Internet Engineering Task Force (IETF)
Request for Comments: 5812
Category: Standards Track
ISSN: 2070-1721

J. Halpern
Self
J. Hadi Salim
Znyx Networks

March 2010

Модель элемента пересылки ForCES

Forwarding and Control Element Separation (ForCES) Forwarding Element Model

Аннотация

Этот документ определяет модель элемента пересылки (FE[1]), используемую протоколом ForCES[2]. Модель представляет возможности, состояние и конфигурацию элементов пересылки в контексте протокола ForCES так, что элементы управления CE[3] могут должным образом управлять элементами FE. Более конкретно, модель описывает логические функции, присутствующие в FE, поддерживаемые этими функциями возможности и способы взаимодействия между функциями. Эта модель FE предназначена для выполнения требований, заданных в RFC 3654.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[4] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[5]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 5741.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке http://www.rfc-editor.org/info/rfc5812.

Оглавление

[1]Forwarding element.
[2]Forwarding and Control Element Separation - разделение элементов управления и пересылки.
[3]Control element.
[4]Internet Engineering Task Force.
[5]Internet Engineering Steering Group.

1. Введение

В RFC 3746 [RFC3746] задана схема, с помощью которой элементы управления CE могут настраивать и управлять одним или множеством раздельных элементов пересылки FE внутри элемента сети NE[1] с помощью протокола ForCES. Архитектура ForCES позволяет элементам пересылки с различной функциональностью принимать участие в работе сетевого элемента ForCES. Вследствие различной функциональности элементы CE могут принимать лишь минимальные допущения о функциях FE в NE. До того, как CE смогут настраивать и контролировать поведение элементов пересылки FE, и требуется запросить и определить возможности и состояния своих FE. В RFC 3654 [RFC3654] указано, что эти характеристики, состояния и конфигурационные данные выражаются в форме модели FE.

В RFC 3444 [RFC3444] показано, что информационные модели (IM[2]) и модели данных (DM[3]) различаются целями. «Основной целью IM является моделирование управляемых объектов на концептуальном уровне независимо от конкретной реализации или используемого протокола». «DM, напротив, определяются на нижнем уровне абстракции и включают множество деталей. Они предназначены для разработчиков и включают зависимые от протокола конструкции». Иногда сложно провести чёткую границу между этими моделями. Описанная в этом документе модель FE является прежде всего информационной моделью, но она также включает некоторые аспекты модели данных типа явных определений схемы класса логических функциональных блоков LFB[4] и схемы FE. Предполагается, что эта модель FE будет служить для определения содержимого (payload) информационного обмена между CE и FE в протоколе ForCES.

1.1. Требования к модели FE

В RFC 3654 [RFC3654] приведены требования к модели ForCES FE, которые должна определять перечисленное ниже.

- Логически разделяемые и отчётливые операции пересылки пакетов в пути данных FE (логические функциональные блоки или LFB).
- Возможные топологические связи (и последовательность операций пересылки пакетов) между разными LFB.
- Операционные возможности (например, пределы ёмкости, ограничения, операционные свойства, детализация настройки) каждого типа LFB.
- Возможные конфигурационные параметры (например, компоненты) каждого типа LFB.
- Метаданные, которые могут передаваться между LFB.

1.2. Модель FE и реализации FE

Предлагаемая здесь модель FE основана на абстракции, использующей различные функциональные блоки LFB, которые соединены в направленный граф и получают, обрабатывают, изменяют и передают пакеты вместе с метаданными. Модель FE разработана и все определяемые классы LFB следует организовывать так, чтобы различные реализации пути пересылки данных можно было логически отобразить на модель с корректной функциональностью и последовательностью операций. Однако модель не предназначена для непосредственного решения вопроса отображения конкретной реализации на топологию LFB. Производители уровня пересылки самостоятельно определяют представление функциональности FE с использованием модели FE. Нашей целью является разработка модели FE, обеспечивающей достаточную гибкость для большинства реализаций общего назначения.

Модель топологии LFB для конкретной реализации пути данных должна корректно фиксировать последовательность действий с пакетом. Генерация метаданных теми или иными LFB всегда **должна** предшествовать использованию этих метаданных последующими LFB в графе топологии, это требуется для логической согласованности операций. Кроме того, изменение полей пакета, которые потом будут служить входными данными для последующей обработки, **должны** выполняться в заданном моделью порядке для этой конкретной реализации, чтобы обеспечить корректность работы.

1.3. Модель FE и протокол ForCES

Базовый протокол ForCES [RFC5810] используется элементами CE и FE для поддержки коммуникационного канала между собой. Протокол ForCES может служить для запроса и раскрытия внутренней топологии FE. Детали конкретной реализации пути данных внутри FE, включая топологию LFB, а также возможности и атрибуты каждого LFB, передаются CE в информационных элементах протокола ForCES. Модели класса LFB следует определять всю информацию, требуемую для обмена между FE и CE с целью подходящей настройки и управления данным LFB.

[1]Network element.
[2]Information model.
[3]Data model.
[4]Logical Functional Block.

Спецификация разных данных (payload) сообщений ForCES систематизированным путём затруднена без формального определения объектов для настройки и управления (FE и LFB в нем). Документ для модели FE определяет набор классов и компонент для описания и манипуляций с состояниями LFB внутри элемента FE. Эти определения классов сами по себе обычно не появляются в протоколе ForCES. Вместо этого операции протокола ForCES ссылаются на определённые в этой модели классы, которые включают нужные компоненты и определяют операции.

В разделе 7 приведено более подробное рассмотрение способов использования модели FE протоколом ForCES.

1.4. Язык моделирования FE

Хотя это и не обязательно, будет полезно использование формального языка моделирования для представления концептуальной модели FE, описываемой в этом документе. Использование формального языка позволит обеспечить согласованность и логическую совместимость разных LFB. Полная спецификация будет написана с использованием такого языка моделирования данных. Формальное определение классов LFB может облегчить возможную автоматизацию того или иного процесса генерации кода и функциональную проверку пригодности произвольных топологий LFB. Это определения классов формируют библиотеку LFB. Документы, описывающие классы LFB называются документами библиотеки LFB.

Читаемость для человека была одним из важнейших факторов при выборе языка спецификации, а кодирование, декодирование и эффективность передачи не принимались во внимание при этом выборе. Метод кодирования для передачи в среде не зависит от выбранного языка спецификации, выходит за рамки этого документа и определяется протоколом ForCES.

Для этого документа в качестве языка спецификации был выбран XML, поскольку этот язык понятен человеку и удобен для машинного анализа, поддерживаемого множеством доступных инструментов. В этом документе применяется схема XML для определения структуры документов библиотеки LFB в соответствии с определениями [RFC3470], [Schema1] и [Schema2]. Хотя эти определения классов LFB не передаются в протоколе ForCES, они соответствуют рекомендациям RFC 3470 [RFC3470] по использованию XML в протоколах IETF.

За счёт применения схемы XML для определения структуры документов библиотеки LFB обеспечен чёткий набор желательных семантических описаний и синтаксических ограничений в этом документе. В результате этого при изменении синтаксиса в будущем потребуется определить новую схему. Это будет указано другим идентификатором URN, указывающим пространство имён для новой схемы.

1.5. Структура документа

В разделе 3 приведён концептуальный обзор модели FE, являющийся основой для более подробного рассмотрения и спецификации в последующих разделах. Разделы 4 и 5 описывают ядро модели FE, с подробным рассмотрением двух основных аспектов - общей модели LFB и определения FE Object LFB с компонентами, включая возможности FE и топологическую информацию LFB. Раздел 6 посвящён требованиям к модели в соответствии с приведёнными в RFC 3654 [RFC3654] требованиями к протоколу ForCES, а в разделе 7 разъясняется как модель FE следует применять в протоколе ForCES.

2. Определения

Уровни требований (**должно**, **следует**, **может**, **недопустимо**) указываются в соответствии с RFC 2119 [RFC2119]. Эти уровни применяются при описании требуемого поведения элементов пересылки или управления ForCES для поддержки или манипуляций с описываемой в этой модели информацией.

Термины, связанные с требованиями к ForCES, определены в RFC 3654 [RFC3654] и не дублируются здесь. Набор терминов, относящихся к модели FE, приведён ниже.

FE Model - модель элемента пересылки
: Модель, описывающая функции логической обработки в FE. Модель FE, предлагаемая в документе, включает три компоненты - модель отдельного логического функционального блока (модель LFB[1]), логическое соединение между LFB (топология LFB) и атрибуты уровня FE, включая возможности FE. Модель FE служит основой для определения информационных элементов, передаваемых между CE и FE в протоколе ForCES [RFC5810].

Data Path - путь (передачи) данных
: Концептуальный путь, по которому пакеты проходят через уровень пересылки в FE. Отметим, что в FE может быть более одного пути данных.

LFB (Logical Functional Block) Class (или type) — класс (или тип) LFB
: Шаблон, представляющий небольшой, логически выделяемый аспект обработки FE. Большинство LFB относится к обработке пакетов в пути данных. Классы LFB являются базовыми элементами модели FE.

LFB Instance - экземпляр LFB
: При перемещении потоков по пути данных FE эти потоки проходя через один или множество LFB, каждый из которых является экземпляров конкретного класса LFB. В пути данных FE может присутствовать множество экземпляров одного класса LFB. Отметим, что в документе термин LFB часто упоминается безотносительно класса или экземпляра, если контекст позволяет различить их.

LFB Model - модель LFB
: Модель LFB описывает содержимое и структуры LFB, а также даёт определения связанных с блоком данных. Для формальных определений требуемых при моделировании структур используется язык XML. В модели LFB определены четыре типа информации. Ядром модели являются определения классов LFB, а три оставшихся типа определяют конструкции, связанные с определениями классов или используемые в них, - это типы данных многократного использования, поддерживаемые форматы кадров (пакетов) и метаданные.

Element - элемент
: Термин «элемент» в документе применяется в соответствии с практикой XML и указывает на помеченную тегом XML часть документа XML. Точное определение можно найти в спецификациях XML от W3C. Термин включён в этот список для полноты, поскольку формальная модель ForCES использует XML.

[1]Logical Function Block.

Attribute - атрибут

Атрибуты используются в формальной модели ForCES в соответствии с принятой в XML практикой, т. е. для обозначения свойств, включённых в тег XML.

LFB Meta Data - метаданные LFB

Метаданные служат для передачи информации о состоянии на уровне отдельного пакета из одного блока LFB в другой, но без передачи через сеть. Модель FE определяет для таких данных идентификацию, обработку, и восприятие (потребление) другими LFB, но не на реализацию состояния отдельных пакетов в реальном оборудовании. Метаданные передаются между FE и CE в перенаправляемых пакетах.

ForCES Component - компонента ForCES

Компонентой ForCES называют чётко определённый, однозначно идентифицируемый и адресуемый блок модели ForCES. Компонента имеет 32-битовый идентификатор, имя, тип и необязательное описание, которые зачастую называют просто компонентами.

LFB Component - компонента LFB

Компонентами LFB являются компоненты ForCES, определяющие рабочие параметры LFB, которые должны быть видны CE.

Structure Component - компонента структуры

Компонента ForCES, являющаяся частью комплексной структуры данных, которая будет использоваться в определениях данных LFB. Отдельные части, составляющие структурированный набор данных, называют компонентами структуры. Они могут быть любыми допустимыми типами данных, включая таблицы и структуры.

Property - свойство

Компоненты ForCES имеют связанные с ними свойства типа читабельности. Другим примером является длина компонент переменного размера. Эти свойства доступны CE для чтения, а в некоторых случаях и для записи. Подробное описание свойств ForCES приведено в параграфе 4.8.

LFB Topology - топология LFB

Представление логических связей между экземплярами LFB и их размещения в пути данных внутри одного элемента FE. Иногда используется термин «внутренняя топология FE», но не следует путать её с топологией соединений между FE (inter-FE topology). Топология LFB не входит в модель LFB, но является частью модели FE.

FE Topology - топология FE

Представление соединений между множеством FE в одном NE. Иногда это называют внешней топологией FE, чтобы отличать от внутренней топологии FE (т. е. топологии LFB). Отдельный элемент FE может не иметь информации о всей топологии FE, но локальное представление его соединений с другими FE считается частью модели FE. Топология FE раскрывается базовым протоколом ForCES или иным способом.

Inter-FE Topology - внешняя топология FE

См. FE Topology.

Intra-FE Topology - внутренняя топология FE

См. LFB Topology.

LFB Class Library - библиотека классов LFB

Библиотека классов LFB представляет собой набор классов LFB, которые отмечены как наиболее общие функции в большинстве FE, и поэтому рабочей группе ForCES следовало определить их в первую очередь.

3. Концепции модели ForCES

В этом разделе вводятся некоторые важные концепции ForCES, которые используются на протяжении всего документа. К ним относятся абстракции возможностей и состояния, конструкция моделей FE и LFB, а также однозначная адресация различных структур модели. Детали этих аспектов рассмотрены в разделах 4 и 5. Назначение данного раздела состоит в обсуждении этих концепций на верхнем уровне и создание основы для подробного рассмотрения в следующих разделах.

Модель ForCES FE включает абстракции возможностей и состояния.

- Модель возможностей FE/LFB описывает «способности и ёмкость» FE/LFB путём задания вариаций, поддерживаемых функциями, и всех ограничений. «Ёмкость» описывает пределы конкретных компонент (например, предельный размер таблиц).

- Модель состояния описывает текущее состояние FE/LFB, т. е. мгновенные значения или рабочее поведение FE/LFB.

В параграфе 3.1 рассмотрены различия между моделями возможностей и состояния, а также описано их объединение в модель FE.

Конструкция модели ForCES описанная в этом документе, позволяет элементу FE предоставить информацию о своей структуре для работы. Её можно рассматривать как информацию уровня FE и информацию об отдельных экземплярах LFB, обеспечиваемых FE.

- Модель ForCES включает конструкции для определения классов логических функциональных блоков (LFB), которые FE может поддерживать. Классы определены в этом и других документах. Определение такого класса обеспечивает информационное содержимое экземпляров класса LFB для мониторинга и управления в целях ForCES. Каждый класс модели LFB формально определяет операционные компоненты LFB, возможности и события LFB. В параграфе 3.2 представлена концепция LFB как базовых функциональных блоков для построения модели ForCES.

- Модель FE также предоставляет конструкции, требуемые для мониторинга и управления элементом FE в целом для протокола ForCES. Для согласованности операций и простоты эта информация представляется как LFB - класс FE Object LFB и одиночный экземпляр LFB данного класса, определённые с использованием модели LFB. Класс FE Object определяет компоненты для предоставления информации на уровне FE, в частности, возможности FE на грубом уровне, без их полного перечисления и детализации. Частью информации уровня FE является топология LFB, которая показывает логические соединения между экземплярами LFB на пути данных внутри элемента FE. Эта топология рассматривается в параграфе 3.3. FE Object также включает информацию о классах LFB, которые FE может поддерживать.

Модель ForCES позволяет однозначно указывать различные конструкции, которые она определяет. Это включает идентификацию классов LFB и экземпляров LFB внутри класса, а также идентификацию компонент экземпляра.

Протокол ForCES [RFC5810] инкапсулирует целевые адреса, чтобы получать доступ к объектам, указываемым CE. Иерархия адресации показана ниже.

- Элементы FE однозначно указываются 32-битовыми идентификаторами FEID.
- Каждый класс LFB имеет уникальный 32-битовый идентификатор LFB Class, который является глобальным для элемента сети и может быть выделенным IANA значением.
- Внутри FE может присутствовать множество экземпляров каждого класса LFB. Эти классы указываются 32-битовыми идентификаторами, уникальными в рамках отдельного класса LFB в данном FE.
- Для всех компонент экземпляра LFB используются свои 32-битовые идентификаторы.

Адресация более подробно рассматривается в параграфе 3.3.

3.1. Модели возможностей и состояний ForCES

Моделирование возможностей и состояний применимо к абстракциям FE и LFB.

На рисунке 1 показаны возможности, состояние и конфигурация FE для коммуникаций CE-FE по протоколу ForCES.

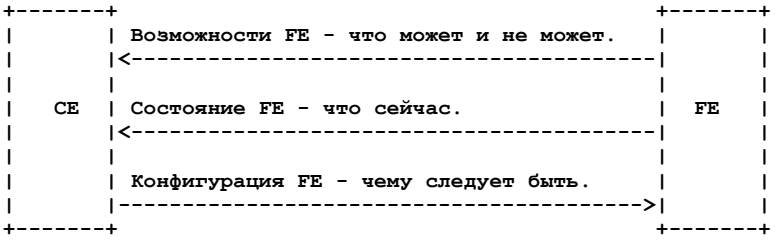


Рисунок 1. Обмен возможностями, состоянием и конфигурацией FE в контексте коммуникаций CE-FE по протоколу ForCES.

3.1.1. Модели возможностей и состояний FE

Концептуально модель возможностей FE говорит элементу CE какие состояния разрешены в FE с информацией о ёмкости, включающей некоторые количественные пределы или ограничения. За счёт этого CE имеет общие сведения о конфигурации, которая применима для конкретного FE.

3.1.1.1. Модель возможностей FE

Модель возможностей FE может быть использована для описания FE без детализации. Например, FE можно определить следующим образом:

- FE может обслуживать пересылку IPv4 и IPv6;
- FE может выполнять классификацию на основе IP-адресов отправителя и получателя, портов отправителя и получателя и т.п.;
- FE может выполнять измерения;
- FE может обслуживать до N очередей (ёмкость);
- FE может добавлять и удалять заголовки инкапсуляции, включая IPsec, GRE, L2TP.

Хотя можно было попытаться построить объектную модель для полного представления возможностей FE, требуемые для реализации такого подхода усилия оказались слишком велики. Основная сложность связана с описанием подробных ограничений типа максимального числа классификаторов, очередей, буферных пулов и измерителей, которые FE может обеспечивать. Мы надеемся, что баланс между простотой и гибкостью может быть достигнут для модели FE путём объединения достаточно грубого описания возможностей с механизмом информирования об ошибках. Т. е. при попытке CE задать для элемента FE поведение, которое тот не способен реализовать, FE будет возвращать ошибку, указывающую проблему. Примерами такого подхода служат Diffserv PIB [RFC3317] и схема PIB [RFC3318].

3.1.1.2. Модель состояний FE

Модель состояний FE представляет мгновенное состояние FE для элемента CE. Например, с помощью модели состояния FE можно описать элемент FE соответствующему CE следующим образом:

- на данном порту пакеты классифицируются с использованием данного фильтра;
- данный классификатор приводит к «измерению» и маркировке пакетов определёнными способами;
- пакеты от конкретных маркировщиков доставляются в общую очередь для обработки, тогда как другие пакеты доставляются в иную очередь;
- конкретный планировщик с определенным поведением и параметрами будет обслуживать собранные очереди.

3.1.1.3. Модели возможностей и состояний LFB

Информация о возможностях и состоянии LFB формально определяется с использованием схемы XML.

Информация о возможностях на уровне LFB является неотъемлемой частью модели LFB и обеспечивает развитую семантику. Например, когда некоторые функции класса LFB являются необязательными, CE требуется возможность определить какие из необязательных функций поддерживаются данным экземпляром LFB. Схема для определения классов LFB обеспечивает способы идентификации таких компонент.

Информация о состоянии формально определяется с использованием компонент LFB.

3.1.2. Связи между моделями возможностей и состояний LFB и FE

Информация о возможностях FE описывает классы LFB, которые FE может создавать, число экземпляров каждого класса, которые могут быть созданы, топологические ограничения (связность) для экземпляров LFB и т. п. В разделе 5 определены компоненты уровня FE, включающие информацию о возможностях. Поскольку вся информация представляется в виде LFB, это обеспечивается одним экземпляром класса FE Object LFB. Благодаря использованию известного класса LFB с известным идентификатором экземпляра, протокол ForCES обеспечивает элементам CE доступ к этой информации при возникновении потребности в ней в любой момент, включая организацию соединения между CE и FE.

После того, как возможности FE описаны элементу CE, информация о состоянии FE может быть представлена на двух уровнях. Первый уровень представляют разные логически разделяемые функции, которые называют логическими функциональными блоками или LFB. Второй уровень информации описывает порядок и расположение отдельных LFB в пути данных для обеспечения работы уровня пересылки. Соединения и порядок блоков LFB называются топологией LFB. В параграфе 3.2 рассмотрены связанные с LFB концепции верхнего уровня, а параграф 3.3 посвящён топологии LFB. Эта топологическая информация представляется компонентами экземпляра FE Object LFB, чтобы позволить CE извлекать информацию и манипулировать ею.

3.2. Моделирование логического функционального блока (LFB)

Каждый блок LFB выполняет чётко определённое действие или расчёт для проходящих через него пакетов. По завершении предписанных действий пакет либо изменяется тем или иным способом (например, декапсуляция или маркировка), либо результаты обработки генерируются сохраняются, зачастую в форме метаданных (например, классификация). Каждый LFB обычно выполняет одно действие. Примерами таких LFB являются классификаторы, формирователи и измерители. Моделирование LFB с такой тонкой детализацией позволяет использовать небольшое число LFB для точного выражения функций FE более высокого порядка (например, пересылка IPv4), которые в свою очередь описывают более сложные сетевые функции и фирменные реализации программ и оборудования. Эти LFB с тонкой детализацией будут определены в одном или множестве отдельных документов, использующих материалы данной модели.

Имеются также случаи, когда LFB могут применяться для обеспечения набора компонент управления работой FE со стороны CE (т. е., предназначены для управления), без связывания этого управления с конкретными пакетами или частями пути данных. Примером такого LFB является FE Object, который обеспечивает CE информацией о FE в целом, и позволяет CE[1] управлять некоторыми аспектами FE типа самого пути данных. Такие LFB не имеют ориентированных на пакеты свойств, описываемых в этом параграфе.

В общем случае FE включает множество LFB, как показано на рисунке 2, и все этил LFB используют общую точку завершения (интерфейс) протокола ForCES (Fp), которая реализует логику протокола ForCES и обеспечивает коммуникационный канал с элементом CE.

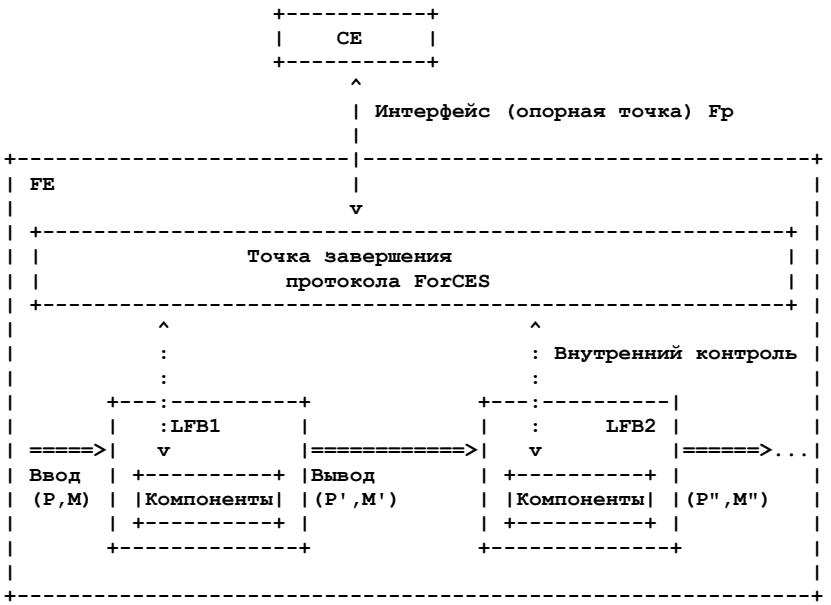


Рисунок 2. Базовая диаграмма LFB.

LFB, как показано на рисунке 2, может иметь входы, выходы и компоненты, которые могут запрашиваться и управляться CE через интерфейс Fp (определён в RFC 3746 [RFC3746]), и точку завершения протокола ForCES. Горизонтальные стрелки на рисунке показывает уровень пересылки для соединения входов и выходов LFB внутри одного FE. P (с маркировкой изменений) показывает пакет данных, а M (с маркерами изменений) показывает связанные с пакетом метаданные. Вертикальная линия между CE и FE указывает опорную точку Fp где происходит двухсторонний обмен между CE и FE - от CE к FE передаётся конфигурация, управления и вставка пакетов, а от FE к CE происходит перенаправление пакетов на уровень управления, передаются данные мониторинга и учёта, сообщения об ошибках и т. п. Отметим, что взаимодействие между CE и LFB является лишь абстрактным и опосредованным. Результатом такого взаимодействия являются манипуляции CE с компонентами экземпляров LFB.

LFB может иметь один или множество входов. Каждый вход принимает пары из пакета и связанных с ним метаданных. В зависимости от определения входного порта LFB пакет или метаданные могут быть пустыми (не представляется на вход). Когда данные приходят на вход LFB, пакет или метаданные должны быть не пусты (иначе это просто отсутствие

[1]В оригинале ошибочно сказано FE. *Прим. перев.*

ввода). Операции LFB в общем случае могут инициироваться поступлением данных на вход, таймером или другим состоянием системы. Когда целью является ввод данных, входные данные не должны быть пустыми.

LFB обрабатывает ввод и даёт один или множество выводов в форме пар из пакета и связанных с ним метаданных. В зависимости от определения порта LFB пакет или метаданные могут быть пусты (отсутствовать). Присоединённые к пакеты на выходе метаданные могут быть полученными раньше или содержать информацию об обработке пакета , которая может применяться последующими LFB при обработке пакета в FE.

Пространство имён служит для связывания уникального имени и идентификатора с каждым классом LFB. Пространство **должно** быть расширяемым для обеспечения возможности определять новые LFB по мере развития уровня пересылки.

Операция LFB задаётся в модели для того, что CE мог понимать поведение пересылки в пути данных. Например, CE нужно понимать, в какой точке пути данных FE будет декрементироваться поле TTL заголовка IPv4. Т. е. CE нужно знать может ли пакет быть доставлен ему до или после данной точки в пути данных. Кроме того, CE нужно понимать где и какие изменения заголовков (например, добавление или исключение заголовка туннеля) могут выполняться элементами FE. CE проверяет совместимость между собой разных LFB на пути данных FE для предотвращения присутствия несовместимых экземпляров LFB, которые нарушать работу пути данных. Поэтому модель разработана с учётом предоставления CE требуемой информации.

Выбор уровня детализации для описания функций LFB является важным аспектом этой модели. Для производителей важна возможность выразить операции классов LFB с уровнем детализации, который позволит объединить физические устройства, реализующие разные функции LFB в один элемент FE. Однако модель и связанная с ней библиотека LFB не должны быть детализированы так, что это будет ограничивать реализации. Поэтому нужна полуформальная спецификация - текстовое описание работы LFB (для человека) с достаточной конкретизацией и однозначностью, который сделают возможными проверку соответствия и эффективное проектирование, чтобы обеспечить взаимодействие между разными CE и FE.

Модель класса LFB среди прочей информации задаёт:

- число входов и выходов (а также возможность его настройки);
- метаданные, считываемые и потребляемые на входах;
- метаданные, создаваемые на выходах;
- типы пакетов, приемлемые на входах и передаваемые на выходах;
- изменения содержимого пакетов (включая инкапсуляцию и декапсуляцию);
- критерии маршрутизации пакетов (при наличии в LFB множества выходов);
- временные изменения пакетов;
- изменение порядка в потоке пакетов
- компоненты информации о возможностях LFB;
- события, которые могут детектироваться LFB с передачей уведомлений CE;
- рабочие компоненты LFB.

В разделе 4 приведено подробное обсуждение модели LFB с формальной спецификацией схемы класса LFB. В оставшейся части этого параграфа содержится концептуальный обзор некоторых важных аспектов моделирования LFB без рассмотрения конкретных деталей.

3.2.1. Выход LFB

Выход LFB представляет собой концептуальный порт, через который LFB может передавать информацию другому LFB. Передаваемая в этот пор информация представляет собой пары из пакета и связанных с ним метаданных, при этом одна из компонент пары может быть пустой (если пусты обе, это просто говорит об отсутствии вывода).

Один выход LFB может быть подключён только к одному входу LFB. Это требуется для обеспечения однозначности пути потока через топологию LFB.

Некоторые LFB будут иметь один вывод, как показано на рисунке 3.a.

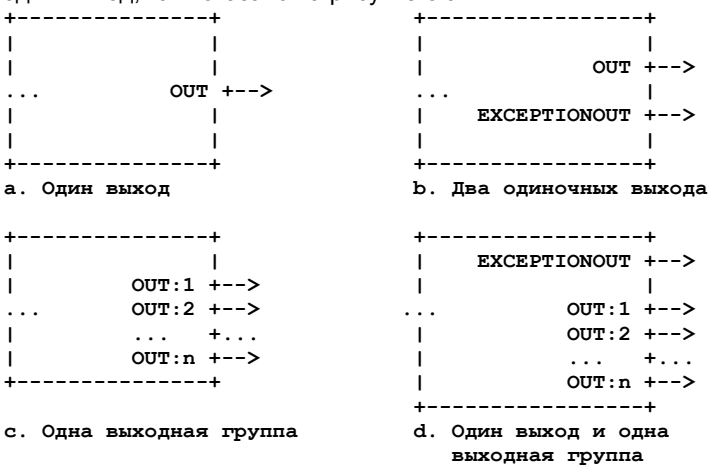


Рисунок 3. Примеры LFB с разными выходными комбинациями.

Для создания нетривиальной топологии LFB требуются блоки LFB со множеством выходов, которые позволят классу LFB организовать ветвление в пути данных. Для ветвления предоставляются два механизма - множество одиночных выходов и выходные группы, которые могут сосуществовать в одном классе LFB.

Множество одиночных выходов определяется в классе LFB для моделирования предопределённого числа семантически различающихся выходов. Т. е. определение класса LFB **должно** включать число выходов и это предполагает, что информация о количестве выходов доступна в момент определения класса LFB. При создании экземпляра LFB или использовании этого экземпляра добавление выходов невозможно.

Например IPv4 LPM[1] LFB может иметь один выход (OUT) для отправки пакетов после успешного поиска LPM с передачей одновременно метаданных META_ROUTEID, а другой (EXCEPTIONOUT) для исключительных ситуаций, когда поиск LPM завершился отказом. Этот пример показан на рисунке 3.b. Пакеты на этих выходах не только требуют разных трактовок нисходящего направления, но и приводят к двум разным условиям в LFB и каждый выход передаёт свои метаданные. Эта концепция предполагает, что число выходов известно в момент определения LFB. Для каждого одиночного выходя определение класса LFB указывает типы передаваемых кадров (пакетов) и метаданные.

Выходная группа, с другой стороны, используется для моделирования случаев когда поток похожих пакетов с идентичным набором разрешённых метаданных расщепляется на несколько путей. В таком случае число путей не известно в момент определения класса LFB, поскольку оно не является неотъемлемым свойством класса. Выходная группа состоит из множества выходов, которые используют общие определения для передачи кадров (пакетов) и метаданных (см. рисунок 3.c). Каждый экземпляр выхода может быть соединён со своим нисходящим LFB, как это делается для одиночных выходов, но число экземпляров выходов может быть разным у разных экземпляров одного класса LFB. Определение класса может включать нижний и/или верхний предел числа выходов. Кроме того, для настраиваемых FE информация о возможностях FE может задавать свои пределы для числа экземпляров в конкретных выходных группах для определённых LFB. Реальное число выходов в группе является компонентой экземпляра LFB, которая доступна только для чтения в статической топологии и может быть изменена в динамической. Экземпляры выходов в группе нумеруются с 0 до N-1 и доступны по номерам в рамках LFB. Для использования групп выходных портов LFB имеет встроенный механизм выбора конкретного экземпляра выхода для каждого пакета. Этот механизм описывается в тестовом определении класса и обычно может быть настроен с помощью тех или иных атрибутов LFB.

В качестве примера рассмотрим LFB редиректора, единственной задачей которого является направлять пакеты в один из N нисходящих путей на основе метаданных, связанных с каждым прибывающим пакетом. Такой LFB достаточно универсален и может использоваться в разных точках топологии. Например, рассмотрим LFB, которые записывают типа пакета в метаданные FRAMETYPE или класс скорости для пакета в COLOR и эти метаданные могут использоваться для пересылки. Редиректор может служить для разделения путей IPv4 и IPv6 на основе FRAMETYPE (N=2) или для разделения путей по скорости на основе метаданных COLOR (red, yellow, green; N=3) и т. п..

Использование выходной группы в рассмотренном выше классе LFB обеспечивает желаемую гибкость для приспособления каждого экземпляра данного класса к выполнению нужных задач. Метаданные, которые будут служить селектором выходного экземпляра, являются свойством LFB. Для каждого пакета значение указанных метаданных может использоваться в качестве индекса выходного интерфейса. Дополнительно LFB может иметь настраиваемую таблицу селекторов для отображения метаданных на выходные экземпляры.

Отметим, что концепцию выходных групп для адаптивного ветвления могут применять и другие LFB. Например, LFB классификатора с одним входом и N выходов легко определить с помощью концепции выходной группы. Такой же результат может быть обеспечен с помощью LFB классификатора с одиночным выходом и LFB редиректора с явно заданными N выходами. Выбор решения об использовании выходной группы остаётся за разработчиками определения класса LFB.

Модель позволяет комбинировать выходную группу с одиночными выходами в одном классе, как показано на рисунке 3.d. Здесь LFB имеет два типа выходов - OUT для обычного вывода пакетов и EXCEPTIONOUT для пакетов, с которыми связаны исключительные ситуации. Обычный выход OUT имеет множество экземпляров, т. е. является выходной группой.

Таким образом, класс LFB может определять один выход, множество одиночных выходов, одну или несколько выходных групп или комбинацию перечисленных вариантов. Множество одиночных выходов следует использовать в тех случаях, когда нужно ветвление пути данных и выполняется хотя бы одно из двух указанных ниже условий.

- Число нисходящих направлений наследуется из определения класса (фиксировано).
- Тип кадров и разрешённых для отправки метаданных любого из выходов отличается от аналогичных параметров других выходов (т. е. разные выходы не могут использоваться для однотипных пакетов и метаданных).

Выходная группа подходит в тех случаях, когда нужно ветвление и выполняется хотя бы одно из условий:

- число нисходящих направлений не было известно при определении класса LFB;
- тип кадров и набор метаданных для этих выходов достаточно похожи (в идеале одинаковы) и для них может применяться одно определение выхода.

3.2.2. Вход LFB

Вход LFB представляет собой концептуальный порт, через который данный LFB может принимать информацию от других LFB. Эта информация обычно состоит из пар «пакет-метаданные». Любая из компонент пары может отсутствовать, а отсутствие обеих означает просто отсутствие ввода.

Для экземпляров LFB, принимающих пакеты от множества других экземпляров LFB (fan-in), имеется три варианта моделирования fan-in, которые поддерживаются моделью и могут комбинироваться в одном LFB:

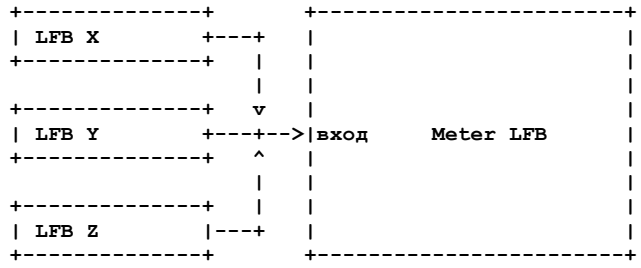
- неявное мультиплексирование через один вход;
- явное мультиплексирование через множество одиночных входов;

[1]Longest-Prefix-Matching - максимальный размер совпадения префиксов.

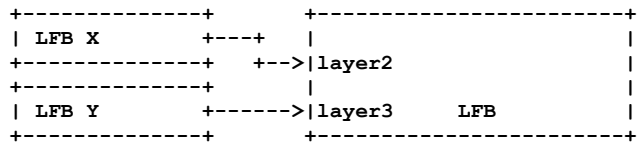
- явное мультиплексирование через входную группу.

Простейший вариант мультиплексирования использует одиночный вход (рисунок 4.a). Большинство LFB имеют лишь один одиночный вход. Мультиплексирование в такой порт возможно за счёт того, что модель позволяет подключать к одному входу множество выходов LFB. Это свойство применимо ко всем входам LFB без каких-либо особых требований к классу LFB. Мультиплексирование в один порт применимо, когда пакеты из восходящих LFB похожи по типам кадров и сопровождающим их метаданным и для них требуется похожая обработка. Отметим, что этот вариант не решает вопросов «конкуренции» между одновременно прибывающими пакетами. Если такая обработка требуется, следует воспользоваться одним из двух других вариантов мультиплексирования.

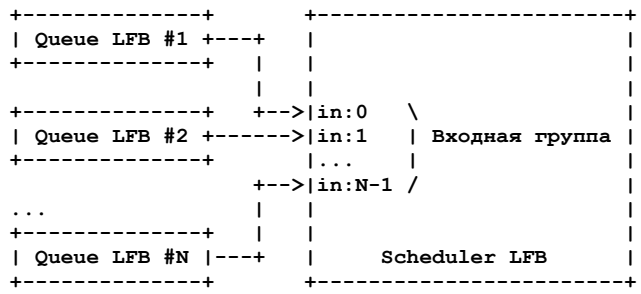
Во втором варианте используются отдельные одиночные порты (рисунок 4.b). Этот вариант подходит для ситуаций, когда LFB нужно обрабатывать разнотипные потоки пакетов, требующие определяемой входом обработки внутри LFB, и число разных входов известно в момент определения класса LFB. Например, LFB, способных выполнять декапсуляцию L2 (в L3) и инкапсуляцию L3 (в L2), может иметь два входа - один будет принимать кадры L2 для декапсуляции, а второй - L3 для инкапсуляции. Этот тип LFB ожидает разные кадры (L2 и L3) на своих входах, имеющие свои наборы метаданных, и будет применять к этим кадрам разную обработку. Эта модель может явно решать проблему «конкуренции» пакетов путём определения их обработки в классе LFB.



(a) LFB соединён с множеством восходящих LFB через один вход.



(b) LFB соединён с множеством восходящих LFB через два раздельных входа.



(c) LFB планировщика использует входную группу для разделения очередей, из которых приходят пакеты.

Рисунок 4. Примеры LFB с разными комбинациями входов.

Третья модель мультиплексирования использует концепцию входной группы. Это похоже на описанную выше концепцию выходной группы и показано на рисунке 4.c. Входная группа включает множество входов с общими свойствами (ожидание однотипных кадров и метаданных). Экземпляры входов нумеруются от 0 до N-1. Извне эти входы выглядят обычными, т. е. совместимый восходящий LFB может подключить свой выход к одному из этих входов. Когда пакет представляется в LFB через конкретный экземпляр входа, индекс этого входа известен LFB и может использоваться при внутренней обработке. Например, индекс входа может служить селектором таблицы или явным селектором предпочтений при возникновении «конкуренции» Как и для выходных групп, число экземпляров входов не определяется классом LFB. Однако определение класса может включать ограничения на диапазон возможных значений. Кроме того, если FE поддерживает настройку топологии, это может вносить дополнительные ограничение на число экземпляров входов для конкретной группы определённого класса LFB. С учётом этих ограничений разные экземпляры одного класса могут иметь разное число входов.

Реальное число экземпляров входов в группе определяется компонентой класса LFB, которая доступна только для чтения в статической топологии и может быть изменена в динамической.

В качестве примера входной группы рассмотрим Scheduler LFB на рисунке 4.c. LFB получает пакеты от множества Queue LFB через экземпляры входов и использует индекс входа для разрешения конфликтов и планирования.

Таким образом, класс LFB может определять один вход, множество одиночных входов, одну или множество входных групп, а также комбинации перечисленного. Любой вход позволяет неявно мультиплексировать потоки похожих пакетов при подключении множества выходов к одному входу. Явное мультиплексирование одиночных входов полезно в тех случаях, когда нужно явно предотвратить «конкуренцию» или класс LFB должен принимать и обрабатывать известное число потоков разнотипных пакетов. Входные группы подходят при необходимости явного предотвращения конфликтов, но число портов не наследуется от класса (и не было известно в момент определения класса) или когда для работы LFB критично знать через какой из входов был получен пакет.

3.2.3. Тип пакета

Когда классы LFB определены, **должны** быть заданы форматы входных и выходных пакетов (например, IPv4, IPv6, Ethernet). Это типы пакетов, которые вход данного LFB способен принимать и обрабатывать или выход LFB способен выдавать. Модель требует однозначно указывать разные типы пакетов именами и/или идентификаторами.

Отметим, что каждый LFB имеет набор пакетов, с которыми он работает, но не имеет значения, передаёт ли нижележащая (базовая) реализация большую часть пакета. Например, IPv4 LFB может работать только с пакетами IPv4, но нижележащая (базовая) реализация может вырезать или не вырезать заголовок L2 из пакета. Наличие или отсутствие такой обработки «непрозрачно» для CE.

3.2.4. Метаданные

Метаданные представляют собой состояние, которое передаётся от одного LFB к другому вместе с пакетом. Метаданные помогают последующим блокам LFB обрабатывать этот пакет.

Модель ForCES определяет метаданные как неделимые элементы в форме пар «имя-значение».

Модель ForCES предоставляет разработчикам классов LFB способы формального определения процедур создания, изменения, чтения и потребления (удаления) метаданных.

Метаданные Inter-FE, пересекающие границу FE, выходят за рамки этого документа, хотя семантически они похожи на обычные метаданные.

Неформальное описание метаданных приведено в разделе 4.

3.2.4.1. Жизненный цикл метаданных в модели ForCES

Каждый элемент метаданных представляется в виде пары <label, value>, где метка (label) указывает тип информации (например, color), а её значение (value) задаёт реальную информацию (например, red). Здесь метка представлена в текстовой форме, но для протокольной обработки она связывается с числовым идентификатором.

Для обеспечения взаимодействия между LFB спецификация класса LFB должна определять, какие данные класс LFB «читает» или «потребляет» на своих входах и какие метаданные он «производит» на своих выходах. Для максимальной расширяемости в таком определении не следует задавать LFB, предполагаемые в качестве потребителей или поставщиков метаданных для этого LFB.

3.2.4.2. Создание и потребление метаданных

Для данного элемента метаданных в данном пути пакета **должен** быть хотя бы один создающий метаданные LFB, а также на этом пути **следует** иметь хотя бы один потребляющий LFB, которому нужны эти метаданные.

В модели ForCES производящие и потребляющие метаданные блоки LFB не обязаны быть смежными. Кроме того, для одних и тех же метаданных может быть множество производителей и потребителей. Когда путь пакета включает множество производителей одних и тех же метаданных, эти метаданные переписываются каждым следующим производителем.

Метаданные, которые производит LFB, задаются определением класса LFB на уровне группы выходных портов. Производитель может генерировать метаданные в группе портов всегда или при некоторых условиях. В первом случае мы называем метаданные безусловными, а во втором - условными. Например, LFB глубокой проверки пакетов производит несколько частей метаданных для пакета. Первым типом метаданных может быть протокол IP (TCP, UDP, SCTP, ...), а вторым - номера портов отправителя и получателя. Эти дополнительные элементы метаданных являются условными и зависят от первого элемента (протокол IP), поскольку они имеют смысл лишь для протоколов, использующих порты. Для условных метаданных в определении LFB следует обеспечивать возможность понять, когда эти метаданные создаются. Поведение потребляющего метаданные LFB, т. е. метаданные, требующиеся для работы LFB, задаётся в определении класса LFB на уровне группы входных портов. Группа может «требовать» определённых метаданных, а может считать их дополнительной информацией. В последнем случае определение класса LFB **должно** явно указывать, что произойдёт, если необязательные метаданные не будут предоставлены. Одним из вариантов является задание принятых по умолчанию значений для каждого необязательного элемента метаданных и использование таких значений в случаях когда метаданные не представлены с пакетом.

При задании тегов метаданных следует приложить усилия по гармонизации, чтобы производитель метаданных использовал те же теги, что и предполагаемые потребители.

3.2.4.3. Операции LFB над метаданными

Когда пакет обрабатывается в LFB (т. е. после получения но до пересылки пакета), этот LFB может выполнять операции чтения, записи и/или поглощения активных метаданных, связанных с пакетом. Если рассматривать LFB как чёрный ящик, с каждым активным элементом метаданных выполняется одна из перечисленных операций.

- IGNORE - игнорирование и пересылка.
- READ - считывание и пересылка.
- READ/RE-WRITE - считывание, запись нового значения и пересылка.
- WRITE - запись и пересылка (это может служить для создания нового элемента метаданных).
- READ-AND-CONSUME - считывание и поглощение.
- CONSUME - поглощение (удаление) метаданных без считывания.

Две последних операции завершают жизненный цикл элемента метаданных, который не уже пересылается вместе с пакетом следующему LFB.

В модели ForCES новый элемент метаданных создаётся LFB путём выполнения операции WRITE к типу метаданных, которого не было при получении пакета блоком LFB. Такое неявное создание может быть непреднамеренным для LFB, который может применять операцию WRITE, не задаваясь вопросом о наличии такого элемента метаданных, Если этот тип уже присутствует, он будет переписан, а в случае отсутствия будет создан заново.

Для LFB, вставляющих пакеты в модель, имеет смысл лишь операция WRITE.

LFB, удаляющие пакет из модели, могут выполнять операцию READ-AND-CONSUME (считывание) или CONSUME (игнорирование) каждого активного элемента метаданных, связанного с пакетом.

3.2.5. События LFB

В процессе работы могут возникать различные условия, которые будут детектироваться LFB. Примерами могут служить отказы каналов или перезапуск по таймеру LFB специального назначения. CE может захотеть уведомлений о таких событиях. Описание передачи таких сообщений и их формата является частью спецификации протокола ForCES [RFC5810]. Очевидно, что указание условий, при которых могут передаваться уведомления относится к модели.

События указываются в определении класса LFB. Декларация события LFB включает:

- уникальный 32-битовый идентификатор;
- указание компоненты LFB, используемой для активизации события, которую называют сигналом события;
- условия, при которых сигнал события будет приводить к генерации сообщения о событии для CE (примерами могут служить создание или удаление объекта, изменение конфигурации и т. п.);
- что элементу FE следует передать CE при выполнении условий.

Объявление событий внутри класса LFB по существу определяет, для каких компонент LFB нужен мониторинг применительно к событию, какие условия должны должен обнаружить элемент FE для констатации события и что следует сообщать CE в ответ на событие.

Хотя события могут объявляться в определении класса LFB, в процессе работы они управляются с помощью встроенных свойств события с использованием свойств компонент LFB (см. параграф 3.2.6). CE подписывается на события экземпляра класса LFB путём установки свойства события для подписки. Каждое событие имеет свойство подписки, которое по умолчанию отключено. Элементу CE, желающему получать определённые события, требуется включить свойство подписки в процессе работы.

Свойства событий также обеспечивают семантику фильтрации в процессе работы. CE может установить свойства для дополнительного подавления событий, на которые он уже подписан. Модель LFB определяет фильтры, включающие пороговые значения, гистерезис, временные интервалы, число событий и т. п.

Содержимое отчётов о событиях обеспечивает получение простой информации, которая может потребоваться CE для реакции на событие. Уведомления не предназначены для передачи сведений, которые уже имеются у CE, больших объёмов информации или данных, относящихся к сложным формам.

С концептуальной точки зрения в процессе работы обслуживание событий делится на 4 части.

1. Детектирование сигнала о событии (объявленного в определении класса LFB). Если CE подписан (во время работы) на событие, выполняется следующий этап.
2. Проверка условий (заданы при определении класса LFB) генерации события. Если условия выполняются, переход к следующему этапу.
3. Проверка установленных (во время работы) фильтров событий для определения дальнейшей судьбы события. Если о событии нужно уведомлять, выполняется следующий этап.
4. Представления отчёта (уведомления) элементу CE.

Более подробное описание событий приведено в параграфе 4.7.6.

3.2.6. Свойства компонент

Блоки LFB и структуры состоят из компонент, содержащих информацию о функционировании LFB, которую CE нужно просматривать и/или менять. Эти компоненты, как описано в параграфе 4.7, могут быть базовыми значениями, комплексными структурами (содержат множество компонент, которые могут быть значениями, структурами и таблицами) или таблицами (которые содержат значения, структуры или таблицы). Компоненты могут быть определены так, что их присутствие в экземплярах LFB будет не обязательно. Доступ к чтению и записи для компонент определяется реализацией FE. Элемент CE должен знать эти свойства. Кроме того, некоторые типы компонент (массивы и таблицы, псевдонимы, события) имеют дополнительную информацию о свойствах, которую элементу CE может потребоваться считывать или записывать. Модель задаёт структуру информации о свойствах для всех определяемых типов данных.

Более подробное описание свойств дано в параграфе 4.8.

3.2.7. Версии LFB

Версии классов LFB обеспечивают возможность поэтапного изменения этих классов. Обычно элементам FE не разрешается включать экземпляры LFB разных версий того или иного класса. Наследование (см. параграф 3.2.8) имеет некоторые правила. Если модель пути данных FE содержит экземпляр LFB некого класса C, а также экземпляр LFB другого класса C', который является «наследником» C, классы C и C' будут иметь разные версии.

Для поддержки версий класса LFB в определение класса нужно включать строку версии. Элементы CE могут поддерживать множество версий определённого класса LFB для обеспечения совместимости, однако FE **недопустимо** поддерживать более одной версии данного класса.

Использование версий не ограничивается обеспечением совместимости с ранними версиями. Предполагается, что они будут служить для внесения изменений, которые не могут быть представлены наследованием. Часто это будет исправление ошибок и совместимости со старыми версиями просто не может быть обеспечено. Версии могут также применяться при удалении компонент, которые признаны бесполезными (особенно в тех случаях, когда удаляемая компонента считалась обязательной).

3.2.8. Наследование LFB

Наследование класса LFB поддерживается в модели FE как метод определения новых классов LFB. Это также позволяет разработчикам FE добавлять фирменные расширения в стандартизованные LFB. Спецификация класса LFB **должна** указывать базовый класс и номер версии класса, которому она наследует (по умолчанию базовый класс LFB). Множественное наследование не поддерживается в целях предотвращения неоправданных сложностей.

Наследование следует применять лишь при использовании значительной части определения родительского класса LFB. При малом объёме или отсутствии повторно используемых определений следует определять новый класс LFB.

Интересным вопросом, связанным с наследованием, является совместимость наследника с классом-предком. Предположим существование некого стандартизованного класса LFB L1. Предположим также, что компания A создаёт FE, который реализует LFB L1, а компания B создаёт CE, который может распознавать и взаимодействовать с LFB L1. Пусть новый класс LFB L2 является наследником L1 и расширяет его возможности. Рассмотрим вопрос совместимости FE с прежней версией в контексте того, что производитель A обновил свой FE с версии L1 до версии L2, а B не обновил свой элемент CE. Основанный на старом L1 элемент CE сможет взаимодействовать с FE версии L2, если производный класс LFB L2 действительно совместим с базовым классом L1.

Обратный случай (переход CE на LFB L2 без обновления FE) вызывает гораздо меньше проблем. Отметим, что пока CE способен работать со старыми классами LFB, это не оказывает влияния на модель, поэтому термин «совместимость со старыми версиями» (backward compatibility) относится только к элементам FE.

Совместимость со старыми версиями может быть встроена в модель наследования путём ограничения наследования LFB требованием включения родительского класса в производный как подмножества (т. е. производный класс может только добавлять функции к базовому классу, но не может удалять имеющиеся в том функции). В дополнение к этому FE должен поддерживать перечисленные ниже механизмы для обеспечения совместимости со старыми версиями.

1. При обнаружении экземпляра LFB неизвестного элементу CE типа CE **должен** быть способен запросить базовый класс этого типа LFB у элемента FE.
2. Экземплярам LFB на FE **следует** поддерживать режим совместимости со старыми версиями (это означает возврат экземпляра LFB к базовому классу), а элементам CE **следует** поддерживать возможность настройки LFB для работы в таком режиме.

3.3. Адресация модели ForCES

На рисунке 5 показана абстракция двух разных объектов модели ForCES. Протокол ForCES обеспечивает механизмы однозначного указания компонент экземпляра класса LFB.

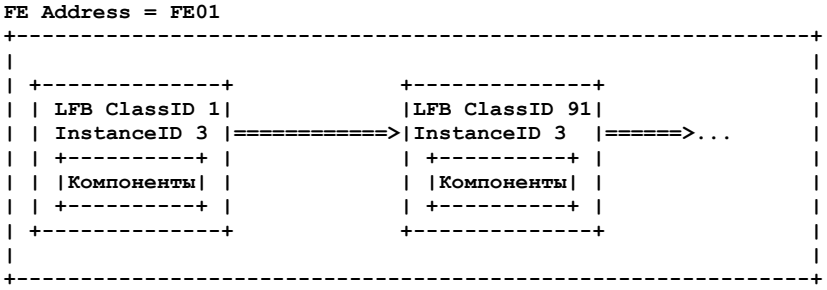


Рисунок 5. Иерархия FE Object.

На верхнем уровне иерархии адресации размещается идентификатор FE. В приведённом выше примере 32-битовый идентификатор FE обозначен FE01. Следующим 32-битовым селектором элемента является LFB ClassID. В нашем примере идентификаторы классов LFB имеют значения 1 и 91. далее на рисунке показано по одному экземпляру каждого класса. 32-битовые идентификаторы экземпляров класса LFB имеют значение только внутри данного класса LFB. Чтобы подчеркнуть это, на рисунке показаны экземпляры разных классов 1 и 91 с одним идентификатором 3.

Используя описанную схему адресации, сообщение можно отправить по адресу FE01, LFB ClassID 1, LFB InstanceID 3, с помощью протокола ForCES. Однако для повышения эффективности, такому сообщению следовало бы указывать также объекты внутри LFB, которые могут включать состояния, возможности и т. п (рисунок 6).

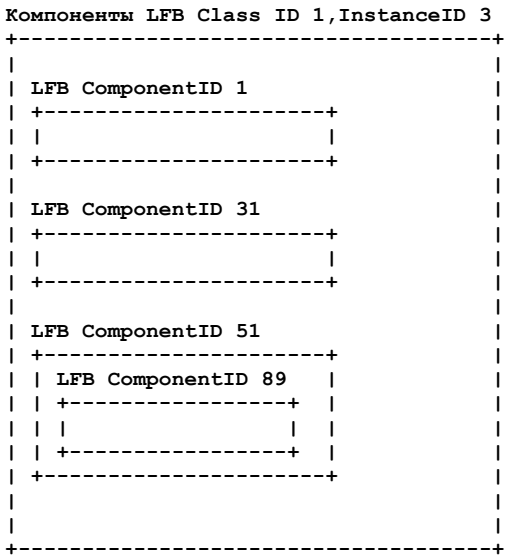


Рисунок 6. Иерархия LFB.

Показанные на рисунке 6 компоненты относятся к LFB Class ID 1, LFB InstanceID 3 на рисунке 5.

LFB ComponentID 51 может представлять таблицу (массив). В этом случае для выбора компоненты LFB с ID 89 из седьмой записи в таблице можно использовать путь 51.7.89. В дополнение к поддержке явного выбора элемента таблицы путём включения индекса в разделённый точками путь модель позволяет указывать элементы таблиц по содержимому. Это называется использованием ключа или индексацией по ключу. Например, если ComponentID 51 представляет собой таблицу с поддержкой индексации по ключу, описывающий содержимое ключ также может передаваться элементом CE вместе и идентификатором таблицы 51 для выбора таблицы, а за ним будет следовать путь 89 для выбора элемента структуры в таблице, который после расчёта в FE будет преобразовываться в LFB ComponentID 89 внутри заданной строки таблицы.

3.4. Моделирование пути данных FE

Пакеты, приходящие в FE из входных портов, обычно проходят через один или множество LFB прежде, чем попадут в выходные порты. Трактовка пакета элементом FE зависит от типа пакета (например, IPv4, IPv6, MPLS), значений заголовков, времени прибытия и других факторов. Результат обработки LFB может влиять на трактовку пакета нисходящими LFB. Эти различия можно концептуально рассматривать как наличие нескольких путей данных в FE. Например результат классификации по 6 элементами в LFB классификатора может влиять на выбор диапазона скорости в LFB измерителя на дальнейшем пути пакета.

Топология LFB представляется в форме направленного графа логических путей данных в FE, где узлы представляют экземпляры LFB, а направленные ребра - направление потока пакетов от одного LFB к следующему. В параграфе 3.4.1 рассмотрено моделирование путей данных FE с помощью топологии LFB, а в параграфе 3.4.2 рассматриваются вопросы, связанные с реконфигурацией топологии LFB.

3.4.1. Другие подходы к моделированию пути данных FE

Есть два базовых подхода к разделению трактовки пакетов в FE, один из которых представляет пути данных напрямую и графически (топология), а другой использует метаданные (представление состояний).

- Топологический подход

  При использовании этого подхода различие трактовки пакетов выражается расщеплением топологии LFB на несколько путей. Иными словами, если результат операции LFB определяет дальнейшую обработку пакета, такой LFB будет иметь свои выходные порты для каждого варианта последующей обработки, задающие нисходящее направление для пакетов.

- Кодирование состояний

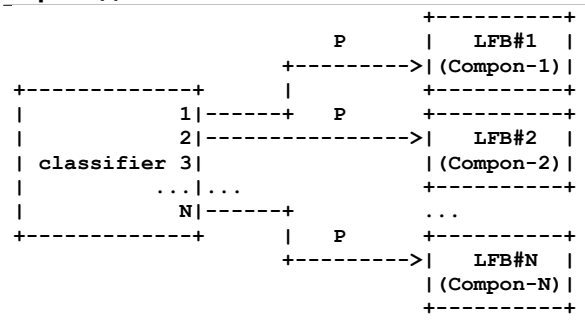
  Другим вариантом разделения трактовки является применение метаданных. Результат операции LFB может быть представлен элементом метаданных, передаваемым вместе с пакетом нисходящим LFB. Такой LFB может использовать полученные метаданные и их значение (например, индекс некой таблицы) для определения способа обработки пакета.

Теоретически эти два подхода взаимозаменяемы, поэтому можно было бы рассмотреть один вариант описания всех путей данных в FE. Однако ни одна из этих моделей сама по себе не даёт лучшего представления для практически важных случаев. Для конкретного FE с некими логическими путями данных применение двух разных вариантов моделирования будет приводить к существенно различающимся представлениям графа топологии LFB. Модель, использующая лишь топологический подход, будет требовать очень большого графа с множеством каналов или путей и узлов (экземпляры LFB) для представления всех возможных путей данных. С другой стороны, модель, использующая лишь кодированные состояния, будет ограничена строками LFB, которые не обеспечат интуитивно понятного описания разных путей данных (таких как MPLS и IPv4). Поэтому на практике скорей всего будет применяться сочетание этих подходов. Как будет показано ниже, эти два подхода можно комбинировать даже в одном LFB.

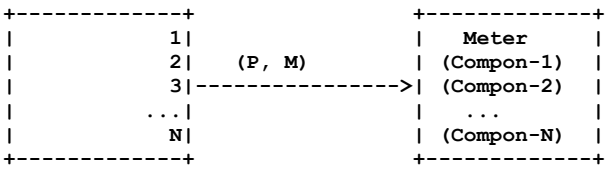
Используя простой пример классификатора с N выходами, подключёнными к другим LFB, рисунок 7.a показывает, как выглядит топология LFB при использовании «чистого» топологического подхода. Каждый выход классификатора подключён к одному из N блоков LFB и метаданные не требуются. Топологический подход прост и интуитивно понятен. Однако, если число N велико и N узлов, следующих за классификатором (LFB#1, LFB#2, ..., LFB#N), относятся к одному типу LFB (например, измеритель), но каждый из них имеет свои независимые компоненты, подход с кодированием состояний обеспечивает более простое представление топологии, как показано на рисунке 7.b. Этот подход требует таблицы из N строк с компонентами измерителей, обеспеченными узлами Meter, где каждая строка представляет атрибуты одного экземпляра измерителя. Нужны также метаданные M, которые передаются вместе с пакетом P от классификатора к измерителю, чтобы тот мог использовать M в качестве ключа поиска (индекса) нужной строки атрибутов, для конкретного пакета P.

Что же будет в случае N разнотипных узлов (LFB#1, LFB#2, ..., LFB#N)? Например, если LFB#1 является очередью, а остальные - измерителями? Хотя и в этом случае можно использовать один из двух описанных вариантов, лучше будет объединить их. На рисунке 7.c показаны два функционально различающихся пути с использованием обеих моделей.

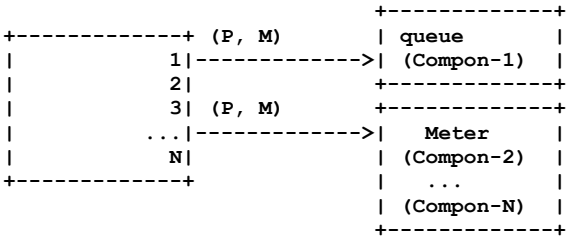
В этом примере были показаны преимущества каждого из подходов в определённых ситуациях. При использовании кодированных состояний обычно требуется меньше соединения между узлом с множеством выходов (fan-out) и следующими экземплярами LFB одного типа, поскольку каждый пакет сопровождается метаданными, которые следующие узлы могут интерпретировать для определения обработки пакета. Для таких случаев топологический подход требует построения сложных и громоздких графов с множеством соединений. С другой стороны, топологическая модель интуитивно понятна за счёт графического представления путей.



(a) Использование «чистого» топологического подхода



(b) Использование состояний для представления топологии LFB (a) с однотипными LFB#1, LFB#2, ..., LFB#N (например, измерители).

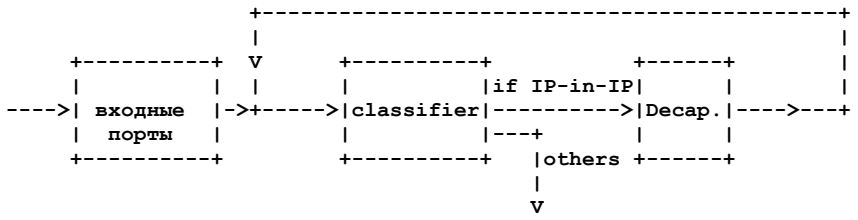


(c) Использование комбинированного подхода при разнотипных LFB#1, LFB#2, ..., LFB#N (например, очереди и измерители).

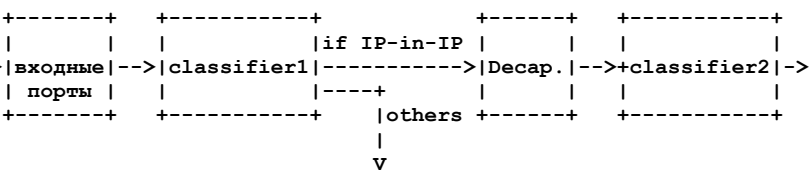
Рисунок 7. Пример моделирования путей данных FE.

Для комплексных топологий более гибкое решение обеспечивает комбинированный подход. Топологический подход следует применять в основном для случаев когда путь пакетов данных разветвляется в разные классы LFB (не просто различающиеся параметрами экземпляры одного класса LFB) и не требуются изменений типа добавления и удаления выходов LFB или такие изменения редки.

Конфигурационную информацию, которую требуется часто менять, следует представлять с использованием внутренних атрибутов одного или множества LFB (и поэтому следует применять кодирование состояний).



(a) Топология LFB с логической петлей.



(b) Топология LFB без петель с двумя независимыми экземплярами классификатора.

Рисунок 8. Пример топологии LFB.

Важно отметить, что описанная здесь топология LFB является логической и не связана с топологией физических соединений оборудования FE. Тем не менее, реальная реализация может оказывать влияние на отображение её функциональности на топологию LFB. На рисунке 8 представлен пример FE, где пакет IP-in-IP от приложения IPsec (типа VPN) может сначала проходить через классификатор, который будет принимать во внимание внешний заголовок IP. После классификации пакета как IP-in-IP он будет передан в декапсулятор для удаления внешнего заголовка IP, а затем будет снова классифицироваться по внутреннему заголовку IP. При использовании одного программного или аппаратного модуля классификации для внутреннего и внешнего заголовка IP с общим набором правил фильтрации в логической топологии LFB естественным образом возникает петля, показанная на рисунке 8.a. Однако при использовании двух разных наборов фильтров (например, один набор для внешних заголовков IP, другой для внутренних) более естественным будет применение двух разных экземпляров LFB классификатора (рисунок 8.b).

3.4.2. Настройка топологии LFB

Хотя сомнений в необходимости настройки индивидуальных LFB нет, вопрос настройки конфигурации для топологии LFB более сложен. Поскольку топология LFB реально является графическим представлением путей данных в FE, настройка топологии LFB означает динамическое изменение путей данных, включая изменение LFB на путях данных FE (например, создание, реплицирование, обновление или удаление LFB), а также организацию и удаление соединений между выходами восходящих и входами нисходящих LFB.

Почему пути данных в FE меняются динамически? Эти пути создаются элементами CE для предоставления неких услуг уровня данных (например, Diffserv, VPN) сетевым элементам (NE) потребителей. Целью изменения конфигурации путей данных является обеспечение элементам CE возможности настройки услуг, предоставляемых NE, в процессе работы. CE нужно менять пути данных при изменении требований к сервису типа добавления нового потребителя или изменении потребностей имеющегося заказчика. Однако следует отметить, что не все изменения путей данных меняют граф топологии LFB. Изменения графа зависят от подхода, используемого при отображении путей данных на топологию LFB. Как было отмечено в параграфе 3.4.1, топологический подход и подход, основанный на кодировании состояний приводят к существенно разным представлениям топологии LFB для одних и тех же путей данных. В общем случае топология LFB, основанная на чистом топологическом подходе, скорей всего будет приводить к более частым изменениям по сравнению с использованием подхода с кодированием состояний. Однако даже для топологии, основанной только на представлении состояний, время от времени может потребоваться изменение (например, для обхода некоторых LFB или добавления LFB). Поскольку для моделирования путей данных используется комбинация обоих подходов, изменение топологии LFB является важным аспектом модели FE.

Мы хотим отметить, что поддержка настраиваемой топологии LFB в модели FE не требует от всех FE реализации такой возможности. Если даже FE поддерживает настраиваемую топологию LFB, он может вносить ограничения для такой настройки. Оптимизированные по производительности аппаратные реализации могут не иметь возможностей такой настройки или сильно ограничивать их, а реализации FE на базе сетевых процессоров могут обеспечивать значительную гибкость и настраиваемость. Разработчики FE могут самостоятельно принимать решение о возможности настройки топологии и ограничениях для неё. Выбор простого решения для включения или отключения (обхода) некоторых LFB или более гибкой программной настройки конфигурации определяется внутренним устройством FE и выходит за рамки модели FE. В любом случае элементы CE **должны** быть способны определить возможности настройки FE. Поэтому модель FE **должна** обеспечивать механизм для описания возможностей настройки топологии LFB внутри элементов FE. Эти возможности могут включать (см. раздел 5):

- указание классов LFB, экземпляры которых может создавать FE;
- максимальное число экземпляров одного класса LFB;
- топологические ограничения:
  - максимальное число экземпляров одного или разных классов, которые могут быть созданы в данной ветви графа;
  - ограничения по упорядочению LFB (например, любой экземпляр LFB класса A всегда должен быть нисходящим для любого экземпляра LFB класса B).

Элементам CE нужна некоторая программная поддержка, чтобы справиться с рядом сложностей. Иными словами, даже при возможности CE настраивать топологию LFB для элемента FE, не предполагается возможность CE интерпретировать произвольную топологию LFB и определить конкретные услуги или приложения (например, VPN, Diffserv), поддерживаемые FE. Однако, если CE может грубо оценить возможности FE, элемент CE **должен** настроить топологию LFB для реализации сетевого сервиса, который требуется от NE. Таким образом, отображение, которое должно быть понятно CE, относится к сервису NE верхнего уровня для конкретной топологии LFB, а не наоборот. Не предполагается наличие у CE интеллектуальных возможностей трансляции политики сервиса на верхнем уровне в данные конфигурации для элементов FE. Однако можно предположить, что внутри данного домена сетевого сервиса некоторый интеллект может быть запрограммирован в CE, чтобы позволить тому понять какие LFB вовлечены в процесс для трансляции политики верхнего уровня в конфигурацию нижнего уровня для FE, задаваемую автоматически. Отметим, что это относится к внутренней реализации уровня управления и выходит за рамки модели FE, поэтому подробно в этом документе не рассматривается.

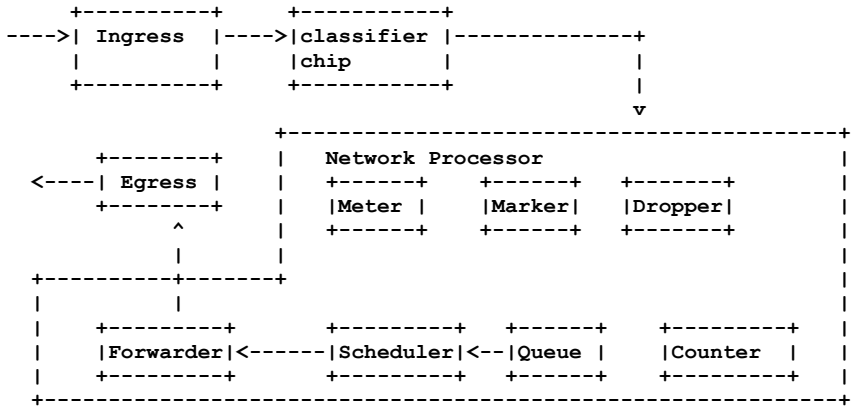


Рисунок 9. Представление возможностей FE элементу CE.

На рисунке 9 приведён пример, где маршрутизатор с поддержкой QoS[1] имеет несколько интерфейсных плат с небольшим числом входных и выходных портов, специализированный контроллер классификатора и сетевой процессор с кодом блоков FE типа измерителей, маркировщиков, отбрасывателей, счётчиков, очередей, планировщиков и модулей пересылки IPv4. Часть топологии LFB уже зафиксирована и остаётся неизменной в соответствии с физическими соединениями интерфейсных плат. Например, все входные порты могут быть жёстко соединены с контроллером классификации, чтобы все входящие пакеты сначала классифицировались. С другой стороны, блоки LFB на сетевом процессоре и порядок их применения являются программируемыми. Однако имеются некоторые ограничения возможностей и соединений между этими LFB. Ограничения возможностей могут включать:

- 8 измерителей;
- 16 очередей в одном FE;
- планировщик не может обслуживать больше 16 очередей;

[1]Quality-of-service - качество обслуживания.

- канальные ограничения могут включать:
  - после классификатора может следовать:
    - измеритель;
    - маркировщик;
    - отбрасыватель;
    - счётчик;
    - очередь или модуль пересылки IPv4но не планировщик;
  - за очередью может следовать только планировщик;
  - за планировщиком должен следовать модель пересылки IPv4;
  - последним LFB в пути данных перед отправкой в выходной порт должен быть модуль пересылки IPv4.

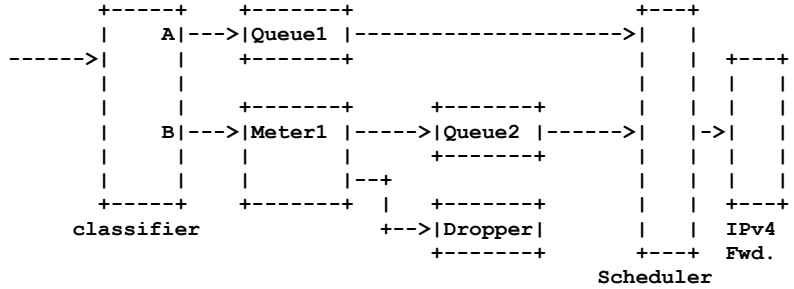


Рисунок 10. Топология LFB, настроенная CE и воспринятая FE.

Когда FE сообщил о своих возможностях и ограничениях элементу CE, тот может транслировать правила QoS в желаемую конфигурацию для FE. На рисунке 9 показаны возможности FE, а рисунки 10 и 11 показывают 2 топологии, которые CE может настроить в FE. Отметим, что рисунок 11 неполон, поскольку каналы между LFB не указаны.

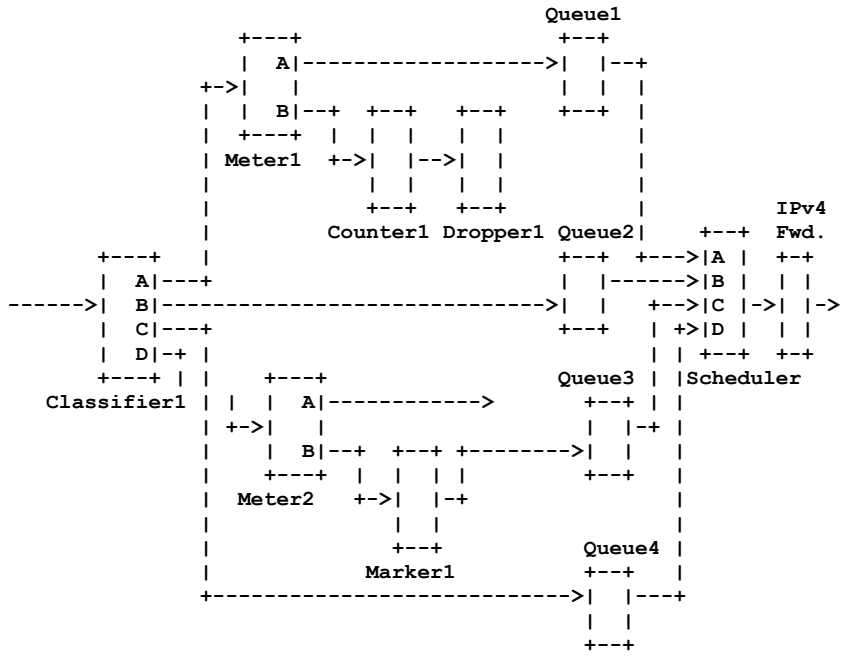


Рисунок 11. Топология LFB, настроенная CE и воспринятая FE.

Отметим также, что на рисунках 10 и 11 для упрощения не показаны входы и выходы. Топология на рисунке 11 существенно сложнее, чем на рисунке 10, но оба варианта реализуемы в рамках возможностей FE и элемент FE может воспринять запрос такой конфигурации от CE.

4. Модель и схема для классов LFB

Основной целью модели FE является предоставление абстрактного базового модульного представления FE, не зависящего от реализации. Эта задача облегчается использованием блоков LFB, создаваемых из классов LFB. Классы LFB и связанные с ними определения будут представлены в наборе документов XML. Этот набор документов XML называется библиотекой классов LFB, а каждый из документов называется документом библиотеки классов LFB (или документом библиотеки для краткости). Каждый документ библиотеки **должен** соответствовать схеме, представленной в этом разделе. Схема и правила соответствия ей определяются консорциумом W3C в определениях схемы XML [Schema1] и типов данных схемы XML [Schema2]. Корневым элементом документа библиотеки является <LFBLibrary>.

Обмен документами библиотек «по проводу» между элементами FE и CE не предполагается. Однако модель служит основой для проектирования и разработки элементов CE (программы) и FE (в основном программная часть). Она будет также служить основой спецификации элементов протокола ForCES для коммуникаций между CE и FE.

В последующих параграфах описаны части документа XML для LFBLibrary. Описания содержат в основном семантическую информацию, требуемую для понимания и использования элементов XML. Приведённая ниже схема XML обеспечивает окончательное определение разрешённых элементов и их базовый синтаксис. К сожалению ограниченность английского языка и XML вносят ограничения, описанные в семантических разделах, которые не могут

быть полностью заданы в схеме XML, поэтому для создания документа библиотеки требуется использовать оба набора информации.

4.1. Пространство имён

Пространство имён требуется для однозначной идентификации типа LFB в библиотеке классов LFB. Ссылки на определение пространства имён приведены в разделе 9. Взаимодействие с IANA.

4.2. Элемент <LFBLibrary>

Элемент <LFBLibrary> является корнем всех библиотечных документом. Документ библиотеки содержит последовательность элементов верхнего уровня. Ниже приведён список всех элементов, которые могут присутствовать непосредственно в <LFBLibrary> (имеющиеся элементы должны располагаться в указанном порядке.

- <description> содержит текстовое описание назначения библиотечного документа;
- <load> задаёт загрузку информации из других библиотечных документов;
- <frameDefs> служит для объявления кадров;
- <dataTypeDefs> служит для определения типов данных общего назначения;
- <metadataDefs> служит для определения метаданных;
- <LFBClassDefs> служит для определения классов LFB.

Все элементы являются необязательными. Один документ библиотеки может содержать лишь метаданные, другой - только определения классов LFB, а третий - все перечисленное выше.

Документ библиотеки может импортировать другие библиотечные документы, если ему нужно ссылаться на содержащиеся в них определения. Это походе на директиву #include в языке программирования C. Импорт указывается с помощью элементов <load>, которые должны предшествовать в документе перечисленным выше элементам. Для однозначных ссылок каждый экземпляр документа LFBLibrary имеет уникальную метку, определённую в атрибуте provide элемента LFBLibrary. Отметим, что это включение относится к протоколу ForCES, а не к XML. Включается семантика содержимого библиотеки, указанной элементом <load>, а не содержимое XML. Кроме того, с точки зрения концептуальной обработки элементов <load> полный набор загружаемых документов считается одним документом. Указанный документ включается в этот набор лишь однократно, даже если он указан в нескольких элементах <load> (даже в разных файлах). Поскольку обработка информации LFBLibrary не зависит от порядка документов, порядок обработки загружаемых элементов определяется разработчиком и общий эффект будет определяться общим набором информации во всех указанных ссылками документах. Отметим, что такая компьютерная обработка библиотечных документов модели ForCES может быть полезна для различных реализаций, но не требуется для определения библиотек или реальных операций самого протокола.

Ниже приведён шаблон библиотечного документа.

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  provides="this_library">

  <description>

  </description>

  <!-- Загрузка внешних библиотек (необязательно) -->
  <load library="another_library"/>
   ...
  <!-- Определения типов кадров (необязательно) -->
  <frameDefs>
   ...
  </frameDefs>

  <!-- Определения типов данных (необязательно) -->
  <dataTypeDefs>
   ...
  </dataTypeDefs>

  <!-- Определения метаданных (необязательно) -->
  <metadataDefs>
   ...
  </metadataDefs>

  <!--
    -
    -
     Определения классов LFB (необязательно) -->
  <LFBCLassDefs>

  </LFBCLassDefs>
  </LFBLibrary>
```

4.3. Элемент <load>

Этот элемент применяется для ссылки на другой документ библиотеки LFB. Подобно директиве #include в языке C, он делает объекты (типы метаданных и данных и т. п.), определённые в указанном документе библиотеки, доступными в текущем документе.

Элемент **должен** содержать метку библиотечного документа для включения и **может** включать идентификатор URL для указания места хранения документа. Элемент <load> может повторяться неограниченное число раз. Ниже представлены три примера элемента <load>.

```
<load library="a_library"/>
<load library="another_library" location="another_lib.xml"/>
<load library="yetanother_library"
 location="http://www.example.com/forces/1.0/lfbmodel/lpm.xml"/>
```

4.4. Элемент <frameDefs> для определения типа кадра

Имена кадров используются в определении LFB для указания типов кадров, которые LFB ожидает на входных портах и выдаёт в выходные порты. Необязательный элемент <frameDefs> в документе библиотеки содержит один или множество элементов <frameDef> каждый из которых указывает один тип кадров.

Каждое указание типа кадра **должно** включать уникальное имя (NMTOKEN) и краткое описание. Дополнительно **может** представляться более подробное описание.

Уникальность имён типов **должна** обеспечиваться в рамках библиотечного документа и всех включённых в него (напрямую или опосредованно) библиотечных документов.

Приведённый ниже пример указывает два типа кадров.

```
<frameDefs>
  <frameDef>
   <name>ipv4</name>
   <synopsis>IPv4 packet</synopsis>
   <description>
    Этот тип кадров относится к пакетам IPv4.
  </description>
 </frameDef>
  <frameDef>
  <name>ipv6</name>
  <synopsis>IPv6 packet</synopsis>
  <description>
    Этот тип кадров относится к пакетам IPv6.
  </description>
 </frameDef>
  ...
</frameDefs>
```

4.5. Элемент <dataTypeDefs> для определения типа данных

Необязательный элемент <dataTypeDefs> может служить для определения типов данных общего пользования. Он содержит один или множество элементов <dataTypeDef>, каждый из которых определяет тип данных с уникальным именем. Такие типы данных могут затем использоваться в разных местах файла библиотеки, включая:

- определения других типов данных;
- определения компонент классов LFB.

Это похоже на концепцию общего заголовочного файла с типами данных общего пользования.

Каждый элемент <dataTypeDef> **должен** включать уникальное имя (NMTOKEN), краткое описание и элемент определения типа. Уникальность имён типов **должна** обеспечиваться в рамках библиотечного документа и всех включённых в него (напрямую или опосредованно) библиотечных документов. Элемент <dataTypeDef> **может** также включать дополнительное, более подробное описание.

```
<dataTypeDefs>
  <dataTypeDef>
    <name>ieeemacaddr</name>
     <synopsis>48-битовый адрес IEEE MAC</synopsis>
      ... определение типа ...
  </dataTypeDef>
  <dataTypeDef>
    <name>ipv4addr</name>
     <synopsis>Адрес IPv4</synopsis>
     ... определение типа ...
  </dataTypeDef>
  ...
</dataTypeDefs>
```

Существует два типа данный - неделимые (atomic) и композитные (compound). Неделимые типы данных подходят для переменных с одним значением (например, integer, string, byte array).

Ниже приведён список встроенных типов atomic, но можно определить другие неделимые типы с помощью элементов <typeRef> и <atomic>.

Имя <name>	***Значение***
`char`	8-битовое целое число со знаком.
`uchar`	8-битовое целое число без знака.
`int16`	16-битовое целое число со знаком.
`uint16`	16-битовое целое число без знака.
`int32`	32-битовое целое число со знаком.
`uint32`	32-битовое целое число без знака.
`int64`	64-битовое целое число со знаком.
`uint64`	64-битовое целое число без знака.
`boolean`	Значение true (1) или false (0).

`string[N]`	Строка символов UTF-8 размером не более N октетов.
`string`	Строка символов UTF-8 без заданного ограничения размера.
`byte[N]`	Массив из N байтов.
`octetstring[N]`	Буфер размером N октетов, который может бы заполнен не до конца. Поэтому кодированное значение всегда включает размер.
`float32`	32-битовое число с плавающей запятой в формате IEEE.
`float64`	4-битовое число с плавающей запятой в формате IEEE.

Эти встроенные типы данных готовы для использования при определении метаданных или атрибутов LFB, но могут также служить для определения новых типов данных. Тип boolean определён здесь по причине его широкого использования, хотя его можно создать путём сужения типа uchar, как делается в типах atomic (параграф 4.5.2).

Композитные типы данных можно создавать или неделимых типов и других композитных типов. Для определения композитных типов имеется четыре способа. Их можно определять как массив компонент некого композитного или неделимого типа, как структуру именованных компонент неделимых или композитных типов (аналог структур C), как объединение (union) именованных компонент неделимых или структурных типов (аналог C union), а также как дополнения существующих композитных типов (см. параграф 4.5.7).

С учётом того, что протокол ForCES будет принимать и устанавливать значения компонент, все используемые здесь неделимые типы должны передаваться протоколом ForCES. Кроме того, для протокола ForCES нужен механизм передачи композитных типов. Однако детали таких представления относятся к документу, определяющему протокол ForCES [RFC5810], а не к данной модели. Типы string и octetstring должны передаваться протоколом вместе с полем размера, поскольку их фактический размер не задаётся определением.

Для строковых типов модель задаёт небольшой набор ограничений и определений способов структурирования. Ограничения размера для string и octetstring могут быть указаны в определениях классов LFB. Свойства компонент string и octetstring также включают действительные размеры и ограничения размера. Дублирование ограничений позволяет реализациям сократить максимальные размеры переменных, заданные в определении класса LFB. В любом случае ограничения задаются числом октетов, а не символов. При работе протокола если указанный размер не превосходит возможности FE, элемент FE полностью сохраняет содержимое строки, представленной CE, и возвращает такие строки по запросу. Элемент FE не выполняет преобразований строк. Компоненты типа string или string[n] **могут** применяться для хранения идентификаторов сопоставляемых с компонентами других LFB. В таких случаях должно проверяться пооктетное соответствие без каких-либо преобразований. Протокол ForCES [RFC5810] не проверяет и не требует проверки пригодности содержимого строк UTF-8. Однако строки UTF-8 **следует** кодировать в кратчайшей форме для предотвращения возможных проблем безопасности, описанных в [UNICODE]. Предполагается, что любой объект, отображающий такие строки, сам проверяет их пригодность (например, для корректировки многобайтовых символов или гарантии того, что строка не завершается в середине многобайтовой последовательности). Определения конкретных классов LFB **могут** ограничивать содержимое строк с учётом их применения (например, компоненты с именами DNS могут быть ограничены использованием лишь допустимых в таких именах октетов). Элементам FE следует проверять содержимое запросов SET для компонент с такими ограничениями при установке запрошенных значений путём простого сравнения с диапазоном разрешённых для компоненты символов. Протокол ForCES определяет нормативную обработку для используемых протоколом запросов.

Для задания типа в элементах <dataTypeDef> доступны элементы <typeRef>, <atomic>, <array>, <struct> и <union>.

Предопределённый псевдоним (alias) типа занимает промежуточное положение между типами данных atomic и compound. Псевдонимы служат для того, чтобы компоненты внутри LFB могли опосредованно ссылаться на другие компоненты внутри того же или другого класса или экземпляра LFB. Компонента alias ведёт себя подобно структуре, в которой одна компонента имеет специальное назначение. С учетом того, что особое поведение связано с другими частями структуры, результат считается предопределённой конструкцией.

4.5.1. Элемент <typeRef> для переименования имеющихся типов данных

Элемент <typeRef> указывает на существующий тип данных по его имени. Упомянутый тип данных **должен** быть определён в том же документе или в одном из включённых в него библиотечных документов. Если указанный тип данных является неделимым, определённый заново тип так же будет относится к числу типов atomic. Если указанный тип является композитным, вновь определённый тип также будет композитным. Пример использования показан ниже.

```
<dataTypeDef>
  <name>short</name>
  <synopsis>Псевдоним для int16</synopsis>
  <typeRef>int16</typeRef>
</dataTypeDef>
<dataTypeDef>
  <name>ieeemacaddr</name>
  <synopsis>48-битовый адрес IEEE MAC</synopsis>
  <typeRef>byte[6]</typeRef>
</dataTypeDef>
```

4.5.2. Элемент <atomic> для создания производных неделимых типов

Элемент <atomic> позволяет определить новый неделимый тип на основе существующего типа atomic, задавая ограничения диапазона и/или представляя специальные перечисляемые значения. Отметим, что элемент <atomic> в качестве базовых может использовать только неделимые типы и результатом **должен** быть другой неделимый тип.

Например, приведённый ниже фрагмент определяет новый неделимый тип данных dscp.

```
<dataTypeDef>
  <name>dscp</name>
  <synopsis>Код Diffserv.</synopsis>
  <atomic>
    <baseType>uchar</baseType>
    <rangeRestriction>
      <allowedRange min="0" max="63"/>
    </rangeRestriction>
```

```
      <specialValues>
        <specialValue value="0">
          <name>DSCP-BE</name>
          <synopsis>Best Effort</synopsis>
        </specialValue>
         ...
      </specialValues>
    </atomic>
  </dataTypeDef>
```

4.5.3. Элемент <array> для определения массивов

Элемент <array> может служить для создания новых композитных типов данных в форме массива композитных или неделимых типов. В зависимости от контекста этот и другие документы считают такие массивы и таблицы взаимозаменяемыми без учёта синтаксиса и семантики. Тип элементов массива может быть задан ссылкой на имеющийся тип (с помощью элемента <typeRef>) или определением неименованного типа внутри элемента <array> с помощью элементов <atomic>, <array>, <struct> или <union>.

Массив может иметь фиксированный или переменный размер, что указывается атрибутом type элемента <array>. По умолчанию размер считается переменным и для него может указываться необязательный атрибут maxlength, задающий максимальный размер. Этот атрибут следует применять для задания семантических ограничений, а не ограничений реализации. Последнее (поддержка ограничений реализации) следует задавать с помощью возможностей компонент классов LFB и никогда не следует включать максимальный размер в тип данных array, размер которого считается неограниченным.

Для массивов фиксированного размера **должен** указываться атрибут length, задающий постоянный размер массива.

Результатом этой конструкции всегда является композитный тип, даже если массив имеет постоянный размер 1.

Массивы **должны** индексироваться только целыми числами и предполагается начало отсчёта 0.

В дополнение к индексам **массив** может быть объявлен с использованием ключей. Это даёт несколько эффектов:

- любой объявленный ключ может применяться в контексте протокола ForCES при выборе компонент для операций (см. спецификацию протокола ForCES [RFC5810]);
- в любом экземпляре массива каждый объявленный ключ **должен** быть уникальным в рамках этого экземпляра, т. е. две компоненты массива не могут иметь одинаковых значений во всех полях, образующих ключ.

Каждый ключ объявляется с идентификатором keyID для использования в протоколе ForCES [RFC5810], где уникальный идентификатор формируется из одного или множества заданных полей ключа. Для поддержки случаев где массив неделимого типа с уникальными значениями может быть указан этими значениями в качестве идентификатора ключевого поля **может** применяться шаблон (т. е. элемент массива является ключом). Если типом значения массива является структура или массив, ключом служит одна или множество компонент значения, указываемых именами. Поскольку поле **может** быть компонентой включённой в массив структуры, компонентой компоненты структуры и т. д., имя поля на практике является конкатенацией идентификаторов компонент, разделённых точками. Синтаксис для идентификации полей ключа показан в приведённых ниже примерах массивов.

Этот пример показывает определение массива фиксированного размера с предопределенным типом данных.

```
  <dataTypeDef>
      <name>dscp-mapping-table</name>
      <synopsis>
         Таблица из 64 значений DSCP, используемая для переотображения кодов.
      </synopsis>
      <array type="fixed-size" length="64">
         <typeRef>dscp</typeRef>
      </array>
    </dataTypeDef>
```

Пример массива переменного размера с указанием верхнего предела.

```
    <dataTypeDef>
      <name>mac-alias-table</name>
      <synopsis>Таблица, содержащая до 8 адресов IEEE MAC</synopsis>
      <array type="variable-size" maxlength="8">
          <typeRef>ieeemacaddr</typeRef>
      </array>
    </dataTypeDef>
```

Пример массива с локальным (неименованным) определением типа содержимого.

```
    <dataTypeDef>
      <name>classification-table</name>
      <synopsis>Таблица правил классификации и кодов результата.</synopsis>
      <array type="variable-size">
        <struct>
          <component componentID="1">
            <name>rule</name>
            <synopsis>Правило для сопоставления</synopsis>
            <typeRef>classrule</typeRef>
          </component>
          <component componentID="2">
            <name>opcode</name>
            <synopsis>Код результата</synopsis>
            <typeRef>opcode</typeRef>
          </component>
       </struct>
      </array>
```

```
    </dataTypeDef>
```

В приведённом выше примере каждый элемент массива является структурой с двумя компонентами (rule и opcode).

Следующий пример показывает таблицу префиксов IP, доступ к элементам которой осуществляется по многокомпонентному ключу из адреса IP, префикса и источника данных. Это значит, что в любом экземпляре такой таблицы нет двух строк с совпадающими полями адреса, префикса и источника данных.

```
<dataTypeDef>
  <name>ipPrefixInfo_table</name>
  <synopsis>Таблица данных об известных префиксах</synopsis>
  <array type="variable-size">
    <struct>
      <component componentID="1">
        <name>address-prefix</name>
        <synopsis>Префикс, который будет описан</synopsis>
        <typeRef>ipv4Prefix</typeRef>
      </component>
      <component componentID="2">
        <name>source</name>
        <synopsis>Протокол или процесс, обеспечивший эту информацию</synopsis>
        <typeRef>uint16</typeRef>
      </component>
      <component componentID="3">
        <name>prefInfo</name>
        <synopsis>Информация, о которой мы заботимся</synopsis>
        <typeRef>hypothetical-info-type</typeRef>
      </component>
    </struct>
    <contentKey contentKeyID="1">
      <contentKeyField> address-prefix.ipv4addr</contentKeyField>
      <contentKeyField> address-prefix.prefixlen</contentKeyField>
      <contentKeyField> source</contentKeyField>
    </contentKey>
  </array>
</dataTypeDef>
```

Отметим, что элементами keyField могут также быть просто address-prefix и source, поскольку они включают все поля.

4.5.3.1. Ссылки на поля ключа

Чтобы использовать объявление ключа, нужно обращаться к компонентам, которые могут быть вложенными в другие компоненты массива. При наличии вложенных массивов возможно даже использование в качестве ключа элемента массива (следует принять меры по обеспечению уникальности).

Ключ представляет собой комбинацию значений всех полей, указанных в элементе keyField. Следовательно, значение элемента keyField **должно** быть конкатенацией идентификаторов полей, разделённых точками. Пробелы в ключе допустимы, но будут игнорироваться.

Допустимая строка для одного идентификатора поля в keyField зависит от текущего контекста. Изначально в объявлении ключа массива контекстом служит тип массива. Далее контекстом может стать любой тип, выбранный идентификаторами полей, которые к тому моменту были обработаны в текущем объявлении поля ключа.

Когда текущий контекст является массивом (например, при объявлении ключа для массива массивов), единственным допустимым значением для идентификатора поля является явное число.

Если текущим контекстом является структура, пригодными значениями идентификаторов полей являются имена компонент структуры. В специальном случае объявления ключа для массива неделимых типов, где содержимое уникально и применяется в качестве ключа, идентификатор одного поля ключа **должен** указываться символом *.

При использовании в ссылках массивов и структур возможно создать keyFields, которого не существует. В ссылках keyField не **следует** указывать необязательные компоненты структуры. Для ссылок на элементы массива следует соблюдать осторожность, чтобы обеспечить наличие элементов массива при создании или изменении элемента массива в целом. Несоблюдение этого будет приводить к ошибке FE при попытке создания и возврату соответствующего сообщения.

4.5.4. Элемент <struct> для определения структур

Структура состоит из набора компонент с данными. Каждая компонента имеет тип (неделимый или композитный) и имя, которое должно быть уникальным в области действия определения композитного типа. Это похоже на struct в языке C. Компоненты определяются с помощью элементов <component>. Элемент <struct> **может** содержать элемент <derivedFrom>, указывающий базовую структуру. Определение структуры **должно** включать не менее одного элемента <component>.

Фактический тип компоненты может определяться ссылкой на имеющийся тип (элемент <typeRef>) или задаваться локальным (неименованным) типом, созданным с использованием элементов <atomic>, <array>, <struct> или <union>.

Элемент <component> **должен** включать атрибут componentID, обеспечивающий числовой идентификатор компоненты для использования протоколом. В элемент <component> **должны** включаться также имя и краткое описание компоненты, дополнительно **может** включаться элемент <description> с более подробным описанием. Определение **может** также включать элемент <optional>, указывающий, что определяемая компонента не является обязательной. Определение **должно** содержать элементы, определяющие тип компоненты, как описано выше.

Для dataTypeDef определение структуры **может** быть унаследовано от ранее определённого структурированного типа и дополнено. Это указывается включением необязательного атрибута derivedFrom в объявление структуры до определения дополнений или замены компонент (см. параграф 4.5.7).

Атрибуты componentID для разных компонент структуры (или LFB) **должны** различаться. Задавать какой-либо порядок идентификаторов не требуется. Для удобочитаемости и упрощения обслуживания обычно определяется набор последовательных значений, но это не является требованием протокола.

Результатом этой конструкции всегда будет композитный тип, даже при наличии в <struct> лишь одного поля.

Пример определения структуры приведён ниже.

```
<dataTypeDef>
 <name>ipv4prefix</name>
 <synopsis>Префикс IPv4, определяемый адресом и размером префикса</synopsis>
 <struct>
  <component componentID="1">
   <name>address</name>
   <synopsis>Адресная часть префикса</synopsis>
   <typeRef>ipv4addr</typeRef>
  </component>
  <component componentID="2">
   <name>prefixlen</name>
   <synopsis>Размер префикса</synopsis>
   <atomic>
    <baseType>uchar</baseType>
    <rangeRestriction>
     <allowedRange min="0" max="32"/>
    </rangeRestriction>
   </atomic>
  </component>
 </struct>
</dataTypeDef>
```

4.5.5. Элемент <union> для определения типов объединений

Подобно объявлению union в языке C, эта конструкция позволяет определять оверлейные типы. Формат аналогичен формату элемента <struct>.

Эта конструкция всегда создаёт композитный тип даже при наличии в объединении лишь одного элемента.

4.5.6. Элемент <alias>

Иногда в LFB или структуре нужна компонента, ссылающаяся на информацию (или компоненту) в другом LFB. Это может позволять, например, использовать в ARP LFB таблицу трансляции адресов IP->MAC совместно с LFB локальной передачи без дублирования информации. Аналогично можно позволить LFB измеритель трафика разделять информацию с LFB формирования трафика. Для создания таких конструкций служит элемент <alias>. Конструкция позволяет сказать элементам CE и FE, что любые манипуляции с определёнными данными на деле применяются к данным, находящимся в некой заданной части другого указанного экземпляра LFB. Содержимое элемента <alias> **должно** быть именованным типом. Независимо от того, на какую компоненту указывает псевдоним (это определяется описанными ниже свойствами компоненты псевдонима), эта компонента должна иметь такой же тип, какой был заявлен для псевдонима. Таким образом, когда CE или FE разыменовывает компоненту псевдонима, тип возвращаемой информации известен. Этим типом может быть как базовый, так и производный тип. Когда операция GET или SET указывает на псевдоним, возвращается или изменяется значение, на которое этот псевдоним указывает (цель). Запись в псевдоним будет разрешена, если имеется возможность записи в цель псевдонима.

Цель компоненты, объявленной элементом <alias>, определяется информацией в свойствах компоненты. Подобно другим компонентам, её свойства включают поддержку разрешений на считывание и запись для псевдонима. В дополнение к этому в свойствах псевдонима имеется несколько полей (компонент), определяющих цель. Этими компонентами являются идентификаторы класса и экземпляра для LFB, а также последовательность целых чисел, представляющая пути в целевом LFB к целевой компоненте. Тип целевого элемента должен совпадать с объявленным типом псевдонима. Подробности структуры псевдонима описаны в параграфе 4.8.

Отметим, что свойства чтения и записи для псевдонима указывают на значения. CE может проверить возможность записи, лишь предприняв попытку выполнения такой операции (компоненты свойств сами не имеют свойств).

4.5.7. Дополнения

Композитные типы можно определять также путём дополнения имеющихся композитных типов. Если существующий тип является структурой, дополнение **может** добавлять в структуру новые элементы. Тип имеющейся компоненты **может** быть заменён в определении дополняющей структуры, но для замены **может** использоваться лишь дополнение, созданное из текущего типа имеющейся компоненты. Существующие компоненты нельзя удалить. Если имеющаяся компонента является массивом, дополнением будет являться дополнением типа элементов массива.

Дополнения **недопустимо** применять к объединению (union).

Одним из следствий этого является совместимость дополнений с базовым типом, из которого дополнение создано. Таким образом, дополнения полезны для определения компонент субклассов LFB, совместимых со своими «прародителями». Кроме добавления в класс новых компонент типы данных существующих компонент **могут** быть заменены их дополнениями и при этом будут соответствовать правилам совместимости для субклассов. Именно это служит причиной неприменимости дополнений к типу union.

Рассмотрим, например, простой базовый блок LFB класса A, имеющий одну компоненту (comp1) типа X. Одним из способов создать класс A1 из A является простое добавление второй компоненты (любого типа). Другим способом создать класс A2 из A является замена в A исходной компоненты (comp1) типа X на тип Y, который является дополнением (расширением) X. Оба класса A1 и A2 совместимы с классом A.

Синтаксис дополнения заключается во включении в определение структуры элемента <derivedFrom>, показывающего, что структура была дополнена. Именам и идентификаторам для новых компонент в дополнении **недопустимо**

совпадать с именами или идентификаторами в дополняемой структуре. Для компонент, где тип имеющихся данных был замене подходящим для дополнения типом данных, имеющееся имя и идентификатор компоненты **должны** использоваться в дополнении. Кроме ограничений на существующие компоненты, не задаётся других требований к идентификаторам компонент (последовательность, возрастание или иные связи с имеющимися идентификаторами). Предполагается, что в общем случае используемые значения будут последовательными при дополнении и отличающимися от ранее использованных значений, что удобно для человеческого восприятия.

4.6. Элемент <metadataDefs> для определения метаданных

Необязательный элемент <metadataDefs> в библиотечном документе содержит один или множество элементов <metadataDef>, определяющих метаданные.

Каждый элемент <metadataDef> **должен** содержать уникальное имя (NMTOKEN). Уникальность определяется среди всех метаданных в масштабе библиотечного документа и всех включённых в него (напрямую или косвенно) библиотечных документов. Элемент <metadataDef> **должен** также включать краткое описание (synopsis), значение тега, используемое для этих метаданных и информацию об определении типа значения. В качестве значений метаданных могут применяться лишь неделимые (atomic) типы данных. Элемент <metadataDef> **может** содержать элемент с подробным описанием.

Разрешены две формы определения типа. Первая форма использует элемент <typeRef> для указания имеющегося неделимого типа данных, определённого в элементе <dataTypeDefs> того же библиотечного документа или включённого в него документа. Использование элемента <typeRef> идентично использованию элементов <dataTypeDef>, за исключением того, что они могут указывать лишь неделимые типы. Последнее ограничение не проверяется схемой XML.

Вторая форма использует явное определение типа с помощью элемента <atomic>, который используется здесь так же, как в элементах <dataTypeDef>.

Ниже приведён пример использования обеих форм.

```
<metadataDefs>
 <metadataDef>
  <name>NEXTHOPID</name>
  <synopsis>Указывает элемент Next Hop в NH LFB</synopsis>
  <metadataID>17</metadataID>
  <typeRef>int32</typeRef>
 </metadataDef>
 <metadataDef>
  <name>CLASSID</name>
  <synopsis>Результат классификации (0 говорит о несоответствии).</synopsis>
  <metadataID>21</metadataID>
  <atomic>
   <baseType>int32</baseType>
   <specialValues>
    <specialValue value="0">
     <name>NOMATCH</name>
     <synopsis>Классификация не нашла соответствия.</synopsis>
    </specialValue>
   </specialValues>
  </atomic>
 </metadataDef>
</metadataDefs>
```

4.7. Элемент <LFBClassDefs> для определения классов LFB

Необязательный элемент <LFBClassDefs> можно использовать для определения одного или множества классов LFB с помощью элементов <LFBClassDef>. Каждый элемент <LFBClassDef> **должен** определять класс LFB и включать перечисленные ниже элементы.

- <name> указывает символьное имя класса LFB (например ipv4lpm).
- <synopsis> содержит краткое описание класса LFB (например, IPv4 Longest Prefix Match Lookup LFB).
- <version> указывает номер версии.
- <derivedFrom> указывает предка.
- <inputPorts> перечисляет входные порты и их спецификации.
- <outputPorts> перечисляет выходные порты и их спецификации.
- <components> определяет рабочие компоненты LFB.
- <capabilities> определяет компоненты возможностей (capability) LFB.
- <description> содержит операционную спецификацию LFB.
- Атрибут LFBClassID элемента LFBClassDef определяет идентификатор класса, который должен быть уникальным в глобальном масштабе.
- <events> определяет события, которые могут генерироваться экземплярами этого класса LFB.

Имена классов LFB должны быть уникальными, чтобы другие документы могли указывать классы по имени, а читатели могли понимать ссылки на классы по именам. Комплексное именование но возбраняется, но приветствуется простота. Как указано в разделе 9. Взаимодействие с IANA, агентство IANA поддерживает реестр имён и идентификаторов классов LFB со ссылками на определяющие классы документы.

Ниже приведена структура определения класса LFB. Отметим, что в целях упрощения схемы XML порядок размещения элементов в определении класса фиксирован. Имеющиеся в определении элементы должны размещаться в указанном ниже порядке.

```
<LFBClassDefs>
 <LFBClassDef LFBClassID="12345">
  <name>ipv4lpm</name>
  <synopsis>LFB поиска максимального размера соответствия префикса IPv4</synopsis>
  <version>1.0</version>
  <derivedFrom>baseclass</derivedFrom>

  <inputPorts>
   ...
  </inputPorts>

  <outputPorts>
   ...
  </outputPorts>

  <components>
   ...
  </components>

  <capabilities>
   ...
  </capabilities>

  <events>
   ...
  </events>

  <description>
   Этот класс LFB представляет операцию поиска префикса IPv4 с
   максимальной длиной совпадения.
   Моделируемым поведением является ...
  </description>

 </LFBClassDef>
 ...
</LFBClassDefs>
```

Отдельные компоненты и возможности имеют идентификаторы для использования протоколом ForCES. Эти componentID используются в структурах, подобно именам. Значения componentID для компонент и возможностей должны быть уникальными в рамках определения класса LFB.

Отметим, что элементы <name>, <synopsis> и <version> являются обязательными, а все остальные элементы могут отсутствовать в <LFBClassDef>. Однако при наличии необязательных элементов они должны размещаться в указанном выше порядке.

Атрибуты componentID для разных элементов в определении класса LFB (или компонент в структуре) **должны** различаться. Для них не требуется упорядоченности или последовательного выделения. Для удобства читателей и упрощения поддержки значения обычно выделяют последовательно, но этого не требует ни модель, ни протокол.

4.7.1. Элемент <derivedFrom> для выражения наследования LFB

Необязательный элемент <derivedFrom> может служить для указания того, что данный класс происходит от другого класса. Содержимым этого элемента **должно** быть уникальное имя (<name>) другого класса LFB, который **должен** быть определён в том же библиотечном документе или во включённом в него документе. При отсутствии элемента <derivedFrom> класс концептуально выводится из пустого базового класса.

Предполагается, что производный класс совместим со своим базовым классом. Производный класс **может** добавлять в родительский класс компоненты, но не может удалять имеющиеся компоненты. Это относится также к входным и выходным портам и возможностям.

4.7.2. Элемент <inputPorts> для определения входов LFB

Необязательный элемент <inputPorts> служит для определения входных портов. Класс LFB **может** не иметь входов или иметь один или более вход. Если у класса LFB нет входных портов, элемент <inputPorts> **должен** отсутствовать. Элемент <inputPorts> может содержать один или множество элементов <inputPort>, по одному для каждого порта или группы портов. Предполагается, что большинство LFB будет иметь один вход. Множество входов одного типа моделируется как входная группа. Группы определяются так же, как входные порты с помощью элемента <inputPort>, отличаясь лишь атрибутом group.

Следует избегать множества разнотипных портов (см. параграф 4.7.3). Некоторые LFB специального назначения совсем не имеют входов. Например, LFB генератора пакетов входы не требуются.

Отдельные входные порты и группы входных портов определяются элементами <inputPort> и различаются лишь необязательным атрибутом group.

Элемент <inputPort> **должен** включать перечисленные ниже элементы.

- <name> указывает символьное имя входного порта (например, in). Отметим, что имя должно быть уникальным в рамках класса LFB.
- <synopsis> содержит краткое описание входа (например, Normal packet input).

- <expectation> перечисляет все разрешённые форматы кадров (например, {ipv4 и ipv6}. Отметим, что в список следует включать имена, заданные в элементе <frameDefs> того же библиотечного документа или включённых в него документов. Элемент <expectation> может также содержать список требуемых метаданных (например, {classid, vpnid}. В список следует включать имена, заданные в элементе <metadataDefs> того же библиотечного документа или включённых в него документов. Для каждого элемента метаданных должно быть указано, является он обязательным или опциональным. Для необязательных метаданных должно быть указано значение, которое LFB будет использовать при отсутствии соответствующих метаданных у пакета.

В дополнение к этому необязательный атрибут group в элементе <inputPort> может указывать, что порт может вести себя как группа портов, т. е позволяет создавать экземпляры. Это указывается значением атрибута true (по умолчанию создание экземпляров запрещено - false).

Пример элемента <inputPorts>, определяющего два входных порта, из которых второй является группой, показан ниже.

```
<inputPorts>
 <inputPort>
  <name>in</name>
  <synopsis>Обычный вход</synopsis>
  <expectation>
   <frameExpected>
    <ref>ipv4</ref>
    <ref>ipv6</ref>
   </frameExpected>
   <metadataExpected>
    <ref>classid</ref>
    <ref>vifid</ref>
    <ref dependency="optional" defaultValue="0">vrfid</ref>
   </metadataExpected>
  </expectation>
 </inputPort>
 <inputPort group="true">
  ... другой входной порт ...
 </inputPort>
</inputPorts>
```

Для каждого элемента <inputPort> ожидаемые типы кадров определяются элементом <frameExpected> с одним или несколькими элементами <ref> (см. пример выше). При указании нескольких типов кадров ожидается один из перечисленных. Пакеты всех остальных типов считаются несовместимыми с этим входным портом в данном классе LFB. В приведённом выше примере ожидаемыми типами кадров являются ipv4 и ipv6.

Ожидаемые метаданные указываются элементом <metadataExpected>. В простейшей форме этот элемент может содержать список элементов <ref>, каждый из которых указывает элемент метаданных. При указании множества экземпляров метаданных в элементах <ref> это означает, что каждый принятый пакет должен сопровождаться **всеми** этими метаданными (за исключением метаданных, помеченных как optional в атрибуте dependency соответствующего элемента <ref>). Для необязательных (optional) метаданных **должно** указываться принятое по умолчанию значение с помощью атрибута defaultValue. В приведённом выше примере указаны три ожидаемых элемента метаданных, два из которых обязательны (classid и vifid), а один не обязателен (vrfid).

Схема также позволяет более сложные определения ожидаемых метаданных. Например, с помощью элемента <one-of> можно указать список метаданных, из которого с пакетом должен быть представлен один элемент. Например,

```
<metadataExpected>
 <one-of>
  <ref>prefixmask</ref>
  <ref>prefixlen</ref>
 </one-of>
</metadataExpected>
```

Здесь метаданные prefixmaskили prefixlen должны сопровождать каждый пакет.

Две формы указания метаданных можно комбинировать, как показано ниже.

```
<metadataExpected>
 <ref>classid</ref>
 <ref>vifid</ref>
 <ref dependency="optional" defaultValue="0">vrfid</ref>
 <one-of>
  <ref>prefixmask</ref>
  <ref>prefixlen</ref>
 </one-of>
</metadataExpected>
```

Хотя схема позволяет определять и более сложные конструкции для ожидаемых метаданных, здесь они не рассматриваются.

4.7.3. Элемент <outputPorts> для определения выходов LFB

Необязательный элемент <outputPorts> служит для определения выходных портов. Класс LFB **может** совсем не иметь выходов или содержать один или несколько выходных портов. Если у класса LFB нет выходных портов, элемент <outputPorts> **должен** отсутствовать. Элемент <outputPorts> **должен** включать один или несколько элементов <outputPort>, по одному для каждого порта или группы портов. При наличии множества однотипных выходов они моделируются как группа выходных портов. Некоторые LFB специального назначения не имеют выходных портов (например, Dropper).

Одиночные выходные порты и отдельные группы определяются элементами <outputPort> и различаются лишь необязательным атрибутом group.

Элемент <outputPort> **должен** содержать перечисленные ниже элементы.

- <name> указывает символьное имя выходного порта (например, out). Отметим, что имя должно быть уникальным в рамках класса LFB.
- <synopsis> содержит краткое описание входа (например, Normal packet output).
- <product> перечисляет все разрешённые форматы кадров (например, {ipv4 и ipv6}. Отметим, что в список следует включать имена, заданные в элементе <frameDefs> того же библиотечного документа или включённых в него документов. Элемент <product> **может** также содержать список выдаваемых (генерируемых) метаданных (например, {classid, color}. В список следует включать имена, заданные в элементе <metadataDefs> того же библиотечного документа или включённых в него документов. Для каждого генерируемого элемента метаданных следует указать, создаётся он всегда или при заданных условиях. Эта информация нужна при оценке совместимости LFB.

В дополнение к этому необязательный атрибут group в элементе <outputPort> может указывать, что порт может вести себя как группа портов, т. е позволяет создавать экземпляры. Это указывается значением атрибута true (по умолчанию создание экземпляров запрещено - false).

Пример элемента <outputPort>, определяющего 2 выходных порта, из которых второй является группой, показан ниже.

```
<outputPorts>
 <outputPort>
  <name>out</name>
  <synopsis>Обычный вывод</synopsis>
  <product>
   <frameProduced>
    <ref>ipv4</ref>
    <ref>ipv4bis</ref>
   </frameProduced>
   <metadataProduced>
    <ref>nhid</ref>
    <ref>nhtabid</ref>
   </metadataProduced>
  </product>
 </outputPort>
 <outputPort group="true">
  <name>exc</name>
  <synopsis>Группа портов для вывода в исключительных случаях</synopsis>
  <product>
   <frameProduced>
    <ref>ipv4</ref>
    <ref>ipv4bis</ref>
   </frameProduced>
   <metadataProduced>
    <ref availability="conditional">errorid</ref>
   </metadataProduced>
  </product>
 </outputPort>
</outputPorts>
```

Типы кадров и метаданных, выдаваемых портом определяются в элементе <product> каждого <outputPort>. Внутри <product> список производимых портом типов кадров указывается в элементе <frameProduced>. При указании нескольких типов это значит, что порт будет выдавать один из указанных типов.

Список метаданных, выдаваемых портом указывается в необязательном элементе <metadataProduced> внутри <product>. В простейшей форме этот элемент содержит список элементов <ref>, каждый из которых задаёт тип метаданных. При наличии списка все метаданные из него выдаются с каждым пакетом (за исключением помеченных необязательным атрибутом availability со значением conditional). Подобно элементу <metadataExpected> в <inputPort>, элемент <metadataProduced> поддерживает более сложные формы, которые здесь не рассматриваются.

4.7.4. Элемент <components> для определения рабочих компонент LFB

Рабочие параметры LFB, которые должны быть видимыми для CE, собираются в модели как компоненты LFB. Это включает, например, флаги, аргументы с одним параметром, составные аргументы и таблицы. Отметим, что компонентами здесь называются лишь рабочие параметры LFB, которые должны быть видимыми для CE. Другие переменные, которые являются внутренними для реализации LFB, не считаются компонентами LFB и не указываются здесь.

Ниже перечислены некоторые из компонент LFB.

- Настраиваемые флаги и переключатели рабочих режимов LFB.
- Число входов или выходов в группе портов.
- Настраиваемые таблицы поиска, включая таблицы интерфейсов, префиксов, классификации, отображения DSCP, MAC-адресов и т. п.
- Счётчики пакетов и байтов.
- Счётчики событий.
- Текущее число входов или выходов для каждой группы входов или выходов.

Модель ForCES поддерживает определение ограничений доступа, указывающих, что может делать CE с компонентами LFB. Ниже перечислены поддерживаемые моделью категории доступа.

- No-access - нет доступа. Применяется для полноты и позволяет определять объекты, используемые другими, но не указываемые непосредственно элементами управления CE. Это полезно также для FE, сообщающих, что для некоторых определённых и обычно доступных компонент не поддерживается доступ со стороны CE.

- Read-only - только чтение.
- Read-write - чтение и запись.
- Write-only - только запись. Это могут быть любые настраиваемые данные, чтение которых не разрешено для CE (например, ключи защиты).
- Read-reset - чтение и сброс. CE может считывать и сбрасывать этот ресурс, но не может установить для него произвольное значение. Примером могут служить счётчики.
- Firing-only - инициирование. Попытка записи в этот ресурс будет вызывать в LFB конкретные действия, но записанное значение игнорируется.

Класс LFB **должен** определять для данной компоненты лишь один из возможных режимов доступа.

Компоненты класса LFB перечисляются в элементе <components>, где для каждой компоненты используется элемент <component>, содержащий все или некоторые из перечисленных ниже элементов, часть которых обязательна.

- <name> определяет имя компоненты, которое **должно** присутствовать. Имя должно быть уникальным среди компонент класса LFB (например, version).
- <synopsis> обеспечивает краткое описание назначения компоненты и является обязательным.
- <optional/> является необязательным элементом, наличие которого указывает необязательность компоненты.
- Тип данных компоненты может быть указан ссылкой на предопределённый тип данных или локальным определением типа. Ссылка на определение задаётся с помощью элемента <typeRef>, который должен указывать уникальное имя имеющегося типа данных, который определён элементом <dataTypeDefs> в этом же библиотечном документе или в любом из включённых в него документов. При локальном определении данных (безымянный тип) используется один из элементов <atomic>, <array>, <struct> или <union>. Их использование аналогично использованию внутри элементов <dataTypeDef> (параграф 4.5). В определении компоненты **должна** быть включена та или иная форма определения типа данных.
- <defaultValue> является необязательным элементов и при наличии указывает принятое по умолчанию значение компоненты. Если принятое по умолчанию значение задано, элемент FE должен обеспечить установку этого значения компоненты при инициализации или сбросе LFB. Если принятое по умолчанию значение для компоненты не задано, элементу CE **недопустимо** принимать какие-либо допущения о значении компоненты при инициализации. Элемент CE должен считать или установить значение, если хочет знать его.
- <description> **может** включаться в определение для более детального описания назначения или использования определяемой компоненты.

Элемент <component> **должен** иметь атрибут componentID, числовое значение которого используется протоколом ForCES.

В дополнение к перечисленным элементам <component> включает необязательный атрибут access, который может принимать значение read-only, read-write (используется по умолчанию), write-only, read-reset или trigger-only.

Поддержка необязательных компонент и возможность реальной записи в компоненты read-write может быть определена для данного экземпляра LFB элементом CE путём считывания информации о свойствах компоненты. Установка режима доступа trigger-only означает включение компоненты для использования лишь в качестве детектора событий.

Приведённый ниже пример определяет две компоненты для LFB.

```
<components>
 <component access="read-only" componentID="1">
  <name>foo</name>
  <synopsis>число объектов</synopsis>
  <typeRef>uint32</typeRef>
 </component>
 <component access="read-write" componentID="2">
  <name>bar</name>
  <synopsis>число других объектов</synopsis>
  <atomic>
   <baseType>uint32</baseType>
   <rangeRestriction>
    <allowedRange min="10" max="2000"/>
   </rangeRestriction>
  </atomic>
  <defaultValue>10</defaultValue>
 </component>
</components>
```

Первая компонента (foo) является доступным лишь для чтения 32-битовым целым числом без знака, определяемым ссылкой на встроенный неделимый тип uint32. Вторая компонента (bar) также является целым числом, но использует элемент <atomic> для задания дополнительных ограничений диапазона. Эта компонента доступна для чтения и записи. По умолчанию для неё. задано значение 10. Хотя доступ разрешён для чтения и записи, некоторые реализации **могут** дополнительно ограничивать доступ и это будет указано в свойствах компоненты.

Отметим, что не все компоненты обязательно присутствуют в каждой реализации. Хотя возможности зачастую указывают отсутствие, элементы CE могут ссылаться на отсутствующие или неразрешенные компоненты. Следует предусматривать механизмы протокола ForCES для индикации таких ошибок.

Определённый выше механизм для неподдерживаемых компонент может также применяться для ссылок на отсутствующие элементы массива или попытки записи в компоненты, для которых разрешено лишь чтение.

4.7.5. Элемент <capabilities> для определения возможностей LFB

Спецификация класса LFB обеспечивает некоторую гибкость для реализации FE в части использования класса LFB. Например, экземпляр может иметь некоторые ограничения, которые не наследуются из определения класса, а внесены реализацией. Некоторые из таких ограничений извлекаются из информации о свойствах компонент LFB. Модель позволяет указывать дополнительную информацию о возможностях.

Такая информация выражается компонентами возможностей класса LFB, которые всегда являются доступными лишь для чтения атрибутами и указываются в отдельном элементе <capabilities> определения <LFBClassDef>. Элемент <capabilities> содержит один или множество элементов <capability>, каждый из которых определяет одну возможность. Формат элемента <capability> похож на формат <component> и отличается лишь отсутствием атрибута режима доступа (всегда read-only) и элемента <defaultValue> (принятое по умолчанию значение не применимо к атрибутам read-only).

Ниже приведены некоторые примеры возможностей.

- Версия класса LFB с которой был собран экземпляр LFB.
- Поддерживаемые необязательные возможности класса LFB.
- Максимальное число настраиваемых выходов в выходной группе.
- Ограничения LFB на передачу метаданных.
- Дополнительные ограничения диапазона рабочих компонент.

Ниже приведён пример с двумя атрибутами возможностей.

```
<capabilities>
 <capability componentID="3">
  <name>version</name>
  <synopsis>
   Версия класса LFB, которой соответствует экземпляр.
  </synopsis>
  <typeRef>version</typeRef>
 </capability>
 <capability componentID="4">
  <name>limitBar</name>
  <synopsis>
   Максимальное значение атрибута bar.
  </synopsis>
  <typeRef>uint16</typeRef>
 </capability>
</capabilities>
```

4.7.6. Элемент <events> для генерации уведомлений LFB

Элемент <events> содержит информацию о событиях, для которых экземпляры этого класса LFB могут генерировать уведомления для CE. Общее описание объявления и работы с событиями LFB приведено в параграфе 3.2.5.

Элемент <events> может (но не обязан) содержать элементы <event>, каждый из которых описывает одно событие и имеет атрибут eventID с уникальным (для класса LFB) идентификатором события и включает элементы:

- <eventTarget>, указывающий поле (компоненту) LFB, проверяемую для генерации события;
- <condition>, указывающий для поля условие генерации события из списка определённых условий;
- <eventReports>, указывает значения, включаемые в уведомление о событии.

В приведённом ниже примере показано несколько разных конструкций.

Элемент <events> имеет атрибут baseID, который обычно имеет форму <events baseID="number">. Значением baseID является стартовый идентификатор componentID для пути, указывающего события. Он должен отличаться от componentID всех компонент верхнего уровня (включая возможности) класса LFB. В производных LFB (т. е. имеющих элемент <derivedFrom>), у которых родительский класс LFB имеет объявления для событий, значению baseID недопустимо присутствовать в элементе <events> производного LFB и взамен используется значение baseID из родительского класса LFB. В приведённом примере baseID имеет значение 7.

```
<events baseID="7">
 <event eventID="7">
   <name>Foochanged</name>
   <synopsis>Пример события для скаляра</synopsis>
   <eventTarget>
     <eventField>foo</eventField>
   </eventTarget>
   <eventChanged/>
   <eventReports>
     <!-- Отчёт о новом состоянии -->
     <eventReport>
       <eventField>foo</eventField>
     </eventReport>
   </eventReports>
 </event>

 <event eventID="8">
   <name>Goof1changed</name>
   <synopsis>
       Пример события для составной структуры
   </synopsis>
   <eventTarget>
```

```
        <!-- Целью служит goo.f1 -->
        <eventField>goo</eventField>
        <eventField>f1</eventField>
      </eventTarget>
      <eventChanged/>
      <eventReports>
        <!-- Отчёт о новом состоянии goo.f1 -->
        <eventReport>
        <eventField>goo</eventField>
        <eventField>f1</eventField>
        </eventReport>
      </eventReports>
    </event>

    <event eventID="9">
      <name>NewbarEntry</name>
      <synopsis>
          Событие для новой записи, созданной в таблице bar
      </synopsis>
      <eventTarget>
        <eventField>bar</eventField>
        <eventSubscript>_barIndex_</eventSubscript>
      </eventTarget>
      <eventCreated/>
      <eventReports>
        <eventReport>
         <eventField>bar</eventField>
         <eventSubscript>_barIndex_</eventSubscript>
       </eventReport>
       <eventReport>
        <eventField>foo</eventField>
       </eventReport>
      </eventReports>
    </event>

    <event eventID="10">
      <name>Gah11changed</name>
      <synopsis>
          Событие для таблицы gah изменена запись с индексом 11
      </synopsis>
      <eventTarget>
        <eventField>gah</eventField>
        <eventSubscript>11</eventSubscript>
      </eventTarget>
      <eventChanged/>
      <eventReports>
        <eventReport>
         <eventField>gah</eventField>
         <eventSubscript>11</eventSubscript>
       </eventReport>
      </eventReports>
    </event>

    <event eventID="11">
      <name>Gah10field1</name>
      <synopsis>
          Событие для таблицы gah изменена запись с индексом 10,
          колонка field1
      </synopsis>
      <eventTarget>
        <eventField>gah</eventField>
        <eventSubscript>10</eventSubscript>
        <eventField>field1</eventField>
      </eventTarget>
      <eventChanged/>
      <eventReports>
        <eventReport>
         <eventField>gah</eventField>
         <eventSubscript>10</eventSubscript>
        </eventReport>
      </eventReports>
    </event>
  </events>
```

4.7.6.1. Элемент <eventTarget>

Элемент <eventTarget> содержит информацию, указывающую поле в LFB, которое отслеживается для событий.

Элемент <eventTarget> содержит одно или множество элементов <eventField, за каждым из которых может следовать один или множество элементов <eventSubscript>. Каждый из этих двух элементов представляет текстовый эквивалент компоненты выбора пути в LFB.

Элемент <eventField> содержит имя компоненты в LFB или компоненты, встроенной в массив или структуру внутри LFB. Имя, используемое в <eventField>, **должно** указывать действительную компоненту в содержащем LFB контексте. Первым в элементе <eventTarget> **должен** быть элемент <eventField>. В приведённом ниже примере в качестве <eventField> используются 4 компоненты LFB - foo, goo, bar и gah.

В простом случае <eventField> указывает компоненту atomic. Этот случай показан в событии Foochanged. Элемент <eventField> служит также для указания сложных компонент, таких как массивы или структуры.

Определённое в примере первым событие Foochanged показывает, как скалярная компонента LFB (foo) может отслеживаться для инициирования событий.

Второе событие Goof1changed показывает отслеживание элемент составной структуры для генерации события.

События NewbarEntry, Gah11changed и Gah10field1 представляют мониторинг массивов bar и gah с разными деталями.

Если <eventField> указывает составную компоненту, **может** применяться дополнительный элемент <eventField> для уточнения пути к целевому элементу. Событие Goof1changed показывает применение второго <eventField> для указания на элемент f1 в структуре goo.

Если <eventField> указывает массив, применимы приведённые ниже правила.

- Элемент <eventSubscript> **должен** быть представлен как следующий элемент XML после <eventField>, указывающего элемент массива. Элементу <eventSubscript> **недопустимо** появляться иначе как после ссылки на массив, поскольку он имеет смысл только в этом контексте.
- Элемент <eventSubscript> содержит один из двух приведённых ниже вариантов.
  - Численное значение для указания применимости события к конкретному элементу массива (по индексу). Например, событие Gah11changed задаёт мониторинг элемента таблицы gah с индексом 11.
  - Предполагается, что наиболее вероятным будет использование событий, определённых для всех элементов массива (т. е. шаблон для всех значений индекса). В этом случае значением <eventSubscript> **должно** быть имя, а не число. Это же имя может использоваться в качестве значения <eventSubscript> в элементах <eventReport>, как описано ниже. Пример шаблонного индекса таблицы показан в событии NewBarentry, где значение <eventSubscript> названо _barIndex_.
- За элементом <eventField> **может** следовать <eventSubscript> для уточнения пути к целевому элементу (это похоже на использование <eventField> для уточнения пути в составной структуре, показанное в событии Goof1changed). Событие Gah10field1 в примере отслеживает изменения столбца field1 в таблице gah.

Следует подчеркнуть, что элемент <eventSubscript> в событии NewbarEntry не является именем компоненты. Это имя переменной для использования в элементах <eventReport> (параграф 4.7.6.3) определения данного класса LFB. Это имя **должно** отличаться от всех имён компонент, которые допустимы в <eventReport>.

4.7.6.2. Элемент <eventCondition>

Элемент условия для событий указывает, при каких условиях создаются уведомления. Список условий приведён ниже.

<eventCreated/>
Целью должен быть массив, завершающийся указанным индексом. Событие генерируется при создании элемента в массиве, даже если это происходит по указанию CE. Условие <eventCreated/> показано в примере NewbarEntry.

<eventDeleted/>
Целью должен быть массив, завершающийся указанным индексом. Событие генерируется при уничтожении элемента в массиве, даже если это происходит по указанию CE.

<eventChanged/>
Событие генерируется при любом изменении целевой компоненты. Для двоичных компонент, таких как включение/выключение (up/down) это отражает смену состояния. Условие может применяться и с цифровыми атрибутами, когда триггером служит смена значения. Условие <eventChanged/> показано в примерах Foochanged, Gah11changed и Gah10field1.

<eventGreaterThan/>
Событие генерируется при превышении целевой компонентой порогового значения, являющегося свойством события.

<eventLessThan/>
Событие генерируется, когда целевая компонента становится меньше порогового значения, являющегося свойством события.

4.7.6.3. Элемент <eventReports>

Элемент <eventReports> в <event> заявляет информацию, доставляемую элементом FE с уведомлением о событии.

Элемент <eventReports> содержит один или несколько элементов <eventReport>, содержащих части данных, сообщаемых классом LFB. Уведомление переносит данные как набор элементов <eventReport>, определённых в структуре. Синтаксис совпадает с синтаксисом <eventTarget> и использует элементы <eventField> и <eventSubscript> с применением тех же правил. Каждый элемент <eventReport> **должен** указывать компоненту класса LFB. Элементы <eventSubcript> **могут** содержать целые числа. Если они содержат имена, это **должны** быть имена из элементов <eventSubscript> цели события <eventTarget>. При выборе используется значение индекса, который указывает конкретный элемент, вызвавший событие. Это может служить для указания компоненты, вызвавшей событие, или ссылки на связанную информацию в параллельных таблицах.

В приведённом примере для события Foochanged отчёт будет содержать значение foo. Для события NewbarEntry, связанного с компонентой LFB bar, которая является массивом, будут сообщаться два элемента, указанные двумя объявлениями <eventReport>.

- Первый элемент <eventReport> указывает добавление новой записи в таблицу bar. Напомним, что _barIndex_ объявлен как <eventTarget> <eventSubcript> для события и можно использовать имя вместо числа. Элемент <eventSubcript> предполагается шаблоном, которому соответствует любой индекс новой записи.
- Второй элемент <eventReport> включает значение компоненты LFB foo в момент создания новой записи в bar. Указание foo в этом случае приведено для демонстрации гибкости отчётов о событиях.

Структура отчётов о события создана для того, чтобы позволить разработчикам LFB задавать информацию, которая вероятно не известна заранее CE и вероятно потребуется CE для обработки события. Хотя структура позволяет указывать большие блоки информации (массив или составная структура целиком), делать это не рекомендуется. Кроме того, переменная ссылки/подписки в отчёте содержит лишь малую часть связанной с событием информации. Например, цепочки через поля индексов в таблице не поддерживаются. В общем случае механизм <eventReports> является оптимизацией для базовых ситуаций, позволяющей CE избавиться от необходимости запрашивать информацию, требуемую для того, чтобы понять событие. Он не представляет всю возможную информацию.

Если какая-либо из компонент, указанных eventReport, является необязательной, отчёт **должен** использовать формат протокола, поддерживающий необязательные элементы и разрешающий им отсутствовать. Отсутствующие компоненты не указываются в отчёте.

4.7.6.4. Управление событиями во время работы

Базовый обзор объявления событий LFB и работы с ними приведён в параграфе 3.2.5.

Элемент <eventTarget> обеспечивает дополнительные компоненты используемые в пути для ссылки на событие. Путь содержит baseID для событий, далее следует идентификатор конкретного события, а затем значения для каждого элемента <eventSubscript>, если они имеются в <eventTarget>.

Путь к событию будет однозначно указывать конкретный факт события в уведомлении о событии для CE. В приведённом выше примере (конец параграфа 4.7.6) уведомление с путём 7.7 однозначно указывает событие, вызванное изменением foo, а путь 7.9.100 однозначно указывает событие, вызванное созданием записи в таблице bar с индексом 100.

Как описано в параграфе 4.8.5, с элементами событий связаны определённые свойства. Эти свойства включают информацию о подписке, указывающую элементы CE, желающие получать уведомления о событиях FE для события в целом, пороги, связанные с пересечением уровней, и условия фильтрации, которые могут снизить число уведомлений, генерируемых FE. Детали применимых условий фильтрации рассмотрены в этом параграфе. Условия фильтрации позволяют элементу FE предотвратить лавинную рассылку уведомлений, которая может возникать при колебаниях около порога условия. Для FE, не желающих поддерживать фильтрацию, свойства фильтра могут быть доступны лишь для чтения или не поддерживаться совсем.

В дополнение к идентификации источников событий CE использует путь к событиям для активации во время работы элементов управления на основании свойств событий (параграф 4.8.5) с использованием операции SET-PROP, определённой в протоколе ForCES [RFC5810].

Для того, чтобы активизировать генерацию событий на FE устройство CE передаёт FE сообщение SET-PROP, указывающее событие и его регистрационное свойство с любым префиксом пути к событию. Таким образом, для события NewbarEntry, определённого выше сообщение SET-PROP с путём 7.9 будет подписывать CE на все факты событий, связанных с любой записью в таблице bar. Это особенно полезно для условий <eventCreated/> и <eventDestroyed/> в таблицах. События, использующие эти условия, обычно будут определяться с последовательностью полей/индексов, указывающей массив и завершающейся элементом <eventSubscript>. Таким образом, уведомление о событии будет указывать созданный или уничтоженный элемент массива. Обычно подписка будет выполняться для массива, а не для его конкретного элемента, поэтому будет применяться более короткий путь.

В приведённом примере подписка 7.8 предполагает получение всех объявленных событий из таблицы bar, подписка 7.8.100 означает получение уведомлений о создании в таблице записи с индексом 100.

Пороги и условия фильтрации могут применяться лишь к отдельным событиям. Для событий, определённых на элементах массива, данная спецификация не позволяет задавать порог или условие фильтрации на всех элементах массива.

4.7.7. Элемент <description> для рабочей спецификации LFB

Элемент <description> в <LFBClass> содержит неструктурированный (в смысле XML) текст, описывающий LFB для человека.

4.8. Свойства

Компоненты LFB имеют свойства, которые важны для CE. Наиболее важными свойствами являются существование и возможности чтения и записи элементов. В зависимости от типа компоненты важными могут быть и другие свойства.

Модель обеспечивает определение структуры информации о свойствах, для чего имеется базовый класс. Для массивов, псевдонимов и событий имеются субклассы информации о свойствах с дополнительными полями. Эта информация доступна CE (и обновляется, когда это применимо) по протоколу ForCES. Хотя некоторая информация о свойствах открыта для записи, в настоящее время нет механизма проверки свойств. Доступность для записи можно проверить лишь попыткой изменить значение.

4.8.1. Базовые свойства

Определение базовых свойств вместе со скаляром dataTypeDef для доступности приведено ниже. Отметим, что доступ к информации о свойствах обычно разрешён лишь в режиме read-only.

```
<dataTypeDef>
  <name>accessPermissionValues</name>
  <synopsis>
    Возможные значения прав доступа к компоненте
  </synopsis>
  <atomic>
    <baseType>uchar</baseType>
    <specialValues>
      <specialValue value="0">
        <name>None</name>
```

```
                <synopsis>Доступ запрещён</synopsis>
              </specialValue>
               <specialValue value="1">
                <name> Read-Only </name>
                <synopsis>Доступ только для чтения</synopsis>
              </specialValue>
              <specialValue value="2">
                <name>Write-Only</name>
                <synopsis>
                  Компонента МОЖЕТ записываться, но чтение недоступно
                </synopsis>
              </specialValue>
              <specialValue value="3">
                <name>Read-Write</name>
                <synopsis>
                  ВОЗМОЖНО чтение и запись компоненты
                </synopsis>
              </specialValue>
            </specialValues>
          </atomic>
        </dataTypeDef>
        <dataTypeDef>
          <name>baseElementProperties</name>
          <synopsis>Базовые свойства, доступность</synopsis>
          <struct>
            <component componentID="1">
              <name>accessibility</name>
              <synopsis>
                  Компоненты не существует, чтение и
                  запись невозможны
              </synopsis>
              <typeRef>accessPermissionValues</typeRef>
            </component>
          </struct>
        </dataTypeDef>
```

4.8.2. Свойства массива

Свойства массива содержат много дополнительной информации и доступны лишь для чтения.

```
<dataTypeDef>
  <name>arrayElementProperties</name>
  <synopsis>Определение свойств элементов массива</synopsis>
  <struct>
    <derivedFrom>baseElementProperties</derivedFrom>
    <component componentID="2">
      <name>entryCount</name>
      <synopsis>Число элементов в массиве</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>highestUsedSubscript</name>
      <synopsis>Последний использованный индекс в массиве</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="4">
      <name>firstUnusedSubscript</name>
      <synopsis>
        Индекс первого неиспользуемого элемента массива
      </synopsis>
      <typeRef>uint32</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

4.8.3. Свойства строки

Свойства строки задают реальный размер в октетах и максимальную длину элемента. Максимальный размер включён потому, что реализации FE **могут** ограничивать размер строк сверх заданного классом LFB предела.

```
<dataTypeDef>
  <name>stringElementProperties</name>
  <synopsis>Определение свойств строкового элемента</synopsis>
  <struct>
    <derivedFrom>baseElementProperties</derivedFrom>
    <component componentID="2">
      <name>stringLength</name>
      <synopsis>Число октетов в строке</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>maxStringLength</name>
      <synopsis>
        Максимальное число октетов в строке
        </synopsis>
      <typeRef>uint32</typeRef>
    </component>
  </struct>
```

```
</dataTypeDef>
```

4.8.4. Свойства строки октетов

Свойства octetstring задают реальный и максимальный размер, поскольку реализации FE **могут** ограничивать сверх заданного определением класса LFB предела.

```
<dataTypeDef>
  <name>octetstringElementProperties</name>
  <synopsis>Определение свойств элемента octetstring
  </synopsis>
  <struct>
    <derivedFrom>baseElementProperties</derivedFrom>
    <component componentID="2">
      <name>octetstringLength</name>
      <synopsis>Число октетов в octetstring</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>maxOctetstringLength</name>
      <synopsis>
        Максимальное число октетов в octetstring
      </synopsis>
      <typeRef>uint32</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

4.8.5. Свойства событий

Свойства событий (обычно) добавляют 3 доступных для записи поля. Одно из них является полем подписки, значение 0 в котором указывает, что уведомление не создаётся. Отличное от 0 значение (обычно 1) задаёт генерацию уведомлений. Поле гистерезиса служит для подавления уведомлений в результате колебаний около условного значения, как описано ниже (параграф 4.8.5.2). Поле порога используется в условиях <eventGreaterThan/> и <eventLessThan/>, задавая значение для сравнений с целью события. Использование свойства позволяет CE задать интересующий уровень. Элементы FE, не поддерживающие порогов для событий, будут делать поле read-only.

```
<dataTypeDef>
  <name>eventElementProperties</name>
  <synopsis>Определение свойств события</synopsis>
  <struct>
    <derivedFrom>baseElementProperties</derivedFrom>
    <component componentID="2">
      <name>registration</name>
      <synopsis>
        Имеется CE, зарегистрированный для уведомлений о событии
      </synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>threshold</name>
      <synopsis>Значение для сравнения в условии</synopsis>
      <optional/>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="4">
      <name>eventHysteresis</name>
      <synopsis>Зона подавления рекурсивных уведомлений</synopsis>
      <optional/>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="5">
      <name>eventCount</name>
      <synopsis>Число уведомлений для подавления</synopsis>
      <optional/>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="6">
      <name>eventInterval</name>
      <synopsis>Интервал между уведомлениями в мсек</synopsis>
      <optional/>
      <typeRef>uint32</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

4.8.5.1. Базовая фильтрация событий

Свойства событий имеют значения для управления некоторыми условиями фильтрации. Поддержка этих условий не обязательна, но все условия **следует** поддерживать. События, о которых достоверно известна невозможность высокой частоты или иные проблемы, **могут** не поддерживать фильтров.

В настоящее время определены три разных переменных для условий фильтрации - eventCount, eventInterval и eventHysteresis. Установка значения 0 (принято по умолчанию) отключает проверку соответствующего условия.

Концептуально при возникновении события проверяются все настроенные условия. Если ни один из фильтров не сработал или не возникло иных препятствующих условий, уведомление о событии создаётся. Если задана фильтрация и условия фильтра не выполняются, уведомление не создаётся. Для условий фильтрации определён сброс при

генерации уведомлений. Если любое условие выполнено и уведомление создано, сброс выполняется для всех условий, даже если они не выполнены. Это обеспечивает точное определения взаимодействия различных условий .

Примером взаимодействия условий может служить событие с eventCount = 5 и eventInterval = 500 мсек. Предположим, что элемент FE обнаружил всплеск генерации таких уведомлений. Первое событие вызвало передачу уведомления CE. Затем, если быстро (менее 0,5 сек) произошли ещё 4 события, они не вызовут уведомлений. Если обнаружены 2 дополнительных события, второе приведёт к передаче уведомления. Если прошло более 500 мсек после уведомления и обнаружено событие, это приведёт к передаче уведомления. В любом случае счётчик и временной интервал подавления сбрасываются, независимо от условия, вызвавшего генерацию уведомления.

4.8.5.2. Фильтрация гистерезиса событий

События с численными условиями могут иметь фильтр гистерезиса. Уровень гистерезиса определяется свойством события. Это позволяет элементу FE уведомлять CE о применении гистерезиса и в случае выбора FE может разрешать CE менять гистерезис. Это применимо для <eventChanged/> с численным полем, а также для <eventGreaterThan/> и <eventLessThan/>. Содержимым элемента <variance> является число. При поддержке гистерезиса элемент FE **должен** отслеживать значение элемента и гарантировать, что условие не выполняется по крайней мере в части гистерезиса из свойства события. Для гистерезиса V выполняется указанное ниже.

- Если при условии <eventChanged/> последнее уведомление было для значения X, то уведомление <changed/> **недопустимо** создавать, пока значение не достигнет X +/- V.
- Для условия <eventGreaterThan/> с порогом T после генерации хотя бы одного уведомления **недопустимо** создавать новые, пока поле сначала не станет меньше или равно T - V, а затем снова превысит T.
- Для условия <eventLessThan/> с порогом T после генерации хотя бы одного уведомления **недопустимо** создавать новые, пока поле сначала не станет больше или равно T + V, а затем снова меньше T.

4.8.5.3. Фильтрация счета событий

В событиях **может** устанавливаться фильтр по числу. Это свойство при отличном от 0 значении указывает число событий, по достижении которого события следует считать избыточными и прекратить создание уведомлений для них. Такие образом, при значении свойства 1 и отсутствии других условий, каждое второе событие не будет приводить к созданию уведомления.

Концептуальной реализацией (не обязательной) этого фильтра может быть внутренний счётчик подавления. Всякий раз при возникновении события значение счётчика проверяет и при нулевом значении генерируется уведомление. Счётчик инкрементируется при каждом событии независимо от передачи уведомления и при достижении заданного значения сбрасывается в 0.

4.8.5.4. Фильтрация времени событий

Для событий **может** устанавливаться фильтрация по времени. Это свойство представляется минимальным временным интервалом (в отсутствие других фильтров) между генерацией уведомлений о последовательных событиях. Это условие **должно** выполняться лишь в том случае, когда с момента генерации последнего уведомления прошло время, превышающее значение параметра фильтра.

Концептуально этот фильтр можно представить как сравнение сохранённой временной метки с моментом обнаружения события или как таймер, сбрасывающий флаг подавления. В любом случае если уведомление было создано по любому условию, временной интервал **должен** отсчитываться заново.

4.8.6. Свойства псевдонимов

В свойства для псевдонимов (обычно) добавляются три открытых для записи поля, которые объединяются для идентификации целевой компоненты, на которую указывает псевдоним.

```
<dataTypeDef>
  <name>aliasElementProperties</name>
  <synopsis>Определение свойств псевдонима</synopsis>
  <struct>
    <derivedFrom>baseElementProperties</derivedFrom>
    <component componentID="2">
      <name>targetLFBClass</name>
      <synopsis>Идентификатор класса для цели псевдонима</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>targetLFBInstance</name>
      <synopsis>Идентификатор экземпляра для цели псевдонима</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="4">
      <name>targetComponentPath</name>
      <synopsis>
        Путь к составной цели. Каждые 4 элемента
        считываются как 1 элемент пути с использованием
        конструкции path протокола ForCES [RFC5810].
      </synopsis>
      <typeRef>octetstring[128]</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

4.9. Схема XML для документов LFB Class Library

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
     xmlns:lfb="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
     targetNamespace="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
     attributeFormDefault="unqualified"
     elementFormDefault="qualified">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Schema for Defining LFB Classes and associated types (frames,
      data types for LFB attributes, and metadata).
      </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="description" type="xsd:string"/>
    <xsd:element name="synopsis" type="xsd:string"/>
    <!-- Корневой элемент документа: LFBLibrary -->
    <xsd:element name="LFBLibrary">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="description" minOccurs="0"/>
          <xsd:element name="load" type="loadType" minOccurs="0"
                       maxOccurs="unbounded"/>
        <xsd:element name="frameDefs" type="frameDefsType"
                       minOccurs="0"/>
        <xsd:element name="dataTypeDefs" type="dataTypeDefsType"
                       minOccurs="0"/>
        <xsd:element name="metadataDefs" type="metadataDefsType"
                       minOccurs="0"/>
        <xsd:element name="LFBClassDefs" type="LFBClassDefsType"
                       minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="provides" type="xsd:Name" use="required"/>
    </xsd:complexType>
    <!-- Ограничения уникальности -->
    <xsd:key name="frame">
      <xsd:selector xpath="lfb:frameDefs/lfb:frameDef"/>
      <xsd:field xpath="lfb:name"/>
    </xsd:key>
    <xsd:key name="dataType">
      <xsd:selector xpath="lfb:dataTypeDefs/lfb:dataTypeDef"/>
      <xsd:field xpath="lfb:name"/>
    </xsd:key>
    <xsd:key name="metadataDef">
      <xsd:selector xpath="lfb:metadataDefs/lfb:metadataDef"/>
      <xsd:field xpath="lfb:name"/>
    </xsd:key>
    <xsd:key name="LFBClassDef">
      <xsd:selector xpath="lfb:LFBClassDefs/lfb:LFBClassDef"/>
      <xsd:field xpath="lfb:name"/>
    </xsd:key>
  </xsd:element>
  <xsd:complexType name="loadType">
    <xsd:attribute name="library" type="xsd:Name" use="required"/>
    <xsd:attribute name="location" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="frameDefsType">
    <xsd:sequence>
      <xsd:element name="frameDef" maxOccurs="unbounded">
        <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="name" type="xsd:NMTOKEN"/>
            <xsd:element ref="synopsis"/>
            <xsd:element ref="description" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="dataTypeDefsType">
    <xsd:sequence>
        <xsd:element name="dataTypeDef" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:NMTOKEN"/>
              <xsd:element ref="synopsis"/>
              <xsd:element ref="description" minOccurs="0"/>
              <xsd:group ref="typeDeclarationGroup"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <!--
       Предопределенными (встроенными) неделимыми типа данных являются
           char, uchar, int16, uint16, int32, uint32, int64, uint64,
           string[N], string, byte[N], boolean, octetstring[N],
           float32, float64
```

```
    -->
    <xsd:group name="typeDeclarationGroup">
      <xsd:choice>
        <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
        <xsd:element name="atomic" type="atomicType"/>
        <xsd:element name="array" type="arrayType"/>
        <xsd:element name="struct" type="structType"/>
        <xsd:element name="union" type="structType"/>
        <xsd:element name="alias" type="typeRefNMTOKEN"/>
      </xsd:choice>
    </xsd:group>
    <xsd:simpleType name="typeRefNMTOKEN">
      <xsd:restriction base="xsd:token">
        <xsd:pattern value="\c+"/>
        <xsd:pattern value="string\[\d+\]"/>
        <xsd:pattern value="byte\[\d+\]"/>
        <xsd:pattern value="octetstring\[\d+\]"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="atomicType">
      <xsd:sequence>
        <xsd:element name="baseType" type="typeRefNMTOKEN"/>
        <xsd:element name="rangeRestriction"
                     type="rangeRestrictionType" minOccurs="0"/>
        <xsd:element name="specialValues" type="specialValuesType"
                     minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="rangeRestrictionType">
      <xsd:sequence>
        <xsd:element name="allowedRange" maxOccurs="unbounded">
          <xsd:complexType>
          <xsd:attribute name="min" type="xsd:integer" use="required"/>
          <xsd:attribute name="max" type="xsd:integer" use="required"/>
       </xsd:complexType>
     </xsd:element>
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="specialValuesType">
   <xsd:sequence>
     <xsd:element name="specialValue" maxOccurs="unbounded">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="name" type="xsd:NMTOKEN"/>
           <xsd:element ref="synopsis"/>
         </xsd:sequence>
         <xsd:attribute name="value" type="xsd:token"/>
       </xsd:complexType>
     </xsd:element>
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="arrayType">
   <xsd:sequence>
     <xsd:group ref="typeDeclarationGroup"/>
     <xsd:element name="contentKey" minOccurs="0"
                  maxOccurs="unbounded">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="contentKeyField" maxOccurs="unbounded"
                        type="xsd:string"/>
         </xsd:sequence>
         <xsd:attribute name="contentKeyID" use="required"
                        type="xsd:integer"/>
       </xsd:complexType>
       <!-- Заявление уникальных идентификаторов ключей -->
       <xsd:key name="contentKeyID">
         <xsd:selector xpath="lfb:contentKey"/>
         <xsd:field xpath="@contentKeyID"/>
       </xsd:key>
     </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="type" use="optional"
                  default="variable-size">
     <xsd:simpleType>
       <xsd:restriction base="xsd:string">
         <xsd:enumeration value="fixed-size"/>
         <xsd:enumeration value="variable-size"/>
       </xsd:restriction>
     </xsd:simpleType>
     </xsd:attribute>
     <xsd:attribute name="length" type="xsd:integer" use="optional"/>
     <xsd:attribute name="maxLength" type="xsd:integer"
                    use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="structType">
     <xsd:sequence>
```

```
        <xsd:element name="derivedFrom" type="typeRefNMTOKEN"
                     minOccurs="0"/>
        <xsd:element name="component" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:NMTOKEN"/>
              <xsd:element ref="synopsis"/>
              <xsd:element ref="description" minOccurs="0"/>
              <xsd:element name="optional" minOccurs="0"/>
              <xsd:group ref="typeDeclarationGroup"/>
            </xsd:sequence>
            <xsd:attribute name="componentID" use="required"
                           type="xsd:unsignedInt"/>
          </xsd:complexType>
          <!-- Объявление ключей для уникальности componentID в структуре
          -->
          <xsd:key name="structComponentID">
            <xsd:selector xpath="lfb:component"/>
            <xsd:field xpath="@componentID"/>
          </xsd:key>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="metadataDefsType">
      <xsd:sequence>
        <xsd:element name="metadataDef" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:NMTOKEN"/>
              <xsd:element ref="synopsis"/>
              <xsd:element name="metadataID" type="xsd:integer"/>
              <xsd:element ref="description" minOccurs="0"/>
              <xsd:choice>
                <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
                <xsd:element name="atomic" type="atomicType"/>
              </xsd:choice>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LFBClassDefsType">
      <xsd:sequence>
        <xsd:element name="LFBClassDef" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:NMTOKEN"/>
              <xsd:element ref="synopsis"/>
              <xsd:element name="version" type="versionType"/>
              <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
                           minOccurs="0"/>
              <xsd:element name="inputPorts" type="inputPortsType"
                           minOccurs="0"/>
              <xsd:element name="outputPorts" type="outputPortsType"
                           minOccurs="0"/>
              <xsd:element name="components" type="LFBComponentsType"
                           minOccurs="0"/>
              <xsd:element name="capabilities"
                           type="LFBCapabilitiesType" minOccurs="0"/>
              <xsd:element name="events"
                           type="eventsType" minOccurs="0"/>
              <xsd:element ref="description" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="LFBClassID" use="required"
                           type="xsd:unsignedInt"/>
          </xsd:complexType>
          <!-- Ограничение ключей для обеспечения уникальности имён
               атрибутов в классе
          -->
          <xsd:key name="components">
            <xsd:selector xpath="lfb:components/lfb:component"/>
            <xsd:field xpath="lfb:name"/>
          </xsd:key>
          <xsd:key name="capabilities">
            <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
            <xsd:field xpath="lfb:name"/>
          </xsd:key>
          <xsd:key name="componentIDs">
            <xsd:selector xpath="lfb:components/lfb:component"/>
            <xsd:field xpath="@componentID"/>
          </xsd:key>
          <xsd:key name="capabilityIDs">
            <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
            <xsd:field xpath="@componentID"/>
          </xsd:key>
        </xsd:element>
```

```
      </xsd:sequence>
    </xsd:complexType>
  <xsd:simpleType name="versionType">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:pattern value="[1-9][0-9]*\.([1-9][0-9]*|0)"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="inputPortsType">
    <xsd:sequence>
      <xsd:element name="inputPort" type="inputPortType"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="inputPortType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:NMTOKEN"/>
      <xsd:element ref="synopsis"/>
      <xsd:element name="expectation" type="portExpectationType"/>
      <xsd:element ref="description" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="group" type="xsd:boolean" use="optional"
                   default="0"/>
  </xsd:complexType>
  <xsd:complexType name="portExpectationType">
    <xsd:sequence>
      <xsd:element name="frameExpected" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
          <!-- Ссылка ref должна указывать определённый тип кадра -->
          <xsd:element name="ref" type="xsd:string"
                       maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="metadataExpected" minOccurs="0">
        <xsd:complexType>
          <xsd:choice maxOccurs="unbounded">
            <!-- Ссылка ref должна указывать имя определённых метаданных -->

            <xsd:element name="ref" type="metadataInputRefType"/>
            <xsd:element name="one-of"
                         type="metadataInputChoiceType"/>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="metadataInputChoiceType">
    <xsd:choice minOccurs="2" maxOccurs="unbounded">
      <!-- Ссылка ref должна указывать имя определённых метаданных -->
      <xsd:element name="ref" type="xsd:NMTOKEN"/>
      <xsd:element name="one-of" type="metadataInputChoiceType"/>
      <xsd:element name="metadataSet" type="metadataInputSetType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="metadataInputSetType">
    <xsd:choice minOccurs="2" maxOccurs="unbounded">
      <!-- Ссылка ref должна указывать имя определённых метаданных -->
      <xsd:element name="ref" type="metadataInputRefType"/>
      <xsd:element name="one-of" type="metadataInputChoiceType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="metadataInputRefType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:NMTOKEN">
        <xsd:attribute name="dependency" use="optional"
                       default="required">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="required"/>
              <xsd:enumeration value="optional"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="defaultValue" type="xsd:token"
                       use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="outputPortsType">
    <xsd:sequence>
      <xsd:element name="outputPort" type="outputPortType"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="outputPortType">
```

```
      <xsd:sequence>
        <xsd:element name="name" type="xsd:NMTOKEN"/>
        <xsd:element ref="synopsis"/>
        <xsd:element name="product" type="portProductType"/>
        <xsd:element ref="description" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="group" type="xsd:boolean" use="optional"
                     default="0"/>
    </xsd:complexType>
    <xsd:complexType name="portProductType">
      <xsd:sequence>
        <xsd:element name="frameProduced">
         <xsd:complexType>
            <xsd:sequence>
              <!-- Ссылка ref должна указывать определённый тип кадра
                   -->
                  <xsd:element name="ref" type="xsd:NMTOKEN"
                               maxOccurs="unbounded"/>
              </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="metadataProduced" minOccurs="0">
           <xsd:complexType>
             <xsd:choice maxOccurs="unbounded">
               <!-- Ссылка ref должна указывать имя определённых метаданных
               -->
               <xsd:element name="ref" type="metadataOutputRefType"/>
               <xsd:element name="one-of"
                            type="metadataOutputChoiceType"/>
             </xsd:choice>
           </xsd:complexType>
         </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="metadataOutputChoiceType">
        <xsd:choice minOccurs="2" maxOccurs="unbounded">
          <!-- Ссылка ref должна указывать имя определённых метаданных -->
          <xsd:element name="ref" type="xsd:NMTOKEN"/>
          <xsd:element name="one-of" type="metadataOutputChoiceType"/>
          <xsd:element name="metadataSet" type="metadataOutputSetType"/>
        </xsd:choice>
      </xsd:complexType>
      <xsd:complexType name="metadataOutputSetType">
        <xsd:choice minOccurs="2" maxOccurs="unbounded">
          <!-- Ссылка ref должна указывать имя определённых метаданных -->
          <xsd:element name="ref" type="metadataOutputRefType"/>
          <xsd:element name="one-of" type="metadataOutputChoiceType"/>
        </xsd:choice>
      </xsd:complexType>
      <xsd:complexType name="metadataOutputRefType">
        <xsd:simpleContent>
          <xsd:extension base="xsd:NMTOKEN">
            <xsd:attribute name="availability" use="optional"
                           default="unconditional">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="unconditional"/>
                  <xsd:enumeration value="conditional"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="LFBComponentsType">
        <xsd:sequence>
          <xsd:element name="component" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="name" type="xsd:NMTOKEN"/>
                <xsd:element ref="synopsis"/>
                <xsd:element ref="description" minOccurs="0"/>
                <xsd:element name="optional" minOccurs="0"/>
                <xsd:group ref="typeDeclarationGroup"/>
                <xsd:element name="defaultValue" type="xsd:token"
                             minOccurs="0"/>
              </xsd:sequence>
              <xsd:attribute name="access" use="optional"
                             default="read-write">
                <xsd:simpleType>
                  <xsd:list itemType="accessModeType"/>
                </xsd:simpleType>
              </xsd:attribute>
              <xsd:attribute name="componentID" use="required"
                             type="xsd:unsignedInt"/>
            </xsd:complexType>
```

```
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="accessModeType">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="read-only"/>
      <xsd:enumeration value="read-write"/>
      <xsd:enumeration value="write-only"/>
      <xsd:enumeration value="read-reset"/>
      <xsd:enumeration value="trigger-only"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="LFBCapabilitiesType">
    <xsd:sequence>
      <xsd:element name="capability" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:NMTOKEN"/>
            <xsd:element ref="synopsis"/>
            <xsd:element ref="description" minOccurs="0"/>
            <xsd:element name="optional" minOccurs="0"/>
            <xsd:group ref="typeDeclarationGroup"/>
          </xsd:sequence>
          <xsd:attribute name="componentID" use="required"
                         type="xsd:integer"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="eventsType">
    <xsd:sequence>
      <xsd:element name="event" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:NMTOKEN"/>
            <xsd:element ref="synopsis"/>
            <xsd:element name="eventTarget" type="eventPathType"/>
            <xsd:element ref="eventCondition"/>
            <xsd:element name="eventReports" type="eventReportsType"
                         minOccurs="0"/>
            <xsd:element ref="description" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="eventID" use="required"
                         type="xsd:integer"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="baseID" type="xsd:integer"
                   use="optional"/>
  </xsd:complexType>
  <!-- Группа подстановки для условий события -->
  <xsd:element name="eventCondition" abstract="true"/>
  <xsd:element name="eventCreated"
               substitutionGroup="eventCondition"/>
  <xsd:element name="eventDeleted"
               substitutionGroup="eventCondition"/>
  <xsd:element name="eventChanged"
               substitutionGroup="eventCondition"/>
  <xsd:element name="eventGreaterThan"
               substitutionGroup="eventCondition"/>
  <xsd:element name="eventLessThan"
               substitutionGroup="eventCondition"/>
  <xsd:complexType name="eventPathType">
    <xsd:sequence>
      <xsd:element ref="eventPathPart" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <!-- Группа подстановки для частей пути к событию -->
  <xsd:element name="eventPathPart" type="xsd:string"
               abstract="true"/>
  <xsd:element name="eventField" type="xsd:string"
               substitutionGroup="eventPathPart"/>
  <xsd:element name="eventSubscript" type="xsd:string"
               substitutionGroup="eventPathPart"/>
  <xsd:complexType name="eventReportsType">
    <xsd:sequence>
      <xsd:element name="eventReport" type="eventPathType"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="booleanType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="0"/>
      <xsd:enumeration value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
    </xsd:schema>
```

5. Компоненты и возможности FE

Элемент пересылки ForCES обрабатывает трафик от имени управляющего элемента ForCES. Хотя стандарты будут описывать протокол и механизмы для этого управления, разные реализации и разные экземпляры будут отличаться возможностями. Элемент CE **должен** быть способен определить, за что отвечает каждый экземпляр и что он реально может делать. Как отмечено выше, это будет аппроксимацией. Предполагается, что элемент CE будет готов справляться с ошибками в запросах и изменениями в деталях, не отражёнными в информации о возможностях FE.

В дополнение к возможностям FE будет иметь информацию, которая может быть использована для понимания и управления операциями пересылки. Часть этой информации доступна лишь для чтения, а другая разрешает и запись.

Чтобы сделать информацию FE легкодоступной, она представлена в LFB. Этот блок LFB имеет класс FEObject с идентификатором LFBClassID = 1. В FE будет присутствовать лишь один экземпляр этого класса и идентификатор экземпляра в протоколе будет иметь значение 1. таким образом, ссылаясь на компоненты class:1, instance:1, CE может получить общую информацию о FE. Класс LFB FEObject описан в этом разделе.

Будет также класс LFB FEProtocol, для которого зарезервирован идентификатор LFBClassID = 2. Этот класс также будет иметь единственный экземпляр, а его детали определены в спецификации протокола ForCES [RFC5810].

5.1. XML для определения класса FEObject

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  provides="FEObject">
  <dataTypeDefs>
    <dataTypeDef>
      <name>LFBAdjacencyLimitType</name>
      <synopsis>Описание Adjacent LFB</synopsis>
      <struct>
        <component componentID="1">
          <name>NeighborLFB</name>
          <synopsis>Идентификатор для данного класса LFB</synopsis>
          <typeRef>uint32</typeRef>
        </component>
        <component componentID="2">
          <name>ViaPorts</name>
          <synopsis>
            Порты, к которым можно подключаться
          </synopsis>
          <array type="variable-size">
            <typeRef>string</typeRef>
          </array>
        </component>
      </struct>
    </dataTypeDef>
    <dataTypeDef>
      <name>PortGroupLimitType</name>
      <synopsis>
        Предельное число портов в данной группе
      </synopsis>
      <struct>
        <component componentID="1">
          <name>PortGroupName</name>
          <synopsis>Имя группы</synopsis>
          <typeRef>string</typeRef>
        </component>
        <component componentID="2">
          <name>MinPortCount</name>
          <synopsis>Минимальное число портов</synopsis>
          <optional/>
          <typeRef>uint32</typeRef>
        </component>
        <component componentID="3">
          <name>MaxPortCount</name>
          <synopsis>Максимальное число портов</synopsis>
          <optional/>
          <typeRef>uint32</typeRef>
        </component>
      </struct>
    </dataTypeDef>
    <dataTypeDef>
      <name>SupportedLFBType</name>
      <synopsis>запись таблицы для поддерживаемых LFB</synopsis>
      <struct>
        <component componentID="1">
          <name>LFBName</name>
          <synopsis>Имя поддерживаемого класса LFB</synopsis>
          <typeRef>string</typeRef>
        </component>
        <component componentID="2">
          <name>LFBClassID</name>
          <synopsis>Идентификатор поддерживаемого класса LFB</synopsis>
          <typeRef>uint32</typeRef>
```

```
            </component>
            <component componentID="3">
              <name>LFBVersion</name>
              <synopsis>Версия класса LFB, используемого этим FE.</synopsis>
              <typeRef>string</typeRef>
            </component>
            <component componentID="4">
              <name>LFBOccurrenceLimit</name>
              <synopsis>
                Максимальное число экземпляров данного класса LFB
              </synopsis>
              <optional/>
              <typeRef>uint32</typeRef>
            </component>
            <!-- Для каждой группы указывается допустимое число портов
            -->
            <component componentID="5">
              <name>PortGroupLimits</name>
              <synopsis>Таблица пределов групп портов</synopsis>
              <optional/>
              <array type="variable-size">
                <typeRef>PortGroupLimitType</typeRef>
              </array>
            </component>
<!-- Классы LFB, за которыми может следовать именованный LFB Class -->
            <component componentID="6">
              <name>CanOccurAfters</name>
              <synopsis>
                Список классов LFB, за которыми может следовать этот класс
              </synopsis>
              <optional/>
              <array type="variable-size">
                <typeRef>LFBAdjacencyLimitType</typeRef>
              </array>
            </component>
<!-- Классы LFB, которые могут следовать за именованным LFB Class -->
            <component componentID="7">
              <name>CanOccurBefores</name>
              <synopsis>
                Список классов LFB, которые могут следовать за этим классом
              </synopsis>
              <optional/>
              <array type="variable-size">
                <typeRef>LFBAdjacencyLimitType</typeRef>
              </array>
            </component>
            <component componentID="8">
              <name>UseableParentLFBClasses</name>
              <synopsis>
                Список классов LFB, из которых унаследован этот класс
                и которые FE разрешает для ссылок на экземпляры этого
                класса.
              </synopsis>
              <optional/>
              <array type="variable-size">
                <typeRef>uint32</typeRef>
              </array>
            </component>
          </struct>
        </dataTypeDef>
        <dataTypeDef>
          <name>FEStateValues</name>
          <synopsis>Возможные значения статуса</synopsis>
          <atomic>
            <baseType>uchar</baseType>
            <specialValues>
              <specialValue value="0">
                <name>AdminDisable</name>
                <synopsis>FE административно запрещён</synopsis>
              </specialValue>
              <specialValue value="1">
                <name>OperDisable</name>
                <synopsis>FE исключён из работы</synopsis>
              </specialValue>
              <specialValue value="2">
                <name>OperEnable</name>
                <synopsis>FE работает</synopsis>
              </specialValue>
            </specialValues>
          </atomic>
        </dataTypeDef>
        <dataTypeDef>
          <name>FEConfiguredNeighborType</name>
          <synopsis>Детали соседей FE</synopsis>
          <struct>
            <component componentID="1">
```

```
            <name>NeighborID</name>
            <synopsis>FEID соседа</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="2">
            <name>InterfaceToNeighbor</name>
            <synopsis>Интерфейс FE к этому соседу</synopsis>
            <optional/>
            <typeRef>string</typeRef>
          </component>
          <component componentID="3">
            <name>NeighborInterface</name>
            <synopsis>
              Имя интерфейса соседа, с которым данных FE
              является смежным. Это нужно при наличии более
              одного смежного FE на интерфейсе.
            </synopsis>
            <optional/>
            <typeRef>string</typeRef>
          </component>
        </struct>
      </dataTypeDef>
      <dataTypeDef>
        <name>LFBSelectorType</name>
        <synopsis>Уникальное имя экземпляра класса LFB</synopsis>
        <struct>
          <component componentID="1">
            <name>LFBClassID</name>
            <synopsis>Идентификатор класса LFB</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="2">
            <name>LFBInstanceID</name>
            <synopsis>Идентификатор экземпляра LFB</synopsis>
            <typeRef>uint32</typeRef>
          </component>
        </struct>
      </dataTypeDef>
      <dataTypeDef>
        <name>LFBLinkType</name>
        <synopsis>
          Канал между двумя экземплярами LFB в топологии
        </synopsis>
        <struct>
          <component componentID="1">
            <name>FromLFBID</name>
            <synopsis>Источник LFB</synopsis>
            <typeRef>LFBSelectorType</typeRef>
          </component>
          <component componentID="2">
            <name>FromPortGroup</name>
            <synopsis>Группа портов источника</synopsis>
            <typeRef>string</typeRef>
          </component>
          <component componentID="3">
            <name>FromPortIndex</name>
            <synopsis>Индекс порта-источника</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="4">
            <name>ToLFBID</name>
            <synopsis>LFBID назначения</synopsis>
            <typeRef>LFBSelectorType</typeRef>
          </component>
          <component componentID="5">
            <name>ToPortGroup</name>
            <synopsis>Группа портов назначения</synopsis>
            <typeRef>string</typeRef>
          </component>
          <component componentID="6">
            <name>ToPortIndex</name>
            <synopsis>Индекс порта назначения</synopsis>
            <typeRef>uint32</typeRef>
          </component>
        </struct>
      </dataTypeDef>
    </dataTypeDefs>
    <LFBClassDefs>
      <LFBClassDef LFBClassID="1">
        <name>FEObject</name>
        <synopsis>Core LFB: FE Object</synopsis>
        <version>1.0</version>
        <components>
          <component access="read-write" componentID="1">
            <name>LFBTopology</name>
            <synopsis>Таблица известных топологий</synopsis>
```

```
            <array type="variable-size">
              <typeRef>LFBLinkType</typeRef>
            </array>
          </component>
          <component access="read-write" componentID="2">
            <name>LFBSelectors</name>
            <synopsis>
               Таблица известных активных классов и экземпляров LFB
            </synopsis>
            <array type="variable-size">
              <typeRef>LFBSelectorType</typeRef>
            </array>
          </component>
          <component access="read-write" componentID="3">
            <name>FEName</name>
            <synopsis>Имя данного FE</synopsis>
            <typeRef>string[40]</typeRef>
          </component>
          <component access="read-write" componentID="4">
            <name>FEID</name>
            <synopsis>Идентификатор данного FE</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component access="read-only" componentID="5">
            <name>FEVendor</name>
            <synopsis>Производитель FE</synopsis>
            <typeRef>string[40]</typeRef>
          </component>
          <component access="read-only" componentID="6">
            <name>FEModel</name>
            <synopsis>Модель FE</synopsis>
            <typeRef>string[40]</typeRef>
          </component>
          <component access="read-write" componentID="7">
            <name>FEState</name>
            <synopsis>Состояние данного FE</synopsis>
            <typeRef>FEStateValues</typeRef>
          </component>
          <component access="read-write" componentID="8">
            <name>FENeighbors</name>
            <synopsis>таблица известных соседей</synopsis>
            <optional/>
            <array type="variable-size">
              <typeRef>FEConfiguredNeighborType</typeRef>
            </array>
          </component>
        </components>
        <capabilities>
          <capability componentID="30">
            <name>ModifiableLFBTopology</name>
            <synopsis>Поддержка изменяемых LFB</synopsis>
            <optional/>
            <typeRef>boolean</typeRef>
          </capability>
          <capability componentID="31">
            <name>SupportedLFBs</name>
            <synopsis>Список всех поддерживаемых LFB</synopsis>
            <optional/>
            <array type="variable-size">
              <typeRef>SupportedLFBType</typeRef>
            </array>
          </capability>
        </capabilities>
      </LFBClassDef>
    </LFBClassDefs>
  </LFBLibrary>
```

5.2. Возможности FE

Информация о возможностях FE содержится в элементе <capabilities> определения класса. Как уже было сказано, информация о возможностях доступна лишь для чтения.

В настоящее время определены возможности ModifiableLFBTopology и SupportedLFBs. Информация о поддерживаемых компонентах LFB FEObject доступна из свойств этих компонент.

5.2.1. ModifiableLFBTopology

Компонента относится к логическому типу (boolean) и показывает, может ли CE изменить топологию LFB для FE. При отсутствии компоненты предполагается значение true и CE считает, что топологию LFB можно изменить. Если присутствует значение false, топология LFB для FE является фиксированной. В таких случаях элемент SupportedLFBs может быть опущен и список поддерживаемых LFB выводится CE из топологической информации LFB. Если список поддерживаемых LFB представлен и ModifiableLFBTopology = false, информацию CanOccurBefore и CanOccurAfter следует опускать.

5.2.2. SupportedLFBs и SupportedLFBType

Одной из возможностей, которую FE следует включать, является список поддерживаемых классов LFB. Компонента SupportedLFBs является массивом, содержащим информацию о каждом поддерживаемом классе LFB. Структура массива определена как SupportedLFBType dataTypeDef.

Каждый элемент массива SupportedLFBs описывает класс LFB, поддерживаемый FE. В дополнение к указанию поддерживаемых классов FE с изменяемой топологией LFB **следует** включать информацию о том, как LFB указанного класса могут соединяться с другими LFB. В этой информации **следует** описывать, порядок размещения классов LFB в топологии LFB. FE **следует** указывать, к какой группе портов может быть подключён данный смежный класс LFB. Если информация о группе портов опущена, предполагается возможность использования всех портов группы. Эта информация о приемлемом порядке и соединениях LFB **может** быть опущена если разработчик считает, что фактические ограничения будут вводить CE в заблуждение.

5.2.2.1. LFBName

Значением этой компоненты является имя описываемого класса LFB.

5.2.2.2. LFBClassID

LFBClassID содержит числовой идентификатор описываемого класса LFB. Концептуально это дублирует имя LFB, но используются оба для чёткости и проверки согласованности.

5.2.2.3. LFBVersion

LFBVersion содержит строку, указывающую версию класса LFB, поддерживаемую этим FE. Как указано выше, FE может поддерживать лишь одну версию данного класса LFB.

5.2.2.4. LFBOccurrenceLimit

При наличии этой компоненты она указывает максимальное число экземпляров данного класса LFB, которые FE может поддерживать. Для FE, не имеющих возможности создавать или уничтожать экземпляры LFB, эта компонента может быть опущена или совпадать с числом экземпляров LFB этого класса в атрибуте списка LFB.

5.2.2.5. PortGroupLimits и PortGroupLimitType

Компонента PortGroupLimits является массивом информации о группах портов, поддерживаемых этим классом LFB. Структура информации об ограничении числа портов в группе определяется PortGroupLimitType dataTypeDef.

Каждый элемент массива PortGroupLimits содержит информацию об одной группе портов класса LFB, включающую имя группы портов в компоненте PortGroupName, наименьшее число портов группы в компоненте MinPortCount и наибольшее число портов группы в компоненте MaxPortCount.

5.2.2.6. CanOccurAfters и LFBAdjacencyLimitType

Компонента CanOccurAfters является массивом, содержащим список LFB, после которых может размещаться описываемый класс. Элементы массива определены в LFBAdjacencyLimitType dataTypeDef.

Элементы массива описывают разрешённое размещение данного класса LFB, называемого здесь SupportedLFB. В частности, каждый элемент именует LFB, который может топологически предшествовать данному классу LFB. То есть SupportedLFB может иметь входной порт, соединённый с выходным портом LFB, указанного в массиве CanOccurAfters. Класс LFB, за которым может следовать SupportedLFB, указывается компонентой NeighborLFB (LFBAdjacencyLimitType dataTypeDef) в массиве CanOccurAfters. Если сосед может быть подключён лишь к определённому набору групп входных портов, включается компонента viaPort, являющаяся массивом с элементом для каждой группы входных портов в SupportedLFB, который может быть соединён с выходным портом NeighborLFB.

Например, в записи SupportedLFBs, каждый элемент массива CanOccurAfters должен иметь уникального соседа NeighborLFB, а в каждом таком элементе массива каждая компонента viaPort должна представлять свою действительную группу входных портов SupportedLFB. Схема определения класса LFB не включает этих требований к уникальности.

5.2.2.7. CanOccurBefores и LFBAdjacencyLimitType

Массив CanOccurBefores содержит информацию о классах LFB, которые могут следовать за описываемым классом. Структурно элемент похож на CanOccurAftersи использует такое же определение типа для элементов массива.

Элементы массива указывают классы LFB, которым SupportedLFB может предшествовать в топологии. В этой компоненте записи viaPort указывают группы выходных портов SupportedLFB, которые могут быть подключены к NeighborLFB. Как и в CanOccurAfters, viaPort может иметь множество записей если многим группам портов разрешено соединение с данным классом NeighborLFB.

Приведённые выше требования к уникальности для CanOccurAfter применимы и к CanOccurBefore, даже если класс LFB указан сразу в CanOccurAfter и CanOccurBefore.

5.2.2.8. UseableParentLFBClasses

При наличии массива UseableParentLFBClasses он используется для хранения списка идентификаторов родительских классов LFB. Все элементы массива должны быть идентификаторами классов, для которых класс SupportedLFB class является наследником[1] (напрямую или через промежуточного предка). Кроме того, включая данный класс в список, FE указывает CE, что данный родительский класс может использовать для манипуляций с экземпляром этого поддерживаемого класса LFB.

Разрешая такое замещение, FE допускает случай, когда созданный экземпляр LFB может относиться к классу, не известному CE, но доступному для манипуляций. Есть надежда, что такие ситуации будут редкими, желательно, чтобы

[1]Если FE включает в этот список непригодные значения, это с высокой вероятностью приведёт к некорректным действиям CE и отказам в работе.

это поддерживалось. Это может произойти, если FE локально определяет некоторые экземпляры LFB или более ранний элемент CE настроил некоторые экземпляры LFB. Также это может происходить, когда FE предпочтёт создать более новый и более подходящий экземпляр LFB.

Чтобы разрешить это, элемент FE **должен** быть более сдержанным при назначении идентификаторов экземпляров LFB. Обычно идентификаторы экземпляров определяются классом LFB. Однако, если два класса LFB имеют общего родителя и этот родитель включён в UseableParentLFBClasses для обоих классов LFB, все экземпляры обоих классов (или всех, если много классов имеет общего родителя) **должны** использовать разные экземпляры. Это позволяет FE определить, какой из экземпляров LFB предназначен для манипуляций CE даже при использовании родительского класса.

5.2.2.9. LFBClassCapabilities

Информация об уровне возможностей класса желательна, но модель её не включает. Хотя такая информация относится к FE Object в таблице поддерживаемых классов, содержимое её зависит от класса. Ожидаемые в настоящее время структуры кодирования при передаче информации между CE и FE таковы, что размещение совершенно не описанной информации может приводить к ошибкам при её анализе. Можно было бы выбрать для этой информации тип octetstring, но это требует определения внутреннего формата строки октетов.

Поскольку в настоящее время отсутствуют какие-либо определения возможностей LFB на уровне класса, о которых FE требуется сообщать, такая информация здесь не представлена, но может быть добавлена в будущих версиях объекта FE (этот то случай, когда требуется управления версиями, а не наследование, поскольку объект FE должен иметь идентификаторы объекта и класса со значением 1, чтобы протокол мог начинать работу с поиска этого объекта).

5.3. Компоненты FE

Элемент <components> включается, если определение класса содержит определение компонент объекта FE, которы не считаются «возможностями» (capabilities). Некоторые из этих компонент доступны для записи, а другие - лишь для чтения, что можно проверить путём просмотра информации компонент.

5.3.1. FEState

Эта компонента передаёт общее состояние FE и может принимать значение AdminDisable, OperDisable или OperEnable. Начальным состояние является OperDisable, а переход в OperEnable управляется FE. Элемент CE контролирует переходы между состояниями OperEnable и AdminDisable. Дополнительная информация приведена в спецификации протокола ForCES [RFC5810].

5.3.2. LFBSelectors и LFBSelectorType

LFBSelectors представляет собой массив информации LFB, доступных в данный момент по протоколу ForCES в FE. Структура информации LFB определена LFBSelectorType dataTypeDef.

Каждый элемент массива описывает один экземпляр LFB в элементе FE. Запись содержит числовые идентификаторы класса экземпляра LFB и данного экземпляра.

5.3.3. LFBTopology и LFBLinkType

Необязательная компонента LFBTopology содержит информацию о соединениях между LFB внутри FE, описывая каждое такое соединение в LFBLinkType dataTypeDef. Компонента LFBLinkType содержит информацию, достаточную для точного указания конечных точек соединения. Компоненты FromLFBID и ToLFBID задают экземпляры LFB на каждой стороне соединения и **должны** указывать LFB из таблицы экземпляров LFB. Компоненты FromPortGroup и ToPortGroup **должны** указывать входную и выходную группу портов, определённые в классах LFB экземпляров LFB, указанных в FromLFBID и ToLFBID. Компоненты FromPortIndex и ToPortIndex выбирают из групп порты, к которым это соединение подключено. Все соединения однозначно указываются полями FromLFBID, FromPortGroup и FromPortIndex. Множество соединений может иметь одинаковые значения ToLFBID, ToPortGroup и ToPortIndex, поскольку эта модель поддерживает входное (но не выходное) разветвление.

5.3.4. FENeighbors и FEConfiguredNeighborType

FENeighbors - это массив информации о настроенных вручную отношениях смежности между данным FE и другими элементами FE. Содержимое массива определяется FEConfiguredNeighborType dataTypeDef.

Массив предназначен для сбора информации, которая может быть задана на FE и нужна CE. Каждый элемент массива соответствует одному соседу. Отметим, что массив не отражает результатов автоматического обнаружения, поскольку они будут указывать свои LFB. Эта компонента не обязательна.

Существует много способов указания соседей, но для сопоставления в CE наиболее эффективно применять FE-ID. Идентификатор интерфейса (строка имени) является лучшим коррелятором. CE сможет получить IP-адрес и данные уровня среды напрямую от соседа. Отсутствие этой информации в таблице позволяет предотвратить риск некорректной «двойной настройки».

Информация о предполагаемых формах обмена данными с соседом не задаётся здесь, включены лишь данные о соседстве.

5.3.4.1. NeighborID

Это идентификатор в неком пространстве имён, значимый для CE применительно к соседу.

5.3.4.2. InterfaceToNeighbor

Идентификатор интерфейса, через который доступен сосед.

5.3.4.3. NeighborInterface

Указывает интерфейс соседа, через который тот подключён. Идентификация интерфейса нужна в тех случаях, когда лишь одна сторона соединения имеет данные конфигурации или два элемент FE имеют между собой несколько соединений.

6. Выполнение требований к модели FE

В этом разделе рассматривается выполнение предложенной моделью FE требований, заданных в разделе 5 RFC 3654 [RFC3654]. Требования разделены на общую часть (раздел 5, параграфы 5.1 - 5.4) и требования к минимальному набору логических функций, которые должна поддерживать модель FE (параграф 5.5).

Общим требованием к модели FE является способность выразить возможность логической обработки пакетов в FE через возможности и состояния модели. Кроме того, предполагается, что модель FE обеспечивает гибкость реализации и может быть расширена для определения новых логических функций.

Основной компонентой предложенной модели FE является логический функциональный блок LFB. Каждая логическая функция в FE моделируется как LFB. Рабочие параметры LFB, которые должны быть видимы CE, называются компонентами LFB. Эти компоненты указывают возможности FE и поддерживают гибкость реализаций, позволяя FE указать поддерживаемые дополнительные возможности. Компоненты также указывают возможность настройки класса LFB элементами CE. Настраиваемые компоненты обеспечивают CE некоторую гибкость при задании поведения LFB. При создании множества экземпляров LFB одного класса LFB в элементе FE каждый из экземпляров LFB может быть настроен с разными параметрами компонент. Запрашивая настройки компонент экземпляров LFB, элемент CE может определить состояние LFB.

Созданные экземпляры LFB соединяются в направленный граф, который описывает упорядочение функций в FE. Этот граф описывается топологической моделью. Комбинация компонент экземпляров LFB и топологии описывает функции обработки пакетов, доступные в FE (текущее состояние).

Другими важными компонентами модели FE являются компоненты FE, которые служат в основном для описания возможностей FE, а также передают информацию о состоянии FE.

Модель FE включает лишь определение самого FE Object LFB. Для выполнения полного набора требований рабочей группы нужны дополнительные LFB. Определение классов для этих LFB будет дано в других документах.

7. Использование модели FE в протоколе ForCES

Фактическая модель уровня пересылки в данном NE заключается в том, что элемент CE должен изучать и контролировать, взаимодействуя с элементами FE (или иными способами). Большая часть таких коммуникаций будет происходить после создания ассоциации с использованием протокола ForCES. Ниже перечислены данные, которыми элементы CE и FE должны обмениваться по протоколу ForCES [RFC5810].

1. Запрос топологии FE.
2. Объявление возможностей FE.
3. Запрос топологии LFB (на уровне FE) и настройки возможностей.
4. Объявление возможностей LFB.
5. Запрос состояния компонент LFB.
6. Манипуляции с компонентами LFB.
7. Изменение топологии LFB.

В пп. 1 - 5 основной поток информации идёт от FE к CE. Пп. 1 - 4 обычно запрашиваются элементами CE на начальном этапе фазы PA[1], хотя они могут повторяться в фазе PA (в любой момент). П. 5 (запрос состояния) используется в начале фазы PA и часто повторяется в этой фазе (особенно запросы счётчиков статистики).

Пп. 6 и 7 представляют собой «команды» и основной поток информации идёт от CE к FE. Сообщения п. 6 (команды изменения конфигурации LFB) предполагаются достаточно частыми. П. 7 (изменение топологии LFB) нужно использовать ли при поддержке FE динамического изменения топологии LFB и они предполагаются нечастыми.

Топология соединений между FE (п. 1) может быть определена CE разными способами. Ни данный документ, ни спецификация ForCES [RFC5810] не задают конкретных механизмов. Определение класса LFB включает возможность настройки идентификации соседей на FE или предоставление этой информации по запросу CE. Могут быть также определены специальные классы LFB и протоколы для обнаружения соседей. CE может применять протоколы маршрутизации для определения отношений смежности. Соответствующая конфигурация может быть задана в CE.

Связи между моделью FE и семью сообщениями фазы PA показаны на рисунке 12.

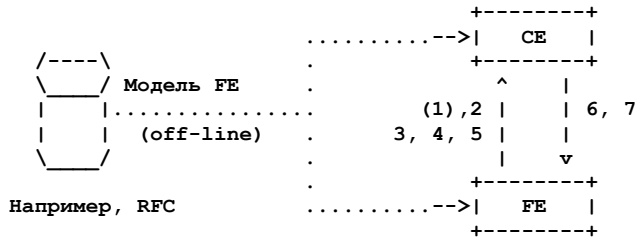


Рисунок 12. Связь между моделью FE и сообщениями протокола ForCES, где (1) - часть базового протокола ForCES, а остальное определено моделью FE

[1]Post-association - после создания ассоциации.

Кодирование этих сообщений определяется протоколом ForCES [RFC5810] и выходит за рамки модели FE. Тем не менее важно рассмотреть этот вопрос здесь в силу перечисленных ниже причин.

- Компоненты модели PA оказывают существенное влияние на модель FE. Например, часть упомянутой выше информации может быть представлена в виде компонент LFB, которые в этом случае должны определяться в классах LFB.
- Знание типа информации, которая может передаваться между FE и CE может помочь при выборе подходящего протокольного формата и метода кодирования (такого как XML, TLV).
- Знание частоты этого типа сообщений должно влиять на выбор протокольного формата (эффективность).

Оставшиеся параграфы этого раздела посвящены каждому из семи типов сообщений.

7.1. Запрос топологии FE

FE может (не обязательно) иметь один или несколько внешних входных портов, а также может (не обязательно) иметь один или несколько внешних выходных портов. Иными словами, не каждый элемент FE имеет внешние входные и/или выходные интерфейсы. Например, на рисунке 13 показан каскад из двух FE. FE #1 имеет внешний входной интерфейс, но не имеет внешнего выходного, а FE #2 - наоборот. Можно соединить эти два FE через их внутренние интерфейсы для обеспечения входной и выходной обработки. Это позволяет гибко распределять функции между множеством FE, а затем соединять их в соответствии с задачами.

Рисунок 13. Пример двух соединённых вместе FE.

Хотя протокол взаимодействия между FE выходит за рамки ForCES, элементам CE нужно понимать, как соединены между собой FE для полной реализации функций входной и выходной обработки, например, как показано на рисунке 13. Топологию соединений могут предоставлять устройства FE, она может быть жёстко закодирована в CE или предоставляться иным путём (например, менеджером шины) независимо от FE. Поэтому, хотя протокол ForCES [RFC5810] и позволяет запросить у FE топологические данные, CE не обязательно использовать это при наличии у CE других способов получения этой информации.

Когда топология соединений между FE получена CE по запросу, далее она считается статичной. Однако элемент FE может быть отключён в процессе работы NE или может быть установлена новая плата и активизирован новый элемент FE и топология соединений между FE может измениться. Протокол ForCES должен обеспечивать для CE механизм детектирования таких событий и обработки изменений в топологии FE, однако топология FE выходит за рамки модели FE.

7.2. Объявление возможностей FE

FE может иметь разные типы ограничений, часть которых может быть представлена CE как часть модели возможностей. Элементы CE должны быть способны запрашивать у отдельного FE:

- возможности передачи метаданных (понимание этих возможностей поможет CE пригодность топологии LFB и определить доступность некоторых служб);
- глобальные ограничения на запрос ресурсов (применимо ко всем LFB в FE);
- LFB, поддерживаемые FE;
- пределы числа экземпляров классов LFB;
- топологические ограничения LFB (соединения, порядок и т. п.).

7.3. Запрос топологии LFB и возможности её настройки

Протокол ForCES должен обеспечивать CE способ определить текущий набор экземпляров LFB в FE и соединения между LFB внутри FE. В дополнение к этому следует обеспечивать информацию, достаточную для определения FE, поддерживающих любые инициированные CE (динамические) изменения топологии LFB, и определения для них разрешённых топологий. Возможность настройки топологии может также рассматриваться как часть запроса возможностей FE, как описано в параграфе 7.2.

7.4. Объявления возможностей LFB

Спецификации классов LFB задают базовый набор возможностей. Когда экземпляры LFB реализованы (созданы) на FE производителя, могут вводиться дополнительные ограничения. Отметим, что здесь рассматриваются лишь ограничения в пределах гибкости спецификации класса LFB. Т. е. экземпляр LFB остаётся соответствующим спецификации класса LFB, несмотря на эти ограничения. Например, некоторые свойства класса LFB могут быть необязательными и в таких случаях нужно давать CE возможность решения вопроса о поддержке этих свойств данным

экземпляром LFB. Кроме того, определения классов LFB будут вероятно включать немного количественных ограничения (например, размер таблиц), поскольку такие ограничения обычно вносятся реализацией. Поэтому количественные ограничения следует всегда выражать как аргументы возможностей.

Экземпляры LFB в модели конкретной реализации FE будут следовать ограничениям возможностей, определенным в соответствующем классе LFB. Спецификации классов LFB должны определять набор аргументов возможностей а элемент CE должен быть способен узнать фактические возможности экземпляра LFB, запрашивая значения таких аргументов. Запросы возможностей обычно будут выполняться при обнаружении LFB элементом CE. Возможности не требуется запрашивать снова в случае статических ограничений. Однако в некоторых случаях (например при добавлении или удалении других LFB, а также настройке компонент в других LFB при использовании блоками LFB общих физических ресурсов) возможности могут изменяться и тогда нужны механизмы информирования об этом CE.

Ниже перечислены два имеющихся типа ограничений.

- Качественные ограничения. Например, класс LFB стандартизованного классификатора по множеству полей может определять большое число полей классификации, но данный FE может поддерживать лишь часть их.
- Количественные ограничения, такие как размер таблиц и т. п.

Параметры возможностей, которые могут быть запрошены для данного класса LFB, будут частью спецификации класса LFB. Эти параметры следует рассматривать как специальные компоненты LFB. Фактические значения этих компонент могут быть, следовательно, получены с использованием обычных механизмов запроса компонент LFB.

Компоненты возможностей являются доступными лишь для чтения аргументами. В случаях, когда можно позволить CE менять эти значения, информация должна представляться как рабочая компонента, а не компонента возможностей.

В предположении достаточно редкого изменения возможностей эффективность протокола/схемы/кодирования отходит на второй план.

Большая часть информации об ограничениях собирается в свойствах компонент, поэтому к ней можно получить доступ обычным для модели путём.

7.5. Запрос состояния компонент LFB

Это свойство должно поддерживаться всеми FE. Протокол ForCES и схема/кодирование данных протокола должны совместно выполнять приведённые ниже требования для упрощения запроса состояния компонент LFB.

- Должен поддерживаться выбор FE. Это относится прежде всего к указанию одного FE, но может поддерживаться и указание группы или всех FE.
- Должен разрешаться выбор экземпляра LFB. Это относится прежде всего к указанию одного экземпляра LFB, но может поддерживаться и указание группы или всех экземпляров LFB.
- Должна поддерживаться адресация отдельных компонент LFB.
- Должно поддерживаться эффективное кодирование и декодирование адресной информации и настроенных данных.
- Должна обеспечиваться эффективная передача данных состояния компонент через «провод» для минимизации загрузки канала между CE и FE.

7.6. Манипуляции компонентами LFB

Модель FE обеспечивает определения для классов LFB, каждый из которых имеет глобально уникальный идентификатор. Информация внутри класса представляется как компоненты и назначенные идентификаторы в области действия этого класса. Модель также указывает, что экземпляры классов LFB имеют идентификаторы. Комбинация идентификатора класса, идентификатора экземпляра и идентификатора компоненты используется протоколом для доступа к информации LFB в протокольных операциях.

7.7. Изменение топологии LFB

Операции, которые требуются для перенастройки топологии LFB, включают:

- создание нового экземпляра данного класса LFB в данном FE;
- подключение данного выхода LFB x к данному входу LFB y;
- отключение путём удаления канала между данным выходом одного LFB и данным входом другого;
- удаление данного LFB (автоматически удаляются все его соединения с другими LFB).

8. Пример определения LFB

В этом разделе приведён пример определения LFB. Хотя некоторые свойства LFB показаны блоком FE Object LFB, здесь показано, как может строиться LFB уровня данных. Этот пример является вымышленным случаем интерфейса, поддерживающего CWDM и передающего трафик Frame Relay. Статистическая информация (включая ошибки) не включена.

Последняя часть этого примера включает ссылки на протокольные операции. Формат и поля здесь указаны исключительно для иллюстрации, поскольку точный синтаксис и семантику операций определяет протокол ForCES [RFC5810].

```
<?xml version="1.0" encoding="UTF-8"?>
      <LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       provides="LaserFrameLFB">
        <frameDefs>
```

```
      <frameDef>
        <name>FRFrame</name>
        <synopsis>
            Кадр Frame Relay с DLCI без «прокладки»</synopsis>
      </frameDef>
      <frameDef>
        <name>IPFrame</name>
         <synopsis>Пакет IP</synopsis>
      </frameDef>
    </frameDefs>
    <dataTypeDefs>
      <dataTypeDef>
        <name>frequencyInformationType</name>
        <synopsis>Информация об одной частоте CWDM</synopsis>
        <struct>
          <component componentID="1">
            <name>LaserFrequency</name>
            <synopsis>Кодированная частота (канал)</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="2">
            <name>FrequencyState</name>
            <synopsis>Состояние частоты</synopsis>
            <typeRef>PortStatusValues</typeRef>
          </component>
          <component componentID="3">
            <name>LaserPower</name>
            <synopsis>Текущая наблюдаемая мощность</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="4">
            <name>FrameRelayCircuits</name>
            <synopsis>
                Информация об устройствах на этой частоте
            </synopsis>
            <array>
              <typeRef>frameCircuitsType</typeRef>
            </array>
          </component>
        </struct>
      </dataTypeDef>
      <dataTypeDef>
        <name>frameCircuitsType</name>
        <synopsis>
            Информация об одном устройстве (канале) Frame Relay
        </synopsis>
        <struct>
          <component componentID="1">
            <name>DLCI</name>
            <synopsis>DLCI of the circuit</synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="2">
            <name>CircuitStatus</name>
            <synopsis>Состояние устройства</synopsis>
            <typeRef>PortStatusValues</typeRef>
          </component>
          <component componentID="3">
            <name>isLMI</name>
            <synopsis>Является ли устройство LMI[1]</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="4">
            <name>associatedPort</name>
            <synopsis>
                Какой входной/выходной порт связан с устройством
            </synopsis>
            <typeRef>uint32</typeRef>
        </struct>
      </dataTypeDef>
      <dataTypeDef>
        <name>PortStatusValues</name>
        <synopsis>
            Возможные значения состояния (рабочего и административного)
        </synopsis>
        <atomic>
          <baseType>uchar</baseType>
          <specialValues>
            <specialValue value="0">
              <name>Disabled </name>
              <synopsis>Компонента отключена</synopsis>
            </specialValue>
            <specialValue value="1">
              <name>Enabled</name>
```

[1]Local Management Interface - локальный интерфейс управления. *Прим. перев.*

```
              <synopsis>Элемент FE оперативно включён</synopsis>
            </specialValue>
          </specialValues>
        </atomic>
      </dataTypeDef>
    </dataTypeDefs>
    <metadataDefs>
      <metadataDef>
        <name>DLCI</name>
        <synopsis>DLCI прибывшего кадра</synopsis>
        <metadataID>12</metadataID>
        <typeRef>uint32</typeRef>
      </metadataDef>
      <metadataDef>
        <name>LaserChannel</name>
        <synopsis>Индекс канала лазера</synopsis>
        <metadataID>34</metadataID>
        <typeRef>uint32</typeRef>
      </metadataDef>
    </metadataDefs>
    <LFBClassDefs>
        <!-- Фиктивный идентификатор класса с действительным значением -->
      <LFBClassDef LFBClassID="255">
        <name>FrameLaserLFB</name>
        <synopsis>Фиктивный блок LFB для демонстрации</synopsis>
        <version>1.0</version>
        <inputPorts>
          <inputPort group="true">
            <name>LMIfromFE</name>
            <synopsis>Порты для передачи трафика LMI</synopsis>
            <expectation>
              <frameExpected>
                <ref>FRFrame</ref>
              </frameExpected>
              <metadataExpected>
                <ref>DLCI</ref>
                <ref>LaserChannel</ref>
              </metadataExpected>
            </expectation>
          </inputPort>
          <inputPort>
            <name>DatafromFE</name>
            <synopsis>
                Порты для данных, передаваемых в каналы (устройства)
            </synopsis>
            <expectation>
              <frameExpected>
                <ref>IPFrame</ref>
              </frameExpected>
              <metadataExpected>
                <ref>DLCI</ref>
                <ref>LaserChannel</ref>
              </metadataExpected>
            </expectation>
          </inputPort>
        </inputPorts>
        <outputPorts>
          <outputPort group="true">
            <name>LMItoFE</name>
            <synopsis>Порты для обрабатываемого трафика LMI</synopsis>
            <product>
              <frameProduced>
                <ref>FRFrame</ref>
              </frameProduced>
              <metadataProduced>
                <ref>DLCI</ref>
                <ref>LaserChannel</ref>
              </metadataProduced>
            </product>
          </outputPort>
          <outputPort group="true">
            <name>DatatoFE</name>
            <synopsis>Порты для обрабатываемого трафика данных</synopsis>
            <product>
              <frameProduced>
                <ref>IPFrame</ref>
              </frameProduced>
              <metadataProduced>
                <ref>DLCI</ref>
                <ref>LaserChannel</ref>
              </metadataProduced>
            </product>
          </outputPort>
        </outputPorts>
        <components>
          <component access="read-write" componentID="1">
```

```
      <name>AdminPortState</name>
      <synopsis>Разрешена ли работа этого порта</synopsis>
      <typeRef>PortStatusValues</typeRef>
    </component>
    <component access="read-write" componentID="2">
      <name>FrequencyInformation</name>
      <synopsis>Таблица информации для частоты CWDM</synopsis>
      <array type="variable-size">
        <typeRef>frequencyInformationType</typeRef>
      </array>
    </component>
  </components>
  <capabilities>
    <capability componentID="31">
      <name>OperationalState</name>
      <synopsis>Работает ли порт всегда</synopsis>
      <typeRef>PortStatusValues</typeRef>
    </capability>
    <capability componentID="32">
      <name>MaximumFrequencies</name>
      <synopsis>Число имеющихся длин волн</synopsis>
      <typeRef>uint16</typeRef>
    </capability>
    <capability componentID="33">
      <name>MaxTotalCircuits</name>
      <synopsis>
          Общее число поддерживаемых устройств Frame Relay
          на всех длинах волн
      </synopsis>
      <optional/>
      <typeRef>uint32</typeRef>
    </capability>
  </capabilities>
  <events baseID="61">
    <event eventID="1">
      <name>FrequencyState</name>
      <synopsis>
          Состояние лазерной частоты изменилось
      </synopsis>
      <eventTarget>
        <eventField>FrequencyInformation</eventField>
        <eventSubscript>_FrequencyIndex_</eventSubscript>
        <eventField>FrequencyState</eventField>
      </eventTarget>
      <eventChanged/>
      <eventReports>
          <!-- Отчёт о новом состоянии -->
        <eventReport>
          <eventField>FrequencyInformation</eventField>
          <eventSubscript>_FrequencyIndex_</eventSubscript>
          <eventField>FrequencyState</eventField>
        </eventReport>
      </eventReports>
    </event>
    <event eventID="2">
      <name>CreatedFrequency</name>
      <synopsis>Обнаружена новая частота</synopsis>
      <eventTarget>
        <eventField>FrequencyInformation></eventField>
        <eventSubscript>_FrequencyIndex_</eventSubscript>
      </eventTarget>
      <eventCreated/>
      <eventReports>
        <eventReport>
          <eventField>FrequencyInformation</eventField>
          <eventSubscript>_FrequencyIndex_</eventSubscript>
          <eventField>LaserFrequency</eventField>
        </eventReport>
      </eventReports>
    </event>
    <event eventID="3">
      <name>DeletedFrequency</name>
      <synopsis>
          Удалена запись в таблице частот
      </synopsis>
      <eventTarget>
        <eventField>FrequencyInformation</eventField>
        <eventSubscript>_FrequencyIndex_</eventSubscript>
      </eventTarget>
      <eventDeleted/>
     </event>
    <event eventID="4">
      <name>PowerProblem</name>
      <synopsis>Проблемы с уровнем мощности лазера</synopsis>
      <eventTarget>
        <eventField>FrequencyInformation</eventField>
```

```
                <eventSubscript>_FrequencyIndex_</eventSubscript>
                <eventField>LaserPower</eventField>
              </eventTarget>
              <eventLessThan/>
              <eventReports>
                <eventReport>
                  <eventField>FrequencyInformation</eventField>
                  <eventSubscript>_FrequencyIndex_</eventSubscript>
                  <eventField>LaserPower</eventField>
                </eventReport>
                <eventReport>
                  <eventField>FrequencyInformation</eventField>
                  <eventSubscript>_FrequencyIndex_</eventSubscript>
                  <eventField>LaserFrequency</eventField>
                </eventReport>
              </eventReports>
            </event>
            <event eventID="5">
              <name>FrameCircuitChanged</name>
              <synopsis>
                  Изменилось состояние устройства FR на частоте
              </synopsis>
              <eventTarget>
                <eventField>FrequencyInformation</eventField>
                <eventSubscript>_FrequencyIndex_</eventSubscript>
                <eventField>FrameRelayCircuits</eventField>
                <eventSubscript>FrameCircuitIndex</eventSubscript>
                <eventField>CircuitStatus</eventField>
              </eventTarget>
              <eventChanged/>
              <eventReports>
                <eventReport>
                  <eventField>FrequencyInformation</eventField>
                  <eventSubscript>_FrequencyIndex_</eventSubscript>
                  <eventField>FrameRelayCircuits</eventField>
                  <eventSubscript>FrameCircuitIndex</eventSubscript>
                  <eventField>CircuitStatus</eventField>
                </eventReport>
                <eventReport>
                  <eventField>FrequencyInformation</eventField>
                  <eventSubscript>_FrequencyIndex_</eventSubscript>
                  <eventField>FrameRelayCircuits</eventField>
                  <eventSubscript>FrameCircuitIndex</eventSubscript>
                  <eventField>DLCI</eventField>
                </eventReport>
              </eventReports>
            </event>
          </events>
        </LFBClassDef>
      </LFBClassDefs>
    </LFBLibrary>
```

8.1. Обработка данных

Этот блок LFB создан для обработки пакетов данных, приходящих извне и уходящих наружу. Это неполный порт и он не поддерживает всей статистики, но показывает многие аспекты поведения. В следующих параграфах описывается возможное рабочее устройство и применение в нем этого определения LFB.

Пакеты, прибывшие без ошибок из физического интерфейса, попадают в Frame Relay DLCI на лазерном канале. Эти два значения используются LFB для поиска при обработке пакета. Если обработка показывает, что пакет относится к LMI, используется выходной индекс для выбора порта LFB в группе LMItoFE. Пакет передаётся в виде полного кадра FR (без вставки битов или байтов) в выбранный порт. Лазерный канал и DLCI передаются как метаданные, хотя DLCI имеется и в пакете.

Пакеты без ошибок, не относящиеся к LMI и имеющие индикатор типа IP, передаются как пакеты IP в порт группы DatatoFE с использованием того же поля индекса из таблицы на основе лазерного канала и DLCI, которые передаются как метаданные для последующего применения (например, классификации).

Текущее определение не указывает поведения для кадров, тип которых не является IP.

Пакеты, приходящие на входные порты сопровождаются метаданными для лазерного канала и DLCI. Поэтому может использоваться один входной порт. Со структурой, которая определена (параллельно выходной структуре) выбор канала и DLCI может быть ограничен группой входного порта (LMI и данные) и индексом порта. При другом устройстве LFB структура может требовать взаимно-однозначного соответствия между DLCI и портом LFB, когда метаданные станут ненужными. Однако это приведёт к усложнению. Промежуточный уровень структуры здесь обеспечивает параллелизм между входом и выходом, не требуя избыточных портов.

8.1.1. Организация DLCI

Когда элемент CE решает организовать DLCI на определённом лазерном канале, он передаёт запрос SET, направленный этому блоку LFB. Запрос имеет вид

```
T = SET
  T = PATH-DATA
    Path: flags = none, length = 4, path = 2, channel, 4, entryIdx
    DataRaw: DLCI, Enabled(1), false, out-idx
```

Это будет создавать DLCI с трафиком, идущим в конкретный порт выходной группы DatatoFE (CE будет гарантировать подключение этого порта к нужному месту до отправки запроса).

Отклик будет подтверждать создание запрошенной записи. Таблица структурирована для использования раздельных внутренних индексов и DLCI. В другом варианте может быть выбрано использование DLCI в качестве индекса, несмотря на сложности.

Можно представить, что FE имеет LMI LFB. Такие LFB будут подключаться к группам портов LMItoFE и LMIfromFE и обрабатывать информацию LMI. Задачей LFB может быть организация устройств Frame Relay. LMI LFB будет иметь псевдоним, указывающий на поддерживаемую таблицу устройств Frame Relay и сможет манипулировать этими записями.

8.1.2. Обработка ошибок

LFB будет иногда получать из линии непригодные пакеты. Многие из них будут просто увеличивать значения соответствующих счётчиков. Разработчик LFB может задать некоторые меры частоты ошибок. Это добавит работы FE, но сделает сигналы более осмысленными.

При некоторых ошибках может потребоваться передача части пакета элементу CE. Ошибка сама по себе не вызывает события в LFB. Имеется два способа решить задачу отправки информации CE.

Одним способом является определение специальной компоненты для учёта ошибок и компоненты LFB для хранения нужной части пакета. Можно определить компоненту для хранения части последнего пакета с ошибкой, затем определить событие, которое происходит при изменении счётчика ошибок и объявить, что сообщение о событии включает поле LFB с частью пакета. Для редких, но критически важных ошибок это будет эффективным решением, поскольку обеспечивается гарантия доставки уведомлений, а CE может указать, нужны ли ему эти уведомления.

Другой подход заключается в создании LFB с портом, подключённым к приёмнику перенаправления. LFB будет добавлять метаданные лазерного канала, DLCI и индикации ошибок, а затем доставлять пакет CE.

Другие аспекты обработки ошибок рассмотрены ниже.

8.2. Компоненты LFB

Этот блок LFB определён с компонентами верхнего уровня, отражающими административное состояние LFB, что позволяет элементу CE полностью отключить LFB.

Другой компонентой является таблица информации о лазерных каналах в виде массива с переменным размером. Каждый элемент массива содержит идентификатор длины волны, с которой связан этот элемент, рабочее состояние этой длины волны, мощность лазера на данном канале и таблицу устройств Frame Relay на этой длине волны. Здесь нет административного состояния, поскольку CE может отключить запись путём её удаления (длины волн и мощность лазера в неиспользуемых каналах практического интереса не представляют, информация о поддерживаемых длинах волн доступна в разделе возможностей).

Информация об устройстве (канале) Frame Relay содержит DLCI, рабочее состояние канала, использование канала для передачи информации LMI и и порт выходной группы LFB, в который передаётся трафик. Как отмечено выше, индекс устройства может в некоторых случаях комбинироваться с DLCI.

8.3. Возможности

Информация о возможностях для этого LFB включает рабочее состояние базового интерфейса, число поддерживаемых длин волн, число разрешённых локальных устройств для всех каналов. Максимальное число устройств для данного лазерного канала можно определить из таблицы FrameRelayCircuits. GET-PROP для пути 2.channel.4 будет давать CE свойства данного массива FrameRelayCircuits, которые включают число используемых элементов, первый доступный элемент и максимально разрешённое число элементов.

8.4. События

Этот блок LFB определён с возможностью генерации нескольких событий, которые могут быть интересны CE. Это уведомления об изменении рабочего состояния лазерных частот, а также создании и удалении частот (длин волн). Кроме того, поддерживаются уведомления о смене состояний отдельных устройств Frame Relay. Например, уведомление 61.5.3.11 будет говорить об изменении статуса устройства с индексом 11 в текущей таблице с индексом 3 таблицы длин волн. Уведомление о событии будет также указывать новое состояние устройства и DLCI. Возможно, DLCI является избыточным, поскольку CE предположительно знает DLCI по индексу устройства. Это поле указано здесь для демонстрации включения в отчёт о событии двух информационных элементов.

Как сказано выше, объявление события определяет его получателя, условия и содержимое уведомления. Свойства события показывают, подписан ли на него элемент CE, порог события и фильтры для этого события.

Другим типом событий являются проблемы с мощностью лазера. Эти события генерируются при падении мощности ниже заданного порога. Таким образом, CE может регистрировать событие потери мощности лазера на всех устройствах. Это может иметь вид

```
T = SET-PROP
  Path-TLV: flags=0, length = 2, path = 61.4
    Path-TLV: flags = property-field, length = 1, path = 2
      Content = 1 (register)
    Path-TLV: flags = property-field, length = 1, path = 3
      Content = 15 (threshold)
```

Это будет регистрировать событие для всех записей в таблице, а также устанавливать порог, в соответствии с которым событие будет генерироваться при падении мощности ниже 15 (предполагается, что CE знает единицы измерения мощности и поймёт это значение).

Если мощность лазера колеблется около значения 15, может возникать множество уведомлений (например при переходе между 14 и 15 каждый переход будет создавать уведомление). Предположим, что CE решает подавить эти осцилляции на канале 5. Это можно сделать путём задания гистерезиса, как показано ниже.

```
T = SET-PROP
  Path-TLV: flags=0, length = 3, path = 61.4.5
    Path-TLV: flags = property-field, length = 1, path = 4
      Content = 2 (hysteresis)
```

Установка гистерезиса 2 подавляет множество ненужных уведомлений. Когда уровень первый раз упадёт ниже 10, будет создано уведомление. Если мощность возрастёт до 10 или 11, а затем снова упадёт ниже 10, уведомления не будет. Если мощность поднимется по меньшей мере до 12, а затем упадёт ниже 10, снова будет создано уведомление. Одной из основных причин таких колебаний является фактическое значение мощности около границы. Если мощность составляет 9,5, незначительные её изменения будут приводить к скачкам между 9 и 10. Гистерезис со значением 1 будет подавлять ненужные уведомления. Другие события могут приводить к более сильным колебаниям и для них потребуется большее значение гистерезиса.

9. Взаимодействие с IANA

Модели ForCES требуется уникальное пространство имён XML для определений библиотеки ForCES, а также уникальные имена и числовые идентификаторы классов.

9.1. Регистрация пространства имён URN

Агентство IANA зарегистрировало новое пространство имён XML в соответствии с рекомендациями RFC 3688 [RFC3688].

```
URI: URI для этого пространства имён имеет значение
urn:ietf:params:xml:ns:forces:lfbmodel:1.0
Registrant Contact: IESG
XML: нет, но это пространство имён XML
```

9.2. Имена и идентификаторы классов LFB

Чтобы иметь чётко определённые классы ForCES LFB и чётко определённые идентификаторы в этих классах агентство IANA создало реестр имён классов LFB, соответствующих идентификаторов классов и документов с определениями классов LFB. Значения в реестре выделяются в порядке подачи заявок (FCFS[1]) Идентификаторы классов LFB со значением меньше 65536 зарезервированы для IETF Standards-Track RFC. Идентификаторы с номерами 65536 и выше в 32-битовом пространстве доступны для выделения в порядке поступления запросов. Все записи реестра должны быть документированы в стабильной форме с открытым доступом.

Поскольку в реестре предусмотрена процедура FCFS для части пространства идентификаторов, нужно определить правила именования классов, использующих идентификаторы из этого пространства. Поскольку они могут определяться любым путём, задаётся лишь требование отсутствия совпадений имён классов с именами классов, определёнными IETF. По этой причине все классы FCFS **должны** начинаться с префикса «Ext-».

Агентство IANA создало реестр ForCES LFB Class Names и соответствующих идентификаторов ForCES LFB Class Identifiers как показано в таблице.

Имя класса LFB	*Идентификатор класса LFB*	*Документ*	*Описание*
Резерв	0	RFC 5812	Резерв
FE Object	1	RFC 5812	Определяет информацию ForCES FE
FE Protocol Object	2	RFC 5810	Определяет параметры операций ForCES
LFB, определяемые IETF	3 - 65535	Standards Track RFC	Зарезервированы для определяемых IETF RFC
Имена классов ForCES LFB с префиксом EXT-	> 65535	Любой документ с открытым доступом	Обслуживание в порядке очерёдности для любых целей

10. Почётные авторы

Ниже приведён список авторов, внёсших свой вклад в создание ранних версий этого документа.

Ellen Delganes, Intel Corp.

Lily Yang, Intel Corp.

Ram Gopal, Nokia Research Center

Alan DeKok, Infoblox, Inc.

Zsolt Haraszti, Clovis Solutions

11. Благодарности

Многие из наших коллег в компаниях и участников почтовой конференции ForCES внесли важный вклад в эту работу. Особая благодарность Evangelos Haleplidis за помощью с XML.

12. Вопросы безопасности

Модель FE описывает организацию и представление наборов данных и компонент в FE. Базовый документ ForCES [RFC3746] включает всеобъемлющий анализ безопасности архитектуры ForCES в целом. Например, объекты протокола ForCES должны быть аутентифицированы в соответствии с требованиями ForCES до того, как они смогут получить доступ к описанной в этом документе информации по протоколу ForCES. Доступ к информации,

[1]First come, first served - первым пришел, первого обслужили.

содержащейся в модели FE, обеспечивается протоколом ForCES, который определён в отдельных документах, включающих рассмотрение вопросов безопасности.

13. Литература

13.1. Нормативные документы

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5810] Doria, A., Ed., Hadi Salim, J., Ed., Haas, R., Ed., Khosravi, H., Ed., Wang, W., Ed., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[Schema1] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, http://www.w3.org/TR/xmlshcema-1/, May 2001.

[Schema2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C REC-xmlschema-2, http://www.w3.org/TR/xmlschema-2/, May 2001.

13.2. Дополнительная литература

[RFC3654] Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", RFC 3654, November 2003.

[RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.

[RFC3317] Chan, K., Sahita, R., Hahn, S., and K. McCloghrie, "Differentiated Services Quality of Service Policy Information Base", RFC 3317, March 2003.

[RFC3318] Sahita, R., Hahn, S., Chan, K., and K. McCloghrie, "Framework Policy Information Base", RFC 3318, March 2003.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, January 2003.

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

[UNICODE] Davis, M. and M. Suignard, "UNICODE Security Considerations", http://www.unicode.org/reports/tr36/tr36-3.html, July 2005.

Адреса авторов

Joel Halpern

Self

P.O. Box 6049

Leesburg, VA 20178

USA

Phone: +1 703 371 3043

EMail: jmh@joelhalpern.com

Jamal Hadi Salim

Znyx Networks

Ottawa, Ontario

Canada

EMail: hadi@mojatatu.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru