t), IsTReq(t), h(Subsort(t1)) ne Succ(h(Sub-
        sort(t))) => t1 IsIndicationOf t=false;
        TUDTind(a1, a3, clq1, d1) IsIndicationOf TUDTreq(a, a2,
        clq, d) = (a1 eq a) and (a3 eq a2) and (clq1 le clq) and
        (d1 eq d);
        not(IsTCON(t) or IsTUDT(t)) => t1 IsIndicationOf t=
        IsTReq(t) and IsTInd(t1) and (h(Subsort(t1)) eq
        Succ(h(Subsort(t)))) and ((TEOTindication Terminates t1)
        iff (TEOTrequest Terminates t)) and (UserData(t1) eq
        UserData(t));
endtype (* TransportServicePrimitive *)
(*-----------------------------------------------------------------------------
```

8.5 Качество услуг (КУ)

8.5.1 *Общее описание*

Структура параметра КУ подразделяется в соответствии с определением, приведенным в разделе 10 ГОСТ 34.960.

Первая декомпозиция выделяет параметры КУ: производительность ТС, приоритет ТС и защита ТС. Эти структуры определяют

ся в 8.5.2—8.5.4 соответственно и ссылаются на вспомогательные определения КУ, приведенные в 8.5.5.

Как конструкция TQOS, так и конструкция TCPerformance, а также их подструктуры в основном состоят из декартова произведения, образованного функциями проекции, каждая из которых дает один множитель значения произведения и булевых функций, специфицирующих равенство и частичное упорядочение КУ, как определено в разделе 10 ГОСТ 34.960.

В 8.5.2 в виде пояснения представлены видовые произведения, относящиеся к производительности ТС.

Примечания

1 Порядок, в котором представлены определения КУ, отличается от порядка, представленного в разделе 10 ГОСТ 34.960, что обеспечивает группирование (в формальном контексте) сходных определений, что облегчает чтение.

2 Представление значений TQOS формально полное только для той области, которая оказывается необходимой, чтобы обеспечить абстрактную спецификацию таких значений, как параметры TSP, и соответствующих требований по согласованию КУ (см. четвертый абзац раздела 10 ГОСТ 34.960) Дополнительные функции, позволяющие оценивать КУ при тестировании, и примеры измерений не определены. Оценка каким-либо образом КУ не связана с динамическими требованиями, представленными настоящим формализованным описанием, так как семантика LOTOS абстрагируется от количественных аспектов, таких как время и вероятности.

3 В настоящем описании не указано, является ли значение КУ абсолютным требованием пользователя или приемлемо также пониженное значение.

Структура параметров КУ режима-без-установления-соединения представлена в виде декартова произведения параметров TCTransitDelay, TCProtection, NCProbability и TCPriority.

---*)

```
type TSQuality
is POThreeTuple actualizedby  TCPerformance, TCPriority,
TCProtection using
sortnames
        TQOS for ThreeTuple
        TCPerformance for Element
        TCPriority for Element2
        TCProtection for Element3
         Bool for FBool
opnnames
        TQOSPerformance for First
        TQOSPriority for Second
        TQOSProtection for Third
        TQOS for Tuple
endtype (* TSQuality *)
```

```
type TSClQuality
is POFourTuple actualizedby CLTransitDelay, TCProtection,
Probability, TCPriority using
sortnames
        Bool for FBool
        CLQOS for FourTuple
        CLTransDelay for Element
        TCProtection for Element2
        Prob for Element3
        TCPriority for Element4
opnnames
        CLQOSTransDelay for First
        CLQOSProtection for Second
        CLQOSProbability for Third
        CLQOSPriority for Fourth
        CLQOS for Tuple
endtype (* TSClQuality *)
(* ----------------------------------------------------------------------
```

8.5.2 *Параметры производительности*

8.5.2.1 *Общее описание*

Параметры производительности, составляющие компонент параметров КУ, имеют следующую структуру четверки:

TCPerformance = Delays × Failures × Throughput × RER,

что представлено в данном ниже определении. Определения компонентов даны в 8.5.2.2—8.5.2.5.

Примечание — Вспомогательные определения КУ (см 8.5.5) полезны для понимания определений параметра производительности, содержащегося в 8.5.2.2—8.5.2.5. При первом чтении настоящего стандарта рекомендуется сначала ознакомиться с 8.5.5.

```
---------------------------------------------------------------------- *)
type TCPerformance
is BasicTCPerformance
opns
        _lt_, _le_, _gt_,
_ge_ : TCPerformance, TCPerformance —> Bool
eqns
forall
        d, d1 : Delays, f, f1 : failures, t, t1 : Throughput, r,
        r1 : TPRER, pf, pf1 : TCPerformance
ofsort Bool
        Performance(d, f, t, r) le Performance(d1, f1, t1, r1) =
        (d ge d1) and (f ge f1) and (t le t1) and (r ge r1);
```

```
        pf lt pf1 = (pf le pf1) and not(pf1 le pf);
        pf eq pf1 = (pf le pf1) and (pf1 le pf);
        pf ne pf1 = not(pf eq pf1);
        pf ge pf1 = pf1 le pf;
        pf gt pf1 = (pf ge pf1) and not(pf1 ge pf);
endtype (* TCPerformance *)
type BasicTCPerformance
is FourTuple actualizedby TCDelays, TCThroughput
  TCResidualErrorRate, TCFailureProbabilities using
sortnames
        TCPerformance for FourTuple
        Delays for Element
        Failures for Element2
        Throughput for Element3
        TPRER for Element4
        Bool for FBool
opnnames
        Performence for Tuple
        Delays for First
        Throughput for Second
        RER for Third
        Failures for Fourth
endtype (* BasicTCPerformance *)
(*-----------------------------------------------------------------------------
```

8.5.2.2 *Параметры задержки*

Параметры задержки имеют следующую структуру тройки:

Delays = EstDelay × TransDelay × RelDelay, где:

EstDelay = Nat

TransDelay = Nat^4

RelDelay = Nat^2

EstDelay — вид параметра задержки установления ТС, который имеет линейную структуру (таким образом определена единственная взаимно однозначная проекция).

TransDelay — вид параметра транзитной задержки, который имеет структуру четверки. Каждая проекция представляет транзитную задержку для отдельного направления передачи и скорости (а именно «максимальная» и «средняя», см. 10.3 ГОСТ 34.960). Это определение представлено как переименование вспомогательного определения, приведенного в 8.5.5.

RelDelay — вид параметра задержки освобождения, который имеет структуру двойки. Каждая проекция представляет задержку освобождения ТС для отдельного пользователя ТС (которому

сигнализируется об успешном освобождении, см. 10.7 ГОСТ 34.960). Форма этого определения —экземпляр общего определения (приведенного в 8.5.5), полученный использованием в качестве актуального типа параметра NatRepresentations, который определен в библиотеке типов данных LOTOS.

---*)

```
type TCDelays
is POThreeTuple actualizedby TCEstablishmentDelay, TransitDelay,
   TCReleaseDelay using
sortnames
        Delays for ThreeTuple
        EstDelay for Element
        TransDelay for Element2
        RelDelay for Element3
        Bool for FBool
opnnames
        TCEstablishment for First
        Transit for Second
        TCRelease for Third
        Delay for Tuple
endtype (* TCDelays *)
type TCEstablishmentDelay
is NatRepresentations
sorts
        EstDelay
opns
        EstDelay                              : Nat       —>EstDelay
        Time                                  : EstDelay  —> Nat
        _eq_, _ne_, _le_, _lt_, _ge_, _gt_ : EstDelay, EstDelay
                                              —> Bool
eqns
forall
        n, n1  :  Nat, e, e1  :  EstDelay
ofsort Nat
        Time(EstDelay(n)) = n;
ofsort Bool
        EstDelay(n) le EstDelay(n1) = n le n1;
        e lt e1 = (e le e1) and not(e1 le e);
        e eq e1 = (e le e1) and (e1 le e);
        e ne e1 = not(e eq e1);
        e ge e1 = e1 le e;
        e gt e1 = (e ge e1) and not(e1 ge e);
```

```
endtype (* TCEstablishmentDelay *)
type TransitDelay
is DTRateDirecitonQOSParameter renamedby
sortnames
        TransDelay for DQOSP
opnnames
        TransDelay for RDQOSP
endtype (* TransitDelay *)
type TCReleaseDelay
is PODoubleParameter actualizedby NatRepresentations using
sortnames
        Nat for Element
        Nat for Element2
        RelDelay for Pair
        Bool for FBool
opnnames
        AtCalling for First
        AtCalled for Second
        RelDelay for Pair
endtype (* TCReleaseDelay *)
type CLTransitDelay
is NaturalNumber renamedby
sortnames
        CLTransDelay for Nat
endtype (*CLTransitDelay *)
(* ----------------------------------------------------------------------
```

8.5.2.3 *Пропускная способность*

Параметры пропускной способности имеют следующую структуру четверки:

Throughput = Nat[4]

```
---------------------------------------------------------------------- *)
type TCThroughput
is DTRateDirectionQOSParameter renamedby
sortnames
        Throughput for RDQOSP
opnnames
        Throughput for RDQOSP
endtype (* TCThrouhput *)
(* ----------------------------------------------------------------------
```

8.5.2.4 *Вероятность отказа*

Параметры вероятности отказа имеют следующую структуру четверки:

Failures = $Prob^4$

```
-------------------------------------------------------------------------------------*)
type TCFailureProbabilities
is POFourTuple actualizedby Probability using
sortnames
        Prob for Element
        Prob for Element2
        Prob for Element3
        Prob for Element4
        Bool for FBool
        Failure for FourTuple
opnnames
        TCEstablishment for First
        Transfer for Second
        TCResilience for Third
        TCRelease for Fourth
        Failures for Tuple
endtype (* TCFailureProbabilites *)
(* ------------------------------------------------------------------------------------
```

8.5.2.5 *Коэффициент необнаруженных ошибок*

Параметры коэффициента необнаруженных ошибок (КНО) имеют следующую структуру двойки: КНО = $Prob^2$

Каждая проекция представляет КНО для отдельного направления передачи. Форма этого определения аналогична той, которая использована для задержки освобождения ТС (см. 8.5.2.2), но с другим типом актуального параметра, а именно Probability (см. 8.5.5). На самом деле, КНО определен в ГОСТ 34.960 как отношение измеренных значений, а не как вероятность. Однако это не имеет значения для образования типа, поскольку значения КНО (любая из двух проекций) и операции на КНО совпадают со значениями и операциями вероятности соответственно.

```
-------------------------------------------------------------------------------------*)
type TCResidualErrorRate
is POPair actualizedby NaturalNumber using
sortnames
        TPRER for Pair
        Nat for Element
        Nat for Element2
```

```
        Bool for FBool
opnnames
        RER for Pair
        Target for First
        Minimum for Second
endtype (* TCResidualErrorRate *)
(*-------------------------------------------------------------------------
```

8.5.3 *Приоритет ТС*

При помощи переименования натуральных чисел уровни приоритета представлены как полностью упорядоченное множество. В ГОСТ 34.960 указано, что число уровней приоритета ограничено. Это также применимо к реализации типа данных NaturalNumber.

```
-------------------------------------------------------------------------*)
type TCPriority
is NaturalNumber renamedby
sortnames
        TCPriority for Nat
opnnames
        Lowest for  0
          Higher for Succ
endtype (* TCPriority *)
(*-------------------------------------------------------------------------
```

8.5.4 *Защита ТС*

Упорядоченное множество из четырех констант представляет варианты защиты, определенные в 10.9 ГОСТ 34.960.

```
-------------------------------------------------------------------------*)
type TCProtection
is OrderedFourTuplet renamedby
sortnames
        TCProtection for Tuplet
opnnames
        NoProtection for TheOne
        Monitoring for TheOther
        Manipulation for TheThird
        FullProtection for TheFourth
endtype (* TCProtection *)
(*-------------------------------------------------------------------------
```

8.5.5 *Вспомогательные определения КУ*

В типе Probability вид Prob задает интервал действительных чи-

сел [0, 1] вместе с элементом Undefined, который представляет отношения, не входящие в этот интервал.

DTRateDirectionQOSParameter поддерживает определение параметра КУ в виде натурального числа, который имеет четыре компонента (или проекции), каждый из которых индексируется скоростью передачи данных и направлением передачи. Виды DTRate и DTDirection обеспечивают эти индексы. Компоненты генерируются функцией Proj. Конструкция этого типа представлена снизу вверх, используя определения типов DTDirectionQOSPar и DTDirectionQOSParameter для построения структуры четверки из двух двоек. Общее определение параметра PODoubleParameter см. в 8.7.

Примечание — Определение в 8.7 необходимо для понимания приведенных ниже определений. При первом чтении рекомендуется сначала прочесть 8.7.

---

*)

```
type Probability
is NaturalNumber
sorts
        Prob
opns
        _/_                 :       Nat, Nat        —> Prob
        Undefined           :                       —> Prob
        _eq_,_ne_,_le_,
_lt_,_ge_,_gt_  :  Prob, Prob                       —> Bool
eqns
forall
        m, n, j, k  :  Nat, p, q  :  Prob
ofsort Prob
        m gt n or (n eq 0) => m/n=Undefined;
        m le n, n ne 0, j ne 0 => (m * j) / (n * j) = m/n;
ofsort Bool
        m le n,  j le k, n ne 0,  k ne 0 => m / n le (j / k) = (m * k)
        le (j * n);
        m le n, n ne 0 => Undefined le (m/n) = false;
        p le Undefined = true;
        p lt q = (p le q) and not(q le p);
        p eq q = (p le q) and (q le p);
        p ne q = not(p eq q);
        p ge q = q le p;
        p gt q = (p ge q) and not (q ge p);
endtype (* Probability *)
```

```
type DTDirectionQOSPar
is POPair actualizedby NaturalNumber using
sortnames
        Nat for Element
        Nat for Element2
        DQOSP for Pair
        Bool for FBool
opnnames
        FromCalling for First
        FromCalled for Second
        DQP for Pair
endtype (* DTDirectionQOSPar *)
type DTRateDirectionQOSParameter
is POPair actualizedby DTDirectionQOSPar using
sortnames
        Nat for Element
        Nat for Element2
        DQOSP for Pair
        Bool for FBool
opnnames
        FromCalling for First
        FromCalled for Second
        DQP for Pair
endtype (* DTDirectionQOSPar *)
type DTRateDirectionQOSParameter
is POPair actualizedby DTDirectionQOSPar using
sortnames
        DQOSP for Element
        DQOSP for Element2
        Bool for FBool
        RDQOSP for Pair
opnnames
        Max for First
        Ave for Second
        RDQOSP for Pair
endtype (* DTRateDirectionQOSParameter *)
(*-----------------------------------------------------------------------------------------------
```

8.6 Параметры сервисных примитивов транспортного уровня

Вариант срочных данных и параметры причины разъединения в TSP определяются типами TEXOption и TDISReason соответственно. Структура параметра КУ определена в 8.5 посредством

TSQuality. Другими типами параметров TSP являются TAddress (см. 8.2) и OctetString (данные пользователя), которые импортируются из библиотеки типов данных LOTOS (см. раздел 7).

Примечание — Представление значений TDISeason соответствует определению в 14.2.1 ГОСТ 34.960, но без представления дополнительной информации и примеров, приведенных в примечаниях (см. 8.7 для определения TwoTuplet).

---*)

```
type TEXOption
is TwoTuplet renamedby
sortnames
        TEXOption for Tuplet
opnnames
        UseTEX for TheOne
        NoTEX for TheOther
endtype (* TEXOption *)
type TDISReason
is TwoTuplet renamedby
 sortnames
        TDISReason for Tuplet
opnnames
         User for TheOne
        Provider for TheOther
endtype (* TDISReason *)
(* ---------------------------------------------------------------------
```

8.7 Вспомогательные определения

GeneralIdentifier определяет бесконечное множество идентификаторов вместе с равенством на нем.

Element определен в библиотеке типов данных LOTOS, тогда как Element2, Element3 и Element4 являются его отдельными изоморфными копиями. Эти типы используются в определениях Pair, ThreeTuple и FourTuple, которые определяют кортежи общего вида из двух, трех или четырех значений, возможно разных видов, с равенством и проекциями. TwoTuplet и FourTuplet определяют множества, состоящие соответственно из двух и четырех элементов, наделенных булевым равенством. OrderedFourTuplet является расширением FourTuplet с общим упорядочением.

POElement — это общий тип Element с добавлением частичной упорядоченности, тогда как POElement2, POElement3 и POElement4 являются его отдельными изоморфными копиями. Эти типы используются при построении POPair, POThreeTuple и

POFourTuple, которые являются кортежами общего вида, расширенными частичной упорядоченностью.

```
---------- *)
type GeneralIdentifier
is Boolean
sorts
        Identifier
opns
        SomeIdentifier          :                    —> Identifier
        AnotherIdentifier       : Identifier         —> Identifier
        _eq_,_ne_               : Identifier, Identifier
                                                     —> Bool
eqns
forall
        a, a1   :   Identifier
ofsort Bool
        SomeIdentifier eq SomeIdentifier=true;
        AnotherIdentifier(a) eq SomeIdentifier=false;
        SomeIdentifier eq AnotherIdentifier(a)=false;
        AnotherIdentifier(a) eq AnotherIdentifier(a1)=a eq a1;
        a ne a1=not(a eq a1);
endtype (* GeneralIdentifier *)
type Element2
is Element renamedby
sortnames
        Element2 for Element
endtype (* Element2 *)
type Element3
is Element renamedby
sortnames
        Element3 for Element
endtype (* Element3 *)
type Element4
 is Element renamedby
sortnames
        Element4 for Element
endtype (* Element4 *)
type Pair
is Boolean, Element, Element2
sorts
        Pair
opns
```

```
        Pair          : Element, Element2     —> Pair
        First         : Pair                  —> Element
        Second        : Pair                  —> Element2
        _eq_, _ne_    : Pair, Pair            —> Bool
eqns
forall
        e1 : Element, e2 : Element2, p, p1 : Pair
ofsort Element
        First(Pair(e1, e2)=e1:
ofsort Element2
        Second(Pair(e1, e2))=e2;
ofsort Bool
        p=p1 => p eq p1=true;
        First(p) ne First(p1) => p eq p1=false;
        Second(p) ne Second(p1) => p eq p1=false;
        p ne p1=not(p eq p1);
endtype (*Pair *)
fype ThreeTuple'
is Element, Element2, Element3, Boolean
sorts
        ThreeTuple
opns
        Tuple     : Element, Element2, Element3  —> ThreeTuple
        First     : ThreeTuple                   —> Element
        Second    : ThreeTuple                   —> Element2
        Third     : ThreeTuple                   —> Element3
        _eq_, _ne_ : ThreeTuple, ThreeTuple      —> Bool
eqns
forall
        x, y : ThreeTuple, x1, y1 : Element, x2, y2 : Element2,
        x3, y3 : Element3
ofsort Element
        First(Tuple(x1, x2, x3))=x1;
ofsort Element2
        Second(Tuple(x1, x2, x3))=x2;
ofsort Element3
        Third(Tuple(x1, x2, x3))=x3;
ofsort Bool
        First(x) eq First(y), Second(x) eq Second(y), Third (x) eq
        Third(y) => x eq y=True;
        not(First(x) eq First(y)) => x eq y=False;
        not(Second(x) eq Second(y)) => x eq y=False;
        not(Third(x) eq Third(y)) => x eq y=False;
```

```
endtype (* ThreeTuple *)
type FourTuple
is Element, Element2, Element3, Element4, Boolean
sorts
        FourTuple
opns
        Tuple  :  Element, Element2, Element3, Element4
                                                    —> FourTuple
      First          : FourTuple                    —> Element
      Second         : FourTuple                    —> Element2
      Third          : FoureTuple                   —> Element3
      Fourth         : FoureTuple                   —> Element4
      _eq_, _ne_     : FourTuple, FourTuple         —> Bool
eqns
forall
        x1, y1      : Element, x2, y2 : Element2, x3, y3 : Element3,
        x4, y4      : Element4
ofsort Element
        First(Tuple(x1, x2, x3, x4)) = x1;
ofsort Element2
        Second(Tuple(x1, x2, x3, x4)) = x2;
ofsort Element3
        Third(Tuple(x1, x2, x3, x4)) = x3;
ofsort Element3
        Fourth(Tuple(x1, x2, x3, x4)) = x4;
ofsort Bool
        Tuple(x1, x2, x3, x4) eq Tuple (x1, x2, x3, x4) = True;
        x1 ne y1 => Tuple(x1, x2, x3, x4) eq Tuple(y1, y2, y3, y4)
        = False;
        x2 ne y2 => Tuple(x1, x2, x3, x4) eq Tuple(y1, y2, y3, y4)
        = False;
        x3 ne y3 => Tuple(x1, x2, x3, x4) eq Tuple(y1, y2, y3, y4)
        = False;
        x4 ne y4 => Tuple(x1, x2, x3, x4) eq Tuple(y1, y2, y3, y4)
        = False;
endtype (* FourTuple *)
type TwoTuplet
is Boolean, NaturalNumber
sorts
        Tuplet
opns
        TheOne, TheOther        :                   —> Tuplet
        _eq_, _ne_              : Tuplet, Tuplet    —> Bool
        h                       : Tuplet            —> Nat
```

```
eqns
forall
        u, v : Tuplet
ofsort Nat
        h(TheOne) = 0;
        h(TheOther) = Succ(h(TheOne));
ofsort Bool
        u eq u = h(u) eq h(v);
        u ne v = not(u eq v);
endtype (* TwoTuplet *)
type FourTuplet
is TwoTuplet
opns
        TheThird, TheFourth      :          —> Tuplet
eqns
ofsort Nat
        h(TheThird) = Succ(h(TheOther));
        h(TheFourth) = Succ(h(TheThird));
endtype (* FourTuplet *)
type OrderedFourTuplet
 is FourTuplet
opns
        _lt_, _le_, _ge_,
_gt_  :   Tuplet, Tuplet     —> Bool
eqns
forall
        x, y  :  Tuplet
ofsort Bool
        x lt y = h(x) lt h(y);
        x le y = h(x) le h(y);
        x ge y = h(x) ge h(y);
        x gt y = h(x) gt h(y);
endtype (* OrderedFourTuplet *)
type POElement
is Element
formalopns
        _le_, _lt_, _ge_,
_gt_  :  Element, Element     —> FBool
endtype (* POElement *)
type POElement2
is POElement renamedby
sortnames
        Element2 for Element
```

```
endtype (* POElement2 *)
type POElement3
is POElement renamedby
sortnames
        Element3 for Element
endtype (* POElement3 *)
type POElement4
is POElement renamedby
sortnames
        Element4 for Element
endtype (* POElement4 *)
type BasicPOPair
is Pair actualizedby POElement, POElement2 using
endtype (* BasicPOPair *)
type POPair
is BasicPOPair
opns
        _le_, _lt_, _ge_,
_gt_  : Pair, Pair        —> Bool
eqns
forall
        x, y  :  Pair
ofsort Bool
        First(x) le First(y), Second(x) le Second(y) =>x  le y=
        true;
        not(First(x) le First(y)) =>x le y=false;
        not(Second(x) le Second(y)) =>x le y=false;
        x ne y =>x lt y=x le y;
        x eq y =>x lt y=false;
        x ge y=y le x;
        x ne y =>x gt y=x ge y;
        x eq y =>x gt y=false;
endtype (* POPair *)
type BasicPOThreeTuple
is ThreeTuple actualizedby POElement,  POElement2,  POElement3
using
endtype (* BasicPOThreeTuple *)
type POThreeTuple
is BasicPOThreeTuple
opns
        _le_, _lt_, _ge_,  _gt_ : ThreeTuple, ThreeTuple  —> Bool
eqns
forall
```

```
        x, y  :  ThreeTuple
ofsort Bool
        First(x) le First(y), Second(x) le Second(y), Third(x)  le
        Third(y) =>  x le y=true,
        not(First(x) le First(y)) =>  x le y=false;
        not(Second(x) le Second(y)) =>  x le y=false;
        not(Third(x) le Third(y)) =>  x le y=false;
        x ne y =>  x lt y=x le y;
        x ge y=y le x;
        x ne y =>  x gt y=x ge y;
endtype (* POThreeTuple *)
type BasicPOFourTuple
is FourTuple actualizedby POElement, POElement2, POElement3,
POElement4 using
endtype (* BasicPOFourTuple *)
type POFourTuple
is BasicPOFourTuple
opns
        _le_, _lt_, _ge_,
_gt_ : FourTuple, FourTuple  —>  Bool
af*
forall
        x, y  :  FourTuple
ofsort Bool
        First(x) le First(y), Second(x) le Second(y), Third(x)  le
        Third(y), Fourth(x) le Fourth(y) =>  x le y=true;
        not(First(x) le First(y)) =>  x le y=false;
        not(Second(x) le Second(y)) =>  x le y=false;
        not(Third(x) le Third(y)) =>  x le y=false;
        not(Fourth(x) le Fourth(y)) =>  x le y=false;
        x ne y =>  x lt y=x le y;
        x ge y=y le x;
        x ne y =>  x gt y=x ge y;
endtype (* POFourTuple *)
(* ----------------------------------------------------------------------
```

9 ГЛОБАЛЬНЫЕ ОГРАНИЧЕНИЯ

Определение поведения поставщика услуг разделено на две независимые части: поставщик услуг в режиме-с-установлением-соединения и поставщик услуг в режиме-без-установления-соединения. Определение поставщика услуг в режиме-с-установлением-соединения представлено в сочетании с отдельными ограничения-

ми. TConnections описывается как способный поддерживать потенциально бесконечное число независимых ТС. Каждое ТС представляется отдельным экземпляром TConnection. Получаемая в результате структура показана на рисунке 1. TConnectionless описано в разделе 16. В разделе 10—15 описаны ограничения, относящиеся к транспортным услугам в режиме-с-установлением-соединения.

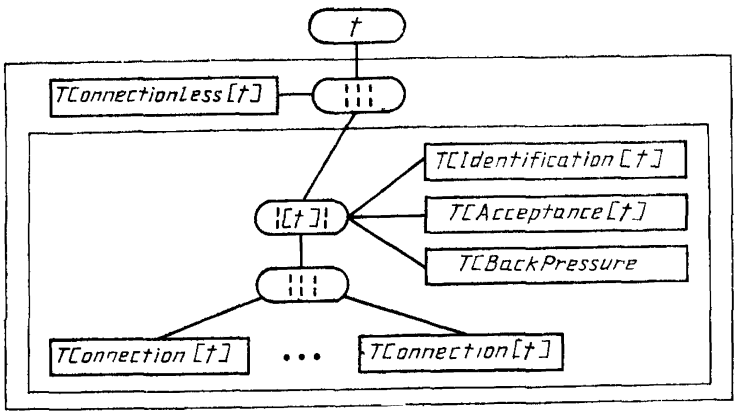

Рисунок 1 — Структура услуг транспортного уровня

Примечания

1 Каждый экземпляр TConnection может закончиться. Общий TConnections никогда не может закончиться, поскольку всегда существует вероятность того, что будет вызван новый экземпляр его компонента TConnection.

2 В любой момент времени может быть активно любое число ТС. Недетерминизм поставщика услуг в ограничении этой потенциально бесконечной параллельности специфицируется как отдельное ограничение (см. раздел 14).

---

*)

```
behaviour
        TSConnectionless [t]
        |||
        TConnections [t] || TCIdentification [t] ||
        TCAcceptance [t]
        ||
        TBackpressure[t]
where
process TConnections [t]  :  noexit : =
```

```
        TConnection [t] >> stop
        |||
        TConnections [t]
endproc (* TConnections *)
(*-----------------------------------------------------------------------------
```

10 ОБЕСПЕЧЕНИЕ ТРАНСПОРТНОГО СОЕДИНЕНИЯ

Требования к поведению поставщика услуг, связанные с обеспечением одного ТС, распадаются на два класса, а именно ограничения, которые являются локальными для оконечной точки ТС, и межоконечные ограничения. Ограничения первого класса связаны с поведением в оконечной точке ТС в зависимости от истории примитивов, выполненных в этой же оконечной точке ТС. Ограничения второго класса связаны с поведением в оконечной точке ТС в зависимости от истории примитивов, выполненных в другой оконечной точке ТС.

Следует формально определить подходящее понятие «история». В частности, удобно принимать во внимание только те события, которые могут влиять на будущее поведение поставщика услуг, не учитывая, таким образом, те события, которые ни на что больше не влияют: такое понятие известно как «влияющая история». Для локальных ограничений влияющая история представлена состоянием процесса в соответствии с диаграммой переходов состояний, представленной на рисунке 5 ГОСТ 34.960. Для межоконечных ограничений влияющая история представлена параметрами процесса, чья структура и операции формулируются при помощи специально созданного определения типа данных (см. 12.3.2). Эти операции позволяют формулировать требования, соответствующие таблице 1 и рисунку 4 ГОСТ 34.960, способом, не зависящим от модели.

Локальные органичения представлены двумя независимыми параллельными процессами, представляющими собой два разных экземпляра процесса TAEP (см. раздел 11) и соответственно ограничивают взаимодействия с вызывающим и вызываемым пользователями транспортных услуг. TSUserRole определяет роли вызывающего и вызываемого пользователя транспортных услуг. Межоконечные ограничения представлены процессом TCEPAssociation (см. раздел 12), который соотносит индикации, возникающие на каждом конце ТС, с соответствующими запросами, возникающими на другом конце ТС. TSPDirection определяет направления примитивов запроса и индикации в полном соответствии с разделом 5.

Этим процессом специфицируется недетерминированность поставщика, которая влияет на данную взаимосвязь. Кроме того, TCEPAssociation определяет недетерминизм, связанный с выполнением примитивов, генерированных поставщиком.

Примечание — Техническое замечание о завершении: завершение обоих процессов, представляющих концы соединения — явно достаточное представление конца времени жизни этого соединения Вот почему параллельный компонент, содержащий TCEPAssociation, описан (с использованием конструкции [> exit) как способный завершиться в любой момент времени, тогда как TCEPAssociation — бесконечный процесс.

```
type TSUserRole
is TwoTuplet renamedby
sortnames
TSUserRole for Tuplet
opnnames
CallingRole for TheOne
CalledRole for TheOther
endtype (* TSUserRole *)
type TSPDirection
is TwoTuplet renamedby
sortnames
TSDirection for Tuplet
opnnames
Request for TheOne
Indication for TheOther
endtype (* TSPDirection *)
process TConnection [t]  :  exit : =
(TCEP [t] (CallingRole) ||| TCEP [t] (CalledRole))
        ||
(TCEPAssociation [t] [> exit)
endproc (* TConnection *)
(*
```

11 ЛОКАЛЬНЫЕ ОГРАНИЧЕНИЯ ДЛЯ ОКОНЕЧНОГО ПУНКТА ТС

11.1 Общее описание

Первые три процесса следующей декомпозиции локальных ограничений на взаимодействия одного ТС в каждой оконечной точке отражают структуру события: это делается с целью отделения ограничений, применяемых к каждому компоненту структуры события, от ограничений, применяемых к другим компонентам. Четвертый процесс специфицирует ограничение, упомянутое в примечании к 8.4.1.

```
---------------------------------------------------------------------------- *)
process TCEP [t] (role : TSUserRole) : exit : =
        TCEPAddress [t]
        || TCEPIdentification [t]
        || TCEPSPOrdering [t] (role)
        || TCEPUserData [t]
endproc (* TCEP *)
(* ----------------------------------------------------------------------------
```

11.2 Адрес и идентификация оконечной точки ТС

Локальные ограничения на адресную и идентификаторную части событий в оконечной точке ТС имеют сходную форму: значение определяется при первом событии совместно с пользователем транспортных услуг, а затем остается постоянным.

Примечание — Завершение обоих процессов допускается в любой момент времени: конец локального (т. е. в оконечной точке ТС) времени жизни ТС на самом деле определяется локальным упорядочением сервисных примитивов (см. 11.3.1).

```
---------------------------------------------------------------------------- *)
process TCEPAddress [t] : exit : =
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ; ConstantTA [t] (ta)
[> exit
endproc (* TCEPAddress *)
process ConstantTA [t] (ta : TAddress) : noexit : =
t!ta ?tcei : TCEI ?tsp : TSP ; ConstantTA [t] (ta)
endproc (* ConstantTA *)
process TCEPIdentification [t] : exit : =
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ; ConstantTCEI [t] (tcei)
[> exit
endproc (* TCEPIdentification *)
process ConstantTCEI [t] (tcei : TCEI) : noexit : =
t?ta : TAddress !tcei ?tsp : TSP ; ConstantTCEI [t] (tcei)
endproc (* ConstantTCEI *)
(* ----------------------------------------------------------------------------
```

11.3 Локальное упорядочение сервисных примитивов в оконечном пункте ТС

11.3.1 *Общее описание*

TCEPSPOrdering определяет ограничения на возможные последовательности примитивов в одном оконечном пункте ТС (см. рисунок 5 и таблицу 4 ГОСТ 34.960), применяемые к одному ТС.

Поскольку задана спецификация локальных ограничений, постольку в любой момент времени примитивы транспортных услуг, которые могут быть выполнены, зависят только от истории примитивов, выполненных в этом оконечном пункте ТС. Влияющие аспекты этой истории в основном представлены ниже в именах процессов из компонентов TCEPOrdering вместе с некоторыми параметрами.

Фаза установления ТС в оконечном пункте ТС представлена в виде последовательности TCEPConnect1 и TCEPConnect2. Это сделано для возможного освобождения ТС даже до успешного установления ТС (предотвращая его таким образом), но только после начала времени жизни ТС. Примитив Т-СОЕДИНЕНИЕ, выполненный в TCEPConnect1, является влияющей историей для TCEPConnect2, поскольку к Т-СОЕДИНЕНИЕ ответ/подтверждение применимы ограничения согласования, которые зависят от Т-СОЕДИНЕНИЕ индикация/запрос.

Успешное установление ТС позволяет войти в фазу передачи информации, которая в каждом оконечном пункте ТС представлена TCEPDataTransfer. Поведение в этой фазе не зависит от роли оконечного пункта ТС. Значение x сообщает результат согласования варианта срочных данных.

Освобождение ТС в оконечной точке ТС состоит из выполнения примитива Т-РАЗЪЕДИНЕНИЕ, как представлено в TCEPRelease. Это может произойти в любой момент времени после первого примитива Т-СОЕДИНЕНИЕ.

Примечание — Последняя альтернатива в определении TCEPSPOrdering введена для обеспечения возможности завершения установления ТС путем освобождения ТС локально в вызывающем оконечном пункте ТС, без выполнения каких-либо примитивов на другом (потенциально но в действительности никогда не наблюдаемом) конце этого ТС.

```
-------------------------------------------------------------------------------------------*)
process TCEPSPOrdering [t] (role : TSUserRole) : exit : =
(TCEPConnect1 [t] (role)
≫ accept tsp : TSP in
(TCEPConnect2 [t] (tsp) ≫ accept x : TEXOption in
TCEPDataTransfer [t] (x))
        [> TCEPRelease [t])
[ ] [role=CalledRole] —> exit
endproc (* TCEPSPOrdering *)
(* ------------------------------------------------------------------------------------------
```

11.3.2 *Фаза установления соединения в оконечном пункте ТС*

В 8.4.3.4 дано определение булевой функции IsValidTCON2For,

которая представляет требования согласования транспортных услуг. Определение TSUserRole дано в разделе 10.

```
-------------------------------------------------------------------------------- *)
process TCEPConnect1 [t] (role : TSUserRole) : exit(TSP) : =
[role=CallingRole]
—>
t?ta : TAddress ?tcei : TCEI ?tcr : TSP [IsTCONreq(tcr) and
(ta IsCallingOf tcr)] ; exit (tcr)
        [ ]
[role=CalledRole]
—>
t ?ta : TAddress ?tcei : TCEI ?tci : TSP [IsTCONind(tci) and
(ta IsCalledOf tci)] ; exit (tci)
endproc (* TCEPConnect1 *)
 process TCEPConnect2 [t] (tc1 : TSP) : exit(TEXOption) : =
t ?ta : TAddress ?tcei : TCEI ?tc2 : TSP [tc2 IsValidTCON2For tc1];
   (choice x : TEXOption [ ] [x IsTEXOptionOf   tc2] —> exit (x))
endproc (* TCEPConnect2 *)
(* ------------------------------------------------------------------------------
```

11.3.3 *Фаза передачи данных в оконечном пункте ТС*

```
-------------------------------------------------------------------------------- *)
process TCEPDataTransfer [t] (x : TEXOption): noexit :=
TCEPNormalDataTransfer [t]
          |||
[x=UseTEX] —> TCEPExpeditedDataTransfer [t]
endproc (* TCEPDataTransfer *)
process TCEPNormalDataTransfer [t] : noexit : =
t ?ta:TAddress ?tcei : TCEI ?tsp : TSP [IsTDT(tsp)] ;
TCEPNormalDataTransfer [t]
endproc (* TCEPNormalDataTransfer *)
process TCEPExpeditedDataTransfer [t] : noexit : =
t ?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsTEX(tsp)] ;
TCEPExpeditedDataTransfer [t]
endproc (* TCEPExpeditedDataTransfer *)
(* ------------------------------------------------------------------------------
```

11.3.4 *Фаза освобождения в оконечном пункте ТС*

```
-------------------------------------------------------------------------------- *)
process TCEPRelease [t] : exit : =
t ?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsTDIS(tsp)] ; exit
```

```
endproc (* TCEPRelease *)
(*
```

11.4 Ограничения на данные пользователя

Длина параметра данных пользователя в примитивах транспортных услуг ограничивается TCEPUserData, что изложено в 12.2.7; 13.1.13; 13.2.3 и 14.2.2 ГОСТ 34.960, отражаемых здесь.

```
*)
process TCEPUserData [t] : exit : =
t ?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsValidUserData(tsp)] ;
        TSEPUserData [t] [ ] exit
endproc (* TCEPUserData *)
type ValidUserData
is TransportServicePrimitive
opns
IsValidUserData          : TSP          —> Bool
eqns
forall
t : TSP
ofsort Bool
IsTCON(t) =>  IsValidUserData(t) = Length(UserData(t)) le
NatNum(3+Dec(2));
IsTDT(t) =>  IsValidUserData(t) = Length(UserData(t)) gt 0;
IsTDIS(t) =>  IsValidUserData(t) = Length(UserData(t)) le
NatNum(6+Dec(4));
IsTEX(t)
=>  IsValidUserData(t)
= Length(UserData(t) gt 0) and (Length(UserData(t)) le NutNum
(1+Dec(6)));
endtype (* ValidUserData *)
(*
```

12 МЕЖКОНЕЧНЫЕ ОГРАНИЧЕНИЯ ДЛЯ ОДНОГО ТС

12.1 Общее описание

После выполнения первого запроса TCEPAssociation расщепляется на два экземпляра TAssoc1 — по одному экземпляру на каждое направление передачи. Эти два экземпляра могут быть созданы независимо и параллельно, поскольку они образуют отдельные части поведения. В самом деле, когда бы один из них ни вызывал взаимодействие, другой не будет вовлечен в это взаимодействие.

Примечание — Начальным взаимодействием по ограничениям может быть только запрос. Это должен быть Т-СОЕДИНЕНИЕ запрос в силу дополнительных локальных ограничений (см. 11.3.2). Разделение между двумя экземплярами корректно только в том случае, если гарантируется, что оконечный пункт, где один из них обрабатывает запросы, является оконечным пунктом, где другой обрабатывает индикации. По этой причине введены параметры адреса, идентификации и направления примитивов в TAssoc1 (см. 12.2), при помощи которых TAssoc1 обрабатывает запросы на одном конце ТС и индикации на другом конце.

```
*)
process TCEPAssociation [t] : noexit : =
t ?ta : TAddress ?tcei : TCEI ?tcr : TSP [IsTReq(tcr)] ;
        (TAssoc1 [t] (ta, tcei, Request, Indication, Append(tcr,
        NoTReqs))
        |||
        TAssoc1 [t] (ta, tcei, Indication, Request, NoTReqs))
endproc (* TCEPAssociation *)
(*
```

12.2 Однонаправленность

TAssoc1 соотносит индикации, которые могут выполняться на одном конце ТС, с историей запросов, выполненных на другом его конце, учитывая недетерминизм поставщика, влияющий на эту взаимосвязь. Самым сложным подпроцессом TAssoc1 является TCRecToInd (см. 12.3.1), который представляет межоконечные ограничения, указывающие взаимодействия в t, связанные с примитивами. Параметр TCRecToInd представляет историю запросов, которые соотносятся с возможными будущими индикациями. Определение TCEPHalf, видимо, требует небольшого дальнейшего пояснения: см. в 11.2 ConstantTA и ConstantTCEI, в 8.4.2.1 IsTReq и IsTInd, в 8.3 TId.

```
*)
process TAssoc1 [t] (ta : TAddress, tcei : TCEI, d1, d2, :
        TSPDirection, rh : TReqHistory) : noexit : =
((TCEPHalf [t] (d1) || ConstantTA [t] (ta) || ConstantTCEI [t]
(tcei))
        |||
TCEPHalf [t] (d2)
        ||
t?ta1 : TAddress ?tcei : TCEI ?tsp : TSP [TId(ta1, tcei1) ne TId(ta,
tcei)] ; (ConstantTA [t] (ta1) || ConstantTCEI [t] (tcei1))) ||
TCReqToInd [t] (rh)
endproc (* TAssoc1 *)
process TCEPHalf [t] (dir : TSPDirection) : noexit : =
```

```
[dir = Request]
—> t?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsTReq(tsp)] ;
TCEPHalf
[t] (dir)
[ ]
[dir = Indication]
—> t?ta : TAddress ?tcei : TCEI ?tsp : TSP (IsTInd(tsp))] ;
TCEPHalf
[t] (dir)
endproc (* TCEPHalf *)
(*-------------------------------------------------------------------------
```

12.3 Межоконечный недетерминизм

12.3.1 *Общее описание*

Как было сказано выше, TCReqToInd параметризуется историей запросов rh, выполненных на одном конце, которые могут влиять на возможные будущие индикации на другом конце. Значения вида TReqHistory являются основными последовательностями, на которых определены некоторые операции, позволяющие рассматривать только влияющие элементы истории (подробнее см. 12.3.2).

Непосредственная форма определения TCReqToInd — это очень простая праворекурсивная форма. В любой момент времени процесс TSPEvent специфицирует ограничения на следующее наблюдаемое событие; первый параметр TSPEvent — это основа представления недетерминизма (операция Tops определена в 12.3.2). После выполнения этого события TCReqToInd выполняется с обновленным значением параметра.

```
-------------------------------------------------------------------------*)
process TCReqToInd [t] (rh : TreqHistory) : noexit :=
TSPEvent [t] (Tops(rh), rh) ≫ accept rh1 : TReqHistory in
TCReqToInd [t] (rh1)
endproc (* TCReqToInd *)
(*-------------------------------------------------------------------------
```

12.3.2 *Определение данных*

Первое определение обеспечивает основную конструкцию историй примитивов запроса посредством операций NoTReqs (пустая история) и OnTopOf (для расширения истории более ранним запросом). Также введены еще несколько булевых функций с обычной интерпретацией.

Примечание — Ради полноты OnTopOf нужно определять также и для случая, когда ее первый аргумент — индикация; в этом случае история остает-

ся без изменения. Этот «безэффектный» подход на индикации соблюдается также и во втором определении

Второе определение расширяет основной тип данных четырьмя операциями, необходимыми для формулировки межконечных ограничений:

a) Reduce — описывает историю своего верхнего элемента (т. е. самого раннего по времени выполненного запроса).

b) Remove — удаляет заданный запрос из заданной истории; если последний имеет далее много экземпляров, то удаляется первый экземпляр.

c) Append — добавляет запрос к истории в качестве последнего элемента. Заметим, что Append действует как OnTopOf, но с другого конца истории.

d) Tops — создает историю, состоящую из тех примитивов в аргументе истории, которые могут привести к последующей индикации.

Введены также другие функции, а именно TDISTops и TEXDISTops, для выразительности определения. TDISTops и TEXDISTops дают непосредственные результаты для вычисления Tops.

В соответствии с таблицей 1 ГОСТ 34.960 определен следующий порядок значимости примитивов:

Т-ДАННЫЕ < Т-СРОЧНЫЕ-ДАННЫЕ
< Т-РАЗЪЕДИНЕНИЕ,
Т-СОЕДИНЕНИЕ < Т-РАЗЪЕДИНЕНИЕ,

причем между примитивами одной услуги нет порядка значимости.

Tops определяется следующим образом: для любого данного значения rh вида TReqHistory, Tops(rh) представляет собой последовательность запросов, которая:

a) содержит только запросы различных услуг, и

b) содержит запрос t тогда и только тогда, когда t содержится в rh и все запросы перед ним в rh (т. е. выполненные раньше по времени) являются ниже по значимости, чем t, и

c) сохраняет порядок запросов в rh.

---*)

```
type TransportServiceBasicTSPRequestHistory
is TransportServicePrimitive
sorts
TreqHistory
opns
NoTReqs          :                          —> TreqHistory
_OnTopOf_        : TSP, TreqHistory         —> TreqHistory
```

```
_IsTopOf_        : TSP, TreqHistory          —> Bool
IsEmpty          : TreqHistory               —> Bool
_eq_, _ne_       : TReqHistory, TReqHistory  —> Bool
eqns
forall
t, t1 : TSP, h, h1 : TReqHistory
ofsort TReqHistory
IsTInd(t) =>  t OnTopOf h=h;
ofsort Bool
IsEmpty(NoTReqs) =true;
IsTReq(t) => IsEmpty(t OnTopOf h) =false;
NoTReqs eq NoTreqs=true;
IsTReq(t) => NoTReqs eq (t OnTopOf h) =false;
IsTReq(t) => t OnTopOf h eq NoTReqs=false;
IsTReq(t), IsTReq(t1)
=> t OnTopOf h eq (t1 OnTopOf h1) = (t eq t1) and (h eq h1);
h ne h1=not(h eq h1);
t IsTopOfNoTReqs=false;
IsTReq(t) => t OnTopOf (t1 OnTopOf h1) =t eq t1;
endtype (* TransportServiceBasicTSPRequestHistory *)
type TransportServiceTSPRequestHistory
is TransportServiceBasicTSPRequestHistory
opns
Reduce                  : TReqHistory       —> TReqHistory
Remove                  : TSP, TReqHistory  —> TReqHistory
Append                  : TSP, TReqHistory  —> TReqHistory
Tops, TDISTops, TEXDISTops
                        : TReqHistory       —> TReqHistory
eqns
forall
t, t1  :  TSP, h, h1  :  TreqHistory
ofsort TReqHistory
Reduce(NoTReqs) =NoTReqs;
IsTReq(t) => Reduce(t OnTopOf h) =h;
Remove(t, NoTReqs) =NoTReqs;
t eq t1 => Remove(t, t1 OnTopOf h1) =h;
t ne t1 => Remove(t, t1 OnTopOf h1) =t1 OnTopOf Remove(t, h1);
Append(t, NoTReqs) =t OnTopOf NoTReqs;
IsTReq(t1) => Append(t, t1 OnTopOf h1) =t1 OnTopOf Append(t,
h1);
Tops(NoTReqs) =NoTReqs;
IsTReq(t), IsTDIS(t) => Tops(t OnTopOf h) =t OnTopOf
```

```
NoTReqs;
IsTReq(t), IsTEX(t) or IsTCON(t) => Tops(t OnTopOf h) =
t OnTopOf TDISTops(h);
IsTReq(t), IsTDT(t) => Tops(t OnTopOf h) = t OnTopOf
TEXDISTops(h);
TDISTops(NoTReqs) = NoTReqs;
IsTReq(t), IsTDIS(t) => TDISTops(t OnTopOf h) = t OnTopOf
NoTReqs;
IsTReq(t), not(IsTDIS(t)) => TDISTops(t OnTopOf h) =
TDISTops(h);
TEXDISTops(NoTReqs) = NoTReqs;
IsTReq(t), IsTDIS(t) => TEXDISTops(T OnTopOf h) = TDISTops(t
OnTopOf h);
IsTReq(t), IsTEX(t) => TEXDISTops(t OnTopOf h) = t OnTopOf
TDISTops(h);
IsTReq(t), not(IsTEX(t)), not(IsTDIS(t))
=> TEXDISTops(t OnTopOf h) = TEXDISTops(h);
endtype (* TransportServiceTSPRequestHistory *)
(*-----------------------------------------------------------------------------------------
```

12.3.3 *Определения процессов*

Для заданной истории запросов rh TSPEvent специфицирует ограничения на возможное следующее событие. Параметр erh — это последовательность выполненных запросов, достаточная для определения следующей возможной индикации.

Следующие четыре требования формулируются для обслуживания недетерминизма поставщика в плане его участия в следующем событии.

a) Поставщик никогда не отказывается участвовать в событии запроса (см. также определение TSPReqEvent ниже).

Примечание — Возможное отклонение запросов определяется процессами TCAcceptance и TBackpressure (см. соответственно разделы 14 и 15).

b) Может быть выполнена только та индикация, которая относится к верхнему запросу непустой erh, соответствующей IsIndicationOf (см. 8.4.3.4). Если такая индикация возникает, соответствующий ей запрос удаляется из rh.

Примечание — Для заданного запроса поставщик озаглавливается, чтобы автономно определить значения параметров индикации, при условии, что они удовлетворяют этому требованию.

c) Для любого заданного верхнего запроса непустого erh поставщик может представить не связанную с ним индикацию только в том случае, если это вызывает участие:

1) в индикации, связанной с запросом следующей более высокой значимости в erh, если таковой существует, или

2) в генерированной поставщиком индикации (см. 8.4.3.4).

В случае 1) требования b), c) и d) применяются к erh, без его верхнего запроса.

d) Если пользователь отклоняет индикацию, то должна быть представлена индикация, соответствующая случаям 1) или 2) требования c)

```
----------------------------------------------------------------------------*)
process TSPReqEvent [t] (rh : TReqHistory) : exit(TReqHistory) :=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsTReq(tsp)] ; exit
(Append(tsp, rh))
endproc (* TSPReqEvent *)
process TSPEvent [t] (erh, rh : TReqHistory) : exit(TReqHistory) :=
TSPReqEvent [t] (rh)
        [ ]
[not(IsEmpty(erh))]
— >
(choice tspr, tspi : TSP
        [ ]
[(tspr IsTopOf erh) and (tspi IsIndicationOf tspr)]
—>
i ;
        (t ?ta : TAddress ?tcei : TCEI !tspi ; exit (Remove(tspr, rh))
        [ ] i ; TSPEvent [t] (Reduce(erh), rh)))
        [ ]
[IsEmpty(erh)]
—>
(choice tdi : TSP
        [ ]
[ProviderGeneratedInd(tdi)]
—>
i ;
(t?ta : TAddress ?tcei : TCEI !tdi ; exit (rh) [ ] TSPReqEvent [t]
(rh)))
endproc (* TSPEvent *)
(*----------------------------------------------------------------------------
```

13 ИДЕНТИФИКАЦИЯ ТРАНСПОРТНЫХ СОЕДИНЕНИЙ

Любые два разных экземпляра TConnection, которые одновременно имеют доступ к одному ПДУТ, должны быть различимы

для пользователя транспортных услуг. Это достигается при помощи идентификатора оконечного пункта ТС — tcei, который передается с каждым сервисным примитивом на каждый ПДУТ. Следовательно, требуется, чтобы:

a) на любом ПДУТ в любой момент времени никакой tcei не мог быть назначен более чем одному TConnection и

b) каждый TConnection использовал один и тот же tcei на каждом ПДУТ в течение всего времени существования ТС, который он представляет.

Если последнее ограничение можно специфицировать внутри определения TConnection (см. 11.2), то первое ограничение имеет более глобальный характер и ниже представлено процессом TCIdentification.

Для каждого ПДУТ хранится след используемых идентификаторов оконечного пункта ТС при помощи параметра Use, который является множеством пар вида TId- TAddress×TCEI (см. TCEIdentification в 8.3). Определение TCEIdentifications представляет эти множества Use вначале пустыми. Пара (ta, tcei) находится в Use тогда и только тогда, когда tcei назначается некоторому ТС, имеющему доступ к ПДУТ с адресом ta.

TCIdent позволяет передавать любой запрос или индикацию Т-СОЕДИНЕНИЕ любому заданному ПДУТ с адресом ta только с таким tcei, чтобы пара (ta, tcei) не присутствовала в Use. На другие примитивы это ограничение не налагается, но при выполнении примитива Т-РАЗЪЕДИНЕНИЕ соответствующая пара (ta, tcei) удаляется из Use.

Примечание — Необходимо учитывать следующую техническую деталь Insert(e, s)={e} U s Следовательно, Insert(e, s)=s всякий раз, когда e C s.

---

*)

```
type TCEIdentifications
is Set actualizedby TCEIdentification using
sortnames
TId for Element
Bool for FBool
TIds for Set
endtype (* TCEIdentifications *)
process TCIdentification [t] : noexit :=
TCIdent [t] ({ } of TIds)
endproc (* TCIdentification *)
process TCIdent [t] (Use : TIds) : noexit :=
t ?ta : TAddress ?tcei : TCEI ?tsp : TSP [IsTCON1(tsp) implies
(TId(ta, tcei) NotIn Use)];
```

```
        (let ti : TId=TId(ta, tcei)
        in
[not(IsTDIS(tsp))] —> TCIdent [t] (Insert(ti, Use))
        [ ]
[IsTDIS(tsp)] —> TCIdent [t] (Remove(ti, Use)))
endproc (* TCIdent *)
(*-------------------------------------------------------------------------------------------------
```

14 ПРИНЯТИЕ ТРАНСПОРТНЫХ СОЕДИНЕНИЙ

В любой момент времени поставщик услуг способен принять установление новых соединений только в конечном множестве пунктов доступа к услугам и в оконечных пунктах соединения. Это определяется процессом TCAcceptance, который внутренне выбирает конечное множество пар (ta, tcei) перед участием в каком-либо взаимодействии. Если взаимодействие образует новое ТС, то оконечный пункт, где произошло это взаимодействие, должен быть среди тех, которые представлены AcceptTC.

Однако при каждом выборе AcceptTC множество оконечных пунктов, где могут быть образованы новые соединения, является в действительности подмножеством AcceptTC, вследствие ограничения разделения на идентификацию ТС (см. раздел 3). Более точно новое соединение может быть образовано только с парой (ta, tcei), которая находится в AcceptTC, а не в Use. Следовательно, при каждом выборе AcceptTC множество оконечных пунктов, где могут быть образованы новые соединения, представлено разностью AcceptTC — Use.

Поставщику транспортных услуг присущ внутренний недетерминизм при динамическом выборе того, сколько и какие оконечные пункты можно выделить для новых соединений, при условии выполнения минимальных функциональных требований — если нет активных ТС, поставщик услуг должен быть способен принять хотя бы одно ТС, т. е. подмножество AcceptTC, где могут быть действительно приняты новые ТС, должно быть в этом случае непустым.

Примечание — В самом деле, требование минимальной функциональности эквивалентно более простому требованию, чтобы AcceptTC было непустым в любом случае, если принимать во внимание ограничения, наложенные TCIdentification.

Это действительно так, поскольку:

а) если нет активных ТС, то Use пустое, таким образом подмножество AcceptTC, где могут быть приняты новые ТС, является самим AcceptTC, в то время как

b) если активно несколько ТС, выбор непустого AcceptTC еще допускает, что подмножество, где могут быть приняты новые ТС, может быть пустым, а именно как только AcceptTC включено в Use.

```
---------------------------------------------------------------------------------------- *)
process TCAcceptance[t] : noexit :=
choice AcceptTC : TIds
          [ ]
[AcceptTC ne { } ]
—>
i ;
t?ta : TAddress  ?tcei : TCEI  ?tsp : TSP  [IsTCON19tsp)  implies
(TId(ta, tcei) IsIn AcceptTC)] ;
          TCAcceptance[t]
endproc (* TCAcceptance *)
(* ----------------------------------------------------------------------------------------
```

15 УПРАВЛЕНИЕ ПОТОКОМ ПРИ ПОМОЩИ ОБРАТНОЙ СВЯЗИ

Допускается любой недетерминизм поставщика услуг в отношении приема запросов на передачу данных, но одна зависимость между ограничениями управления потоком для нормальных и срочных данных задана. Эта зависимость вытекает из утверждения 9.2 ГОСТ 34.960, что «нормальные данные... нельзя добавлять в очередь, если их добавление может помешать добавлению срочных ПБДТ...».

Для поддержки абстрактного представления этого требования, и именно только в терминах взаимодействий услуг, вводится параметр MustAcceptTEX, который хранит трассу оконечных пунктов ТС, в которых последним выполненным запросом был запрос Т-ДАННЫЕ. В любой момент времени поставщик может произвольно выбрать конечные множества AcceptTDT и AcceptTEX, представляющие два множества оконечных пунктов ТС, где поставщик может принять нормальные и срочные СБДТ соответственно.

Вышеупомянутая зависимость представлена требованием, что AcceptTEX должен включать MustAcceptTEX. Никакие другие зависимости, которые могут существовать в реализациях услуг транспортного уровня, не описаны.

Примечание — В любой момент времени при динамическом произвольном выборе множества AcceptTDT множество оконечных пунктов, где запросы Т-ДАННЫЕ могут быть действительно приняты, входит в пересечение множеств

AcceptTDT и Use; это точное подмножество AccеptTDT, представляющее оконечные пункты, находящиеся в фазе передачи данных (см. 11.3.1). Аналогичное замечание применимо и при обратной связи срочных данных.

```
-------------------------------------------------------------------------------*)
process TBackpressure [t] : noexit :=
TBackp [t] ({ } of TIds) ||| RunButTDTreqTEXreqTDIS [t]
endproc (* TBackpressure *)
process TBackp[t] (MustAcceptTEX : TIds) : noexit :=
choice AcceptTDT, AcceptTEX : TIds
        [ ]
[MustAcceptTEX IsSubsetOf AcceptTEX]
—>
i ;
        (t?ta : TAddress ?tcei : TCEI ?tdr : TSP [IsTDTreq(tdr) and
(TId(ta, tcei) IsIn AcceptTDT)] ;
        TBackp [t] (Insert(TId(ta, tcei), MustAcceptTEX))
        [ ]
t?ta : TAddress ?tcei : TCEI ?ter : TSP [IsTEXreq(ter) and (TId(ta,
tcei) IsIn AcceptTEX)] ;
        TBackp [t] (Remove(TID(ta, Tcei), MustAcceptTEX))
        [ ]
t?ta : TAddress ?tcei : TCEI ?td : TSP [IsTDIS(td)] ;
        TBackp [t] (Remove(TId(ta, Tcei), MustAcceptTEX)))
endproc (*TBackp *)
process RunButTDTreqTEXreqTDIS[t] : noexit :=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP [not(IsTDTreq(tsp) or
IsTEXreq(tsp) or IsTDIS(tsp))] ;
        RunButTDTreqTEXreqTDIS [t]
endproc (* RunButTDTreqTEXreqTDIS *)
(* -------------------------------------------------------------------------
```

16 ПЕРЕДАЧА В РЕЖИМЕ-БЕЗ-УСТАНОВЛЕНИЯ-СОЕДИНЕНИЯ

16.1 Определения процесса

TSConnectionless основан на индикациях, которые могут выполняться в ответ на ранее выданные запросы. Для описания истории индикаций используется множество.

```
-------------------------------------------------------------------------------
                                                                        *)
process TSConnectionless[t] : noexit :=
Connectionless [t] (NoTCIReq)
endproc (* TSConnectionless *)
process Connectionless[t] (rh : TCIReqHistory) : noexit :=
```

```
t?ta : TAddress ?tsp : TSP [IsTUDTreq(tsp) and
(Lenght(UserData(tsp))
le MaxTUDTLength)] ; Connectionless [t] (Insert(tsp, rh))
          [ ]
[rh ne NoTClReq]
—>
(choice tspr, tspi : TSP
          [ ]
  [tspr IsIn rh and (tspi IsIndicationOf tspr)]
—>
t?ta : TAddress !tspi [ta IsCallingOf tspi] ;
          (Connectionless [t] (Remove(tspr, rh))
[ ]
i ; Connectionless [t] (rh))
          [ ]
i ; Connectionless [t] (Remove(tspr, rh))
endproc (* Connectionless *)
(* ----------------------------------------------------------------------
   16.2 Определения данных
---------------------------------------------------------------------- *)
type BasicTransportServiceConnectionlessRecHistory
is Set actualizedby TransportServicePrimitive using
sortnames
TSP for Element
Bool for FBool
TClReqHistory for Set
opnnames
NoTClReq for { }
endtype (* BasicTransportServiceConnectionlessRecHistory *)
type MaxTUDTLength
is DecNatRepr
opns
MaxTUDTLength          :          —> nat
eqns
ofsort Nat
MaxTUDTLength=NatNum(6+(3+(4+(8+Dec(8)))));
endtype (* MaxTUDTLength *)
endspec (* TransportService *)
```

Библиографические данные

УДК 681.324:006.354 П85

Ключевые слова: передача данных, обмен информацией между системами, формализованное описание, услуги транспортного уровня, язык LOTOS, эталонная модель, взаимосвязь открытых систем, типы данных, процессы

ОКСТУ 4002

Редактор *В. М. Лысенкина*
Технический редактор *О. Н. Никитина*
Корректор *В. И. Кануркина*

Сдано в наб. 11.02.94 Подп. в печ 29.03.94. Усл. п. л. 3,26. Усл. кр.-отт. 3,26.
Уч.-изд. л. 3,07 Тир. 374 экз. С 1136.

gost.e-ecolog.ru

Ордена «Знак Почета» Издательство стандартов, 107076, Москва, Колодезный пер., 14
Калужская типография стандартов, ул. Московская, 256. Зак. 363