Министерство образования и науки Украины
Харьковский национальный университет имени В.Н. Каразина

На правах рукописи

Дорожинский Владимир Владимирович

УДК 004.042/519.713.2

Математические модели для спецификации и анализа компонентов событийно-управляемых систем

01.05.02 — математическое моделирование и вычислительные методы

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
Жолткевич Григорий Николаевич,
доктор технических наук, профессор

Харьков — 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

РАЗДЕЛ 1 Обработка событий 13

1.1 Системы управляемые событиями 13

1.2 Системы обработки сложных событий 19

1.3 Сферы применения систем обработки сложных событий . . 20

1.4 Способы обработки сложных событий 21

1.5 События и машинное обучение 25

1.6 Применение событийно управляемых систем в сетях 26

Выводы по разделу 1 29

РАЗДЕЛ 2 Математические модели обработки сложных событий 30

2.1 Базовые понятия и обозначения 30

2.2 Сравнительный анализ обработки простых и сложных событий 32

2.3 Формальная модель системы обработки сложных событий . 35

2.4 Основные свойства CEP-машин 37

2.5 CEP-машины и предавтоматы 41

2.6 Синтез CEP-машин с заданным поведением 45

2.7 Обработка сложных событий и вычислимость 48

Выводы по разделу 2 51

РАЗДЕЛ 3 Системы обработки регулярных событий и машинное обучение 53

3.1 Базовые обозначения и определения 53

3.2 Регулярные обработчики событий 55

3.3 Свойства регулярных обработчиков 57

3.4 Регулярные машины обработки сложных событий 58

3.5 Основные свойства регулярных машин 60
3.6 Метод обучения регулярной CEP-машины 62
3.7 Верификация правильности метода обучения 68
3.8 Имплементация эксперимента 70
Выводы по разделу 3 73

РАЗДЕЛ 4 Использование модели регулярной CEP-машины в задачах управления трафиком в распределенных информационных системах 77
4.1 Базовые понятия и обозначения 77
4.2 Анализ методов сетевого управления 79
4.3 Задача формирования трафика 85
4.4 Метод управления трафиком с помощью CEP-машины . . . 92
4.5 Пример имплементации 97
4.6 Эффективность метода 104
Выводы по разделу 4 107

ВЫВОДЫ 109

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 113

ПРИЛОЖЕНИЯ. 127

ВВЕДЕНИЕ

Актуальность темы. Необходимость обеспечения высокого уровня адаптируемости больших информационных и программных систем приводит к все более широкому использованию архитектурных решений основывающихся на концепции событийного управления. Управление такого типа предоставляет достаточную гибкость системе, но в тоже время такой подход приводит к необходимо асинхронному взаимодействию компонентов системы. В случае когда каждое входное сообщение, допускаемое компонентами системы, несет в себе всю необходимую информацию для генерации ответа системы, автоматная модель является лучшим способом спецификации поведения системных компонентов. В противоположность такому подходу, обычно, отдельные сообщения не определяют семантические значения управляющих команд: такие значения управляющих команд определяются уникальными последовательностями сообщений. В таком случае, теория автоматов не обладает достаточной выразительностью, необходимой для спецификации поведения компонентов системы. Следовательно, разработка моделей для спецификации и анализа классов таких систем в настоящее время является актуальной задачей.

Взаимосвязь работы с научными программами, планами, темами. Диссертационная работа выполнена в соответствии с планом научно-исследовательских работ Харьковского национального университета имени В.Н. Каразина в рамках темы "Математическое и компьютерное моделирование информационных процессов в сложных естественных и технических системах"(номер государственной регистрации 0112U002098).

Цель и задание исследования. Целью работы является разработка класса математических моделей направленная на обеспечение возможности спецификации, моделирования структуры и поведения компонентов информационных событийно-управляемых программных систем, а также разработка вычислительных методов анализа таких моделей, что позволяет расширить сферу применения методов статического анализа при разработке и реинжиниринге таких систем и, за этот счет, обеспечить совершенствование процесса их проектирования.

Для достижения этой цели в работе поставлены и решены следующие задачи:

1. Проведен анализ проявившихся в теории и практике программной инженерии тенденций, состоящих в изменении концепций разработки архитектурных решений больших систем обработки информации. Это позволило сделать вывод о расширении использования асинхронных архитектурных решений основанных на событийном управлении, что обосновывает актуальность задачи совершенствования математических моделей, используемых для статического анализа, спецификации и проектирования систем этого класса.

2. Проведен анализ существующих событийно-управляемых архитектурных стилей с целью выявления их инвариантных свойств, которые должны отражаться в общей системной модели событийно-управляемой системы обработки информации. По результатам проведенного анализа построена общая системная модель компонента событийно-управляемой системы обработки информации.

3. Исследованы свойства потоков сложных событий на предмет возможности распознавания в них заданных множеств сложных событий.

4. Обосновано применение абстрактных предавтоматов для математического моделирования компонентов событийно-управляемых систем обработки информации путем доказательства двойственности абстрактных пре-

давтоматов и CEP-машин.

5. Рассмотрен вопрос алгоритмической реализуемости CEP-машин. Выявлены возможные аномалии вычислимости таких машин и предложен метод их устранения.

6. Разработан специальный класс CEP-машин распознающих сложные события описываемые регулярными языками. А также, на базе технологий машинного обучения, построен метод их синтеза

7. Разработаны прототипы программных утилит статического анализа, обеспечивающих верификацию программных компонентов событийно-управляемых систем обработки информации, которые реализуют предложенные и обоснованные в работе вычислительные методы.

8. Проведена проверка прототипов программных утилит статического анализа путем их использования при разработке программного обеспечения.

Объектом исследования являются информационные процессы в событийно-управляемых системах обработки информации.

Предметом исследования являются математические модели и вычислительные методы статического анализа программных компонентов событийно-управляемых систем обработки информации.

Методы исследования. В работе применены методы теории автоматов, общей алгебры, теории графов, теории вероятностей, системного анализа, математической логики.

Научная новизна диссертационной работы. Научная новизна полученных результатов заключается в том, что в работе решена важная научно-прикладная задача – совершенствование моделей и развитие методов статического анализа для событийно-управляемых систем обработки информации. Это позволило усовершенствовать процесс проектирования программного обеспечения систем этого класса за счет расширения в ходе разработки

сферы применения формальных методов анализа проектных решений, при этом

1. Впервые предложен класс математических моделей компонентов событий управляемых систем – машин обработки сложных событий (CEP-машин), который в отличие от существующих классов моделей, базируется не на конечных автоматах, а на их обобщении – конечных предавтоматах. Это обобщение позволило проводить анализ системных компонентов способных обрабатывать не только атомарные события, но и потоки сложных событий, сформулировать структурные и поведенческие свойства CEP-машин, доказать двойственность машин обработки сложных событий и предавтоматов и, как следствие, изучать поведение машин обработки сложных событий опираясь на свойства предавтоматов.

2. Впервые предложен специальный класс обработчиков сложных событий – регулярных акцепторов, и на его основе разработан метод машинного обучения, который в отличии от существующих аналогов, дает возможность существенно упростить процесс анализа возможных комбинаций событий при разработке компонентов событийно-управляемой системы, что позволило значительно упростить процесс проектирования таких систем.

3. Усовершенствован критерий оценки возможности обработки потоков сложных событий, что позволило делать выводы о возможности распознавания и обработки тех или иных сложных событий.

4. Получил дальнейшее развитие важный для приложений метод решения задачи синтеза обработчиков событий с заданным поведением, который был обобщен на случай предавтоматных моделей. Что позволило, задавая поведение с помощью функции отклика, синтезировать машину обработки сложных событий реализующую это поведение.

Практическое значение полученных результатов. Практическая ценность работы заключается в том, что все теоретические разработки доведены автором до конкретных инженерных методик, алгоритмов и реализованы в виде прототипов программных средств, базирующихся на исследовании построенных в работе математических моделей и использовании разработанных вычислительных методов. В частности, реализован пакет программных утилит анализа и синтеза машин обработки сложных событий, включающий:

1. Утилиту синтеза регулярного обработчика сложных событий основанного на множествах примеров и контрпримеров сложных событий. Синтезированный обработчик допускает все сложные события из множества примеров и отвергает все сложные события из множества контрпримеров.

2. Утилиту, реализующую метод машинного обучения синтезированного на основе множеств примеров и контрпримеров регулярного обработчика.

3. Генератор регулярных выражений, необходимый для построения тестовых регулярных обработчиков.

4. Утилиту анализа глубины и дисбаланса синтаксического дерева регулярных выражений.

Практическая ценность работы подтверждается также использованием ее результатов при выполнении проектов разработки программного обеспечения в компаниях разрабатывающих распределенные информационные системы. Акты об использовании результатов работы приведены в приложениях к диссертации.

Личный вклад соискателя. Все результаты диссертационной работы получены автором самостоятельно. Они изложены в шести работах, пять из которых опубликованы в изданиях, рекомендованных для публикаций результатов диссертационных работ, а одна – в материалах международных

конференций и семинаров. В этих работах автору принадлежат следующие результаты:

1. Проведен анализ существующих моделей обработки сложных событий и предложен набор инвариантов, обобщение которых привело к формулировке математической модели компонента системы обработки сложных событий [1].

2. Описан класс регулярных машин обработки сложных событий, а также исследованы их свойства [2].

3. Предложен метод практического применения регулярных CEP-машин при разработке средств управления и оптимизации трафика в распределенных информационных системах [3].

4. Разработан алгоритм обучения регулярного акцептора [4].

5. Изучены сферы практического применения машинного обучения регулярных акцепторов [5].

6. Разработаны алгоритмы генерации тестовых регулярных акцепторов, использующихся при верификации правильности обучающего метода [6].

Апробация результатов исследований. Основные результаты диссертации докладывались на:

1. X Международная конференция ICTERI-2014 "Information Communication Technologies in Education, Research and Industry"(Херсон, 2014);

2. XII Международная конференция КУСС "Контроль и Управление в Сложных Системах"(Винница, 2014);

3. XXI Всеукраїнська наукова конференція "Сучасні проблеми прикладної математики та інформатики"(Львів, 2015);

4. XII Международная конференция ICTERI-2016 "Information Communication Technologies in Education, Research and Industry"(Киев, 2016).

Публикации. По результатам научных исследований опубликовано 5 научных статей в изданиях, входящих в перечень изданий, рекомендованых для публикаций результатов диссертационных исследований, и 3 доклада на международных научных конференциях.

Структура и объем работы. Диссертация состоит из введения, четырех разделов, заключения, списка использованных источников и приложений. Полный объем диссертации составляет 131 страницу, в том числе: основной текст на 112 страницах, содержит 7 рисунков, список использованных источников из 139 наименований на 14 страницах и 2 приложения на 4 страницах.

Дадим краткую характеристику работы.

— В первом разделе проводится детальный обзор современных тенденций развития в области информационных технологий. В частности, проводится анализ развития архитектур компонентов информационных систем основанных на событийно-управляемом подходе. Приводится неформальное определение понятия "событие", рассматриваются все возможные типы событий, а также способы их обработки. Показывается преимущество событийно-управляемых архитектурных решений по сравнению с последовательной пакетной обработкой информации. Проводится обзор существующих подходов к обработке сложных событий и возможные сферы их применения. Также проводится обзор взаимосвязи концепций обработки событий и методов машинного обучения. Показывается, что эти две концепции могут комбинироваться при усовершенствовании методов проектирования больших распределенных событийно-управляемых систем. Заключительная часть раздела содержит обзор примеров использования событийно-управляемых систем в современных телекоммуникационных сетях использующих облачные технологии.

— Второй раздел посвящен построению математической модели компонен-

формальные понятия и определения основанные на анализе инвариантов существующих подходов к обработке сложных событий. Приводится формальное определение простого и сложного события и проводится сравнительный анализ их обработки. Далее, в разделе приводится формальная модель системы обработки сложных событий. Описывается ее структура, поведение, а также свойства. Приводится характиристика потоков сложных событий, которые могут быть обработаны данной моделью. Затем, в разделе доказывается двойственность предложенной формальной модели обработки сложных событий и концепции предавтоматов. Далее, используя факт двойственности предложенной в разделе модели и предавтомата, рассматривается важный для инженерных приложений вопрос синтеза компонентов системы обработки сложных событий основанных на заданном поведении. Заключительная часть раздела посвящена вопросу вычислимости и алгоритмической реализуемости предложенной модели обработки сложных событий. Доказывается критерий определяющий возможность алгоритмической реализуемости в зависимости от вопроса вычислимости отдельных обработчиков сложных событий.

— В третьем разделе рассматривается специальный класс моделей систем обработки сложных событий, работающих с потоками сложных событий, описываемыми регулярными языками. В начале, приводятся базовые понятия и обозначения используемые в разделе. Далее, детально описывается специальный класс регулярных обработчиков сложных событий, формирующих модель системы обработки регулярных сложных событий. Изучаются свойства таких обработчиков. В частности, доказывается возможность их реализации конечными детерминированными автоматами. Затем, в разделе рассматриваются модели обработки сложных регулярных событий, формируемые регулярными обработчиками. Приводится алгоритм работы модели обработки сложных регулярных событий и ее свойства. В частности, доказывается возможность преобразования модели обработки сло-

жных регулярных событий в автоматную путем глобализации двойственной предавтоматной модели. Дальнейшая часть раздела посвящена методу машинного обучения регулярных обработчиков. Показывается что такой подход может значительно упростить процесс проектирования событийно-управляемых систем. Затем, в разделе детально описывается компьютерный эксперемент, верифицирующий правильность обучающего метода. Заключительная часть раздела посвящена способам имплементации данного эксперемента.

— Четвертый раздел посвящен практическому применению модели обработки сложных регулярных событий в средствах управления трафиком в распределенных информационных системах. Как обычно, в начале раздела приводятся необходимые понятия и обозначения. Далее, приводятся существующие способы управления сетями. Затем, рассматривается задача формирования трафика в сетях, решение которой является одним из ключевых факторов улучшения производительности распределенных информационных систем. Дальнейшая часть раздела посвящена описанию способа применения модели обработки сложных регулярных событий для решения задачи формирования трафика в узле сети. Затем, в разделе приводится пример имплементации телекоммуникационной сети, использующий описанное выше решение. В частности, показывается, что использование облачных технологий значительно расширяет возможность использования предложенного решения задачи формирования трафика. В заключительной части раздела обсуждается эффективность предложенного метода, а также приводятся результаты сравнительного тестирования производительности сети, основанного на критериях оценки качества обслуживания в сетях.

— Завершают работу общие выводы, список использованных источников и приложения.

РАЗДЕЛ 1
ОБРАБОТКА СОБЫТИЙ

В этом разделе диссертационной работы проводится анализ тенденций, проявившихся в практике применения систем управляемых событиями. Среди них особое место занимают системы обработки сложных событий, которые получают широкое распространение в настоящее время. Такие системы базируются на идее наблюдения и анализа непрерывных потоков информации. Эти процессы требуют выполнения "на лету", следовательно, должны генерировать ответ системы в режиме реального времени. Как правило, непрерывный поток информации не является непрерывным в математическом смысле этого слова [7, 8]. Такой поток может рассматриваться скорее как последовательность простых (атомарных) событий, которые содержат фрагменты информации. Конечный сегмент потока событий является сложным событием если он имеет семантический смысл в рассматриваемой предметной области. В связи с этим, в разделе рассматриваются известные способы обработки сложных событий, их применения, а также возможные аномалии.

1.1 Системы управляемые событиями

Термин "событие"в общем случае может быть охарактеризован как интересующее явление [9]. События, обычно возникающие асинхронно, могут различаться по размеру и сложности, от простых аппаратных прерываний до сложных обновлений баз данных [10, 11, 12]. Следует отметить, что управление и обработка событий является известным и широко применяемым методом во многих сферах информационных технологий, где дополня-

ет синхронные способы вычисления или даже является нормой [13, 14, 15]. Рассмотрим примеры систем управляемых событиями и отметим их особенности.

— Прерывания.

Аппаратные прерывания [16, 17] являются низкоуровневыми событиями, которыми устройство ввода/вывода уведомляет процессор о необходимости обработки входных/выходных данных. Обычно это случай когда устройство ввода/вывода завершило операцию и либо входные данные теперь могут быть обработаны процессором, либо устройство ввода/вывода завершило текущую операцию вывода данных и ожидает новых данных на вывод [18]. Распознав прерывание, процессор вызывает его обработчик в котором выполняются необходимые действия по обработке события ввода/вывода. После успешной обработки прерывания процессор возвращается из состояния обработки прерывания в состояние выполнения текущих операций. В случае неиспользования прерываний процессору пришлось бы периодически опрашивать устройства для определения необходимости обработки запросов от них. Обычно это очень неэффективный подход т.к. в большинстве случаев обработка не требуется и опрос устройства только расходует ценные процессорные циклы. Современные операционные системы также используют прерывания для переключения контекста с помощью которого они эффективно управляют несколькими задачами, достигая эффекта паралельного выполнения [19]. В этом смысле, применение прерываний делает возможным переход от систем пакетной обработки к современным интерактивным системам.

— Графические интерфейсы пользователя.

Графические интерфейсы пользователя (GUIs) [20, 21] по своей сути являются событийно управляемыми. Входные данные поступают асинхронно в форме событий, инициируемых пользователем, таких как нажатие клавиш, кликов мыши или прикосновением к тачскрину. Перед обработкой при-

ложением, такие события предварительно обрабатываются библиотеками GUIs. С этой целью, инструментарий GUI позволяет разработчикам программного обеспечения создавать универсальные интерфейсы состоящие из элементов управления таких как строки меню, поля ввода или кнопки. Вначале GUI библиотека проверяет к какому элементу управления инициированое пользователем событие может быть применено, например, какой пункт меню или поле ввода было выбрано, или какая кнопка была нажата соответственно. Затем, вызывается соответствующий обработчик события, который предварительно был указан разработчиком для такой комбинации события и элемента управления. Затем выполняется обработка входных данных, которая может производиться как непосредственно в обработчике события, так и через передачу управления специальной функции приложения. Таким образом, GUIs используют события и их обработчики для гибкой передачи визуального управления функциям приложения, в то время как, и GUI элементы, и логика приложения остаются сокрытыми в своих компонентах, тем самым улучшая их повторное использование.

— Активные базы данных.

Активные базы данных являются дополнением стандартных систем управления базами данных [22], которые обеспечивают возможность определения поведения системы в ответ на изменение данных [23]. С этой целью часто используются событие-условие-действие (ECA) правила для спецификации поведения системы [24, 25]. Событие определяет какое правило необходимо применить, в то время как условие позволяет определить контекст появления соответствуещего события. Если согласование события и условия произошло, то выполняется соответствующее действие. В случае реляционных баз данных, событием может быть, например, вставка или удаление кортежа, обновление атрибута, или начало трансакции, ее завершение или откат [26]. Условия состоят из предикатов над атрибутами и запросами, которые активируют механизм событий при изменении данных, хранящихся в базе

данных. Действия, обычно, не ограничены только операциями над базой данных, а могут также включать выполнение внешних процессов и приложений [27]. Более того, выполнение действий может привести также к активизации новых событий и выполнению соответствующих правил обработки. Следовательно, даже сложные бизнес потоки данных, требующие нескольких шагов обработки, могут быть соответствующим образом отражены и реализованы с помощью активных баз данных. Таким образом, они позволяют эффективно отслеживать хранимые данные на предмет интересующих событий, и в случае их возникновения, реагировать на них в реальном времени. По причине экономической важности, почти все современные большые системы управления базами данных включают в себя также поддержку активных баз данных.

— Издатель-подписчик системы. (Publish-Subscribe Systems)

Издатель-подписчик [28] является парадигмой группового взаимодействия позволяющая распределенным компонентам взаимодействовать посредством публикации уведомлений о произошедших событиях и подписки на интересующие уведомления. Поскольку подписка обычно определяет тип или содержание уведомлений, которые отслеживает тот или иной компонент, то взаимодействие между компонентами, публикующими уведомления и отслеживающими их, требует только слабого связывания. Действительно, компонентам, публикующим уведомления, не требуется "знать"своих подписчиков (обратное тоже верно). Такой подход позволяет легко расширять систему добавляя новые компоненты. Для работы такой системы обычно используется служба уведомлений (notification service), которая передает опубликованные уведомления подписчикам. В крупномасштабных проектах служба уведомления часто состоит из набора взаимодействующих посредников (brokers), которые обмениваются опубликованными уведомлениями и существующими подписками [29]. Уведомления поэтапно направляются через сеть посредников, дублируются при необходимости, и затем до-

ставляются подписчикам. Таким образом, гарантируется быстрая доставка уведомлений о событиях, в то время как компоненты, публикующие уведомления, освобождены от необходимости прямого взаимодействия с подписчиками, а подписчики не должны периодически запрашивать обновления. Следовательно, службы уведомления и издатель-подписчик системы очень удобны и, следовательно, широко используются в приложениях, которые требуют эффективного распространения информации среди большого количества клиентов и компонентов.

Приведенные примеры показывают, что события используются во многих сферах вычислений, включая аппаратные и программные системы, а также локальные и распределенные приложения. Многочисленные требования и задачи приводят к различным практическим реализациям. Но, в большинстве областей применения, основным предназначением событий является эффективное уведомление компонентов системы об интересующих изменениях, что позволяет отвечать на эти изменения в реальном времени. В вычислительных инфраструктурах, растущих в размере и сложности, оба аспекта приобретают большую важность. С одной стороны, растущее количество источников данных и вычислительных узлов, требующих эффективного распределения данных, делает периодические запросы обновлений неустойчивыми в крупномасштабных проектах. С другой стороны, такие инфраструктуры должны быть способны к все большей самоорганизации и самоуправлению, т.к. рост их сложности часто затрудняет их эксплуатацию и администрирование. Следовательно, необходимо чтобы системы и приложения были способны быстро реагировать на обнаруженные изменения, например, для адаптации и оптимизации своей конфигурации или активной поддержки и помощи пользователям. Таким образом, можно предвидеть что системы и приложения основанные на событиях будут играть основную роль в будущих вычислительных инфраструктурах. В частности, издатель-подписчик системы идеально подходят для гибкого со-

единения компонентов управляемых событиями приложений и подсистем в распределенных инфраструктурах [30]. Т.к. взаимодействие происходит не явно, а только посредством публикации и подписки на уведомления о возникающих событиях, существует только слабое связывание между взаимодействующими компонентами, поскольку требуется только согласование содержимого уведомлений. С одной стороны, новые компоненты могут легко быть добавлены в систему без необходимости изменения уже существующих и, следовательно, обеспечивается поддержка естественного расширения вычислительных инфраструктур в процессе их роста. С другой стороны, слабое связывание делает сложным определение реальных взаимодействий между компонентами, особенно в случаях использования, выходящих за рамки только обмена данными. На практике, например, почти невозможно определить все результаты и побочные эффекты, которые могут быть вызваны публикацией того или иного уведомления в системе, т.к. любой компонент системы может быть затронут этим изменением. Следовательно, без дальнейшего структурирования, обычные издатель-подписчик системы ограничены в использовании по причине возростания сложности, приводящей к взаимозависимости между компонентами. Для улучшения возможности управления взаимозависимостями между компонентами, инфраструктуры основанные на событиях требуют структурных абстракций позволяющих проектировать и разрабатывать издатель-подписчик системы и событийно управляемые приложения модульными. Хотя исследования в области программной инженерии показывают важность и выгоды концепций структурирования таких как модули [31], классы [32], и компоненты [33], несмотря на это, подобные концепции очевидно отсутствуют для инфраструктур основанных на событиях. Следовательно, разработка математических моделей, а также концепций структур для описания событийно-управляемых систем является актуальной задачей решаемой в работе.

1.2 Системы обработки сложных событий

Под термином обработка сложных событий (Complex Event Processing) подразумевают метод наблюдения и анализа потоков данных о происходящих событиях [34, p.3] и результатов их обработки. Целью метода обработки сложных событий является обнаружение значимых событий (таких как возможности или угрозы) и быстрая реакция на них [35, 36]. Концепции временных последовательностей и непрерывной обработки являются важными для пояснения необходимости нового класса систем [37]. Действительно, традиционные системы управления базами данных (Data Base Management Systems):

— требуют сохранения и индексации данных прежде чем они могут быть обработаны;

— обрабатывают данные только по явному запросу от пользователей, например, асинхронно по отношению к их поступлению.

Оба аспекта противоречат требованиям к современным системам наблюдения [38]. Для выполнения этих требований используются две основные концепции:

— Модель обработки потоков данных (Data Stream Processing Model) [39] представляет задачу обработки потока информации как обработку потоков данных, приходящих из различных источников, и генерирует новые выходные потоки данных. В такой модели задача обработки данных рассматривается как расширение традиционных систем управления базами данных. Следовательно, системы управления потоками данных (Data Stream Management Systems) происходят от систем управления базами данных, но имеют существенные отличия [40]. В то время как традиционные DBMS спроектированы для работы со стабильными данными, где обновления относительно редки, DSMS предназначены для работы с временными данными, которые непрерывно обновляются. Подобным образом как DBMS выпол-

няют запросы чтобы получить полный ответ, DSMS тоже выполняют непрерывные запросы и получают ответы по приходу новых данных;

— Модель обработки сложных событий (Complex Event Processing Model) [41] рассматривает элементы потока информации как уведомления о событиях, происходящих во внешнем мире, которые должны быть отфильтрованы и сгруппированы для определения происходящего в терминах событий более высокого уровня [42]. Следовательно, задачей этой модели является определение возникновения шаблонов событий более низкого уровня, которые представляют собой события более высокого уровня, чье возникновение отслеживается системой [38].

1.3 Сферы применения систем обработки сложных событий

Сложные события могут происходить в различных сферах деятельности таких как: исследование потребительского рынка, служба заказов или поддержки клиентов; финансовые системы, операции и услуги [43]; выявление мошенничества [44]; здравоохранение [45]. Также это могут быть новостные информационные сообщения [46], текстовые сообщения, публикации в социальных сетях, показатели биржевых индексов, информация о загруженности автомобильных дорог, прогнозы погоды или другие виды данных [34].

Важные сферы применения метода обработки сложных событий [47]:

— Мониторинг деловой активности используется с целью распознавания проблем или возможностей на ранних стадиях делового процесса или использования других важных ресурсов. На данном этапе события объединяются в так называемые ключевые показатели, например, среднее время выполнения задачи;

— Контрольно-измерительные сети передающие полученные из окружающей среды данные, например, на диспетчерский пункт или в систему сбо-

ра данных, которые контролируют производственные параметры. С целью уменьшения измерений и связанных с этим ошибок, данные различных датчиков часто должны комбинироваться. Затем, более общие события (например, пожар) обычно получаются из первичных численных измерений (например, температура, задымленность, и т.д.);

— Рыночные показатели такие как цены на сырье или потребительские товары, можно также рассматривать как события. Они должны непрерывно анализироваться в режиме реального времени с целью распознавания трендов и быстрой реакции на них в автоматическом режиме, например, в алгоритмической торговле [48].

1.4 Способы обработки сложных событий

Сложные события это сочетание нескольких простых событий. Следовательно, с помощью метода обработки сложных событий можно получить результаты базирующиеся на взаимосвязи (корреляции) простых событий. Для распознавания сложных событий было разработано множество языков и формализмов, так называемых языков запросов событий (Event Query Languages). В настоящее время используются такие языки запросов событий [47]:

— Язык операторов композиции (Composition Operators) строят запросы сложных событий из запросов простых событий, используя операторы композиции. Запросы сложных событий выражаются через композицию одиночных событий, используя различные операторы композиции. Типичными операторами являются коньюнкции событий (все события должны произойти, возможно, в различное время), последовательности (все события происходят в указанном порядке), и отрицания внутри последовательностей (событие не должно происходить во время между двумя другими событиями);

— Языки запросов потоков данных (Data Stream Query Languages) основываются на языке запросов SQL, а также на следующей идее: потоки данных передают события, представленные в виде кортежей. Каждый поток данных соответствует в точности одному типу события. Такие потоки данных преобразуются в отношения, которые по существу содержат (частично) кортежи, полученные на данном этапе. В таких отношениях вычисляются обычные (почти) SQL запросы. Полученный результат (другое отношение) затем преобразуется обратно в поток данных;

— Порождающие правила (Production Rules) очень удобны и хорошо интегрируются в существующие языки программирования. Однако, их использование приводит к низкому уровню абстракции т.к. порождающие правила основаны на состояниях, а не на событиях. Это отличает их от других языков запросов событий. В особенности затруднено агрегирование и отрицание. Порождающие правила считаются менее эффективными чем языки запросов потоков данных. Однако, это связано с гибкостью, которую они дают, в условиях комбинирования запросов (в условиях правила) и реакции (в действиях правила);

— Временные автоматы (Timed State Machines) обычно используются для моделирования поведения систем сохраняющих состояния (Stateful Systems), которые реагируют на события. Система моделируется как направленный граф. Узлы графа представляют возможные состояния системы. Направленные ребра отмечаются событиями и их временными условиями. Ребра определяют переходы между состояниями, которые происходят как реакция на входные события. Формально временные автоматы базируются на детерминированных или недетерминированных конечных автоматах. Поскольку состояния во временных автоматах являются достижимыми той или иной последовательностью событий, происходящих во времени, они неявно определяют сложные события;

— Логические языки (Logic Languages) выражают запросы событий в виде

логических формул. Логические языки предлагают естественный и удобный путь описания запросов событий. Основное преимущество логических языков их строгое формальное основание, пренебрегаемое другими языками запросов событий. Благодаря разделению различных аспектов обработки событий, логические языки являются очень выразительными, расширяемыми и легкими в изучении и использовании.

Метод обработки сложных событий использует множество подходов, [49] включая:

— Определение шаблонов событий (Event-Pattern Detection);

— Обобщение событий (Event Abstraction);

— Фильтрация событий (Event Filtering);

— Агрегация и преобразование событий (Event Aggregation and Transformation);

— Моделирование иерархий событий (Modeling Event Hierarchies);

— Определение взаимосвязей (Detecting Relationships), таких как причинно-следственная связь, принадлежность или синхронность, между событиями;

— Обобщение событийно-управляемых процессов (Abstracting Event-Driven Processes).

Как правило, большинство способов обработки событий базируются на автоматных моделях. Действительно, теория автоматов предлагает множество способов распознавания потоков данных и способов управления.

Типы автоматов. Автоматы часто подразделяют на:

1. автоматы-преобразователи (Transducers),
2. акцепторы (Acceptors),
3. автоматы-классификаторы (Classifiers) ,
4. секвенсоры (Sequencers).

Теперь рассмотрим каждый тип более детально.

— Акцепторы, иногда называемые распознавателями (recognizers) или де-

текторами последовательностей (sequence detectors), генерируют бинарный ответ, показывающий, является ли входная последовательность допустимой. Каждое состояние такого автомата обязательно является либо допускающим, либо недопускающим. Если после обработки всей входной последовательности текущее состояние автомата является допускающим, то входная последовательность допускается, иначе входная последовательность отвергается. Как правило, входная последовательность – это серия символов, а не набор действий.

— Автомат-классификатор является обобщением акцептора, генерирующий единственный ответ при завершении работы, но имеющий более двух терминальных состояний.

— Автоматы-преобразователи генерируют выход основываясь на заданной входной последовательности и на состоянии с привязанным к нему действием. Такие автоматы обычно используются в управляющих приложениях и в области вычислительной лингвистики. В управляющих приложениях различаются два вида автоматов:

1. Автомат Мура (Moore machine). Данный автомат использует только внутренний набор действий. Например, выход автомата зависит только от его текущего состояния. Преимуществом автомата Мура является возможность упрощения поведения управляющей системы.
2. Автомат Мили (Mealy machine). Набор действий такого автомата зависит от входных символов. Например, генерируемый выход зависит от входного символа и текущего состояния. Использование автомата Мили обычно приводит к уменьшению числа возможных внутренних состояний.

— Секвенсоры или генераторы – это автоматы имеющие односимвольный входной алфавит. Такие автоматы генерируют только одну последовательность, которая может интерпретироваться как выходная последовательность автомата-преобразователя, либо выходы автомата-классификатора.

Конечные автоматы используются для моделирования вычислительных процессов во многих предметных областях: в инженерных приложениях, лингвистике, в системах искусственного интелекта и др. [50, 51]. Однако, поскольку, как правило, каждое входное сообщение (символ считываемый автоматом) несет в себе слишком мало информации для принятия решения, возможность обработки сложных событий (последовательностей простых событий) становится очень ограниченной. Для преодоления этого ограничения в работе предлагается математическая модель обработки сложных событий, которая базируется не на автоматном подходе, а на его обобщении – предавтоматах (см. раздел 2).

1.5 События и машинное обучение

В современном мире информационных технологий многие приложения управляются событиями в том смысле, что их действия зависят от произошедших событий. Эти приложения используются во многих предметных областях и очень полезны при обработке каких либо данных. Обычно дизаин таких систем обработки данных основывается на архитектуре управляемой событиями (Event Driven Architecture). Математические основы анализа поведения компонентов таких систем можно найти, например, в статьях [52, 1, 53]. Важный на практике класс детекторов событий для систем основанных на архитектуре управляемой событиями был определен и рассматривался в [2]. Несмотря на то что все типы событий как правило известны, в большинстве случаев не возможно предусмотреть все их комбинации на этапе дизайна и разработки системы. В результате обрабатывающая система, как правило, не способна сгенерировать правильный ответ на комбинации событий, которые не рассматривались на этапе разработки. Для преодоления этой проблемы в работе предлагается использовать методы машинного обучения, помогающие сделать систему адаптивной к

новым комбинациям событий на этапе выполнения (см. раздел 3). Машинное обучение занимается изучением и разработкой алгоритмов, которые могут самообучаться и прогнозировать используя входные данные [54]. Такие алгоритмы скорее контролируют построение модели используя множество обучающих примеров для создания управляемых данными прогнозов или принятия решений, а не жестко следуют фиксированным программным инструкциям. Машинное обучение используется в тех вычислительных задачах где разработка и программирование явных алгоритмов не эффективно и часто вообще не возможно. Также иногда задачи машинного обучения совпадают с задачами анализа данных в таком направлении, [55] где внимание концентрируется больше на предварительном анализе данных, известном как самообучение (unsupervised learning) [56].

1.6 Применение событийно управляемых систем в сетях

В настоящее время большинство информационных систем являются распределенными. Такой подход к их архитектуре приводит к проблемам связанным с возможностью эффективного управления такими системами. Поскольку возникает необходимость эффективной оптимизации трафика, проходящего через сеть соединяющую компоненты системы. Следовательно, другим направлением, где могут быть использованы событийно управляемые системы, являются сети (см. раздел 4). В частности сети основанные на концепции виртуализации сетевых функций (Network Function Virtualization, NFV), являющейся перспективным направлением развития телекоммуникационных сетей.

Приложения виртуализации сетевых функций. Стандарт виртуализации сетевых функций доказал свою популярность уже с момента своего появления. Существует множество примеров его применения в таких направлениях как:

— виртуализация базовых станций мобильной связи;

— платформа как сервис (platform as a service, PaaS);

— сети доставки контента (content delivery networks, CDN) [57].

Можно предвидить что технология виртуализации сетевых функций имеет значительные перспективы. Виртуализация сетевых функций, развернутая на стандартизированном оборудовании общего назначения, подразумевает снижение затрат на содержание и обслуживание, а также время на разработку и запуск новых продуктов и услуг. Многие известные производители сетевого оборудования заявили о поддержке NFV. Что в свою очередь, совпало с объявлениями о внедрении этой технологии основными поставщиками программного обеспечения, предоставляющих программные платформы производителям аппаратных компонентов. Однако, чтобы реализовать заявленные выгоды от виртуализации, поставщики сетевого оборудования интенсивно работают над улучшением информационной технологии виртуализации для достижения требуемых высокой доступности, масштабирования, производительности, эффективных возможностей управления сетью. Для минимизации общей стоимости владения (total cost of ownership, TCO), функциональность операторского класса (carrier-grade features) должна реализовываться с максимальной эффективностью. Что в свою очередь требует чтобы NFV решения эффективно использовали свободные вычислительные ресурсы для достижения доступности 99.999%,[58] а также обеспечивали предсказуемость вычислительной производительности.

Платформа виртуализации сетевых функций является фундаментом для достижения решений при разработке сетей операторского уровня. Данная платформа является программной, выполняющаяся на стандартном мульти-ядерном (многопроцессорном) оборудовании, а также сформированная с использованием программного обеспечения с открытым исходным кодом (open source software), которое включает в себя функциональность операторского уровня. Программное обеспечение платформы виртуализа-

ции сетевых функций отвечает за динамическое переназначение виртуальных сетевых функций необходимое в случае сбоев или изменения загруженности трафика, и следовательно, играет важную роль в достижении высокой доступности сервисов. Существует множество проектов занимающихся спецификацией, стандартизацией и продвижением NFV функциональности операторского уровня. Примерами таких проектов являются:

— ETSI NFV Proof of Concept [59],

— а также ATIS Open Platform for NFV Project [60].

Примеры компонентов виртуализации сетевых функций. Одним из ключевых компонентов платформ виртуализации сетевых функций является виртуальный коммутатор (vSwitch), отвечающий за установку соединения между виртуальными машинами (VM-to-VM), а также между виртуальными машинами и внешней сетью. Производительность виртуального коммутатора определяет ширину полосы пропускания виртуальных сетевых функций, а также эффективность затрат на NFV решения. Например, производительность стандартного открытого виртуального коммутатора (Open vSwitch, OVS) [61] имеет некоторые недостатки, которые должны быть устранены для соответствия требованиям NFVI (NFV Infrastructure) решений. В части 4.4 будет представлено решение данной проблемы, помогающее улучшить производительность виртуальных коммутаторов.

Принципы спецификации доступности сервисов в сетях. Технология виртуализации также меняет способ спецификации доступности сервисов, измеряемой и достигнутой в решениях основанных на виртуализации сетевых функций. Поскольку виртуальные сетевые функции все больше заменяют традиционное функционально-ориентированное оборудование, это приводит к переходу от доступности услуг на основе оборудования к доступности услуг основанной на многоуровневых сервисах. Виртуальные сетевые функции устраняют явную зависимость между специальным оборудо-

ванием и необходимой функциональностью, а следовательно, доступность услуг определяется доступностью VNF (Virtual Network Function) сервисов, а не физического оборудования. Поскольку технология NFV может виртуализировать широкий диапазон типов сетевых функций, и каждый тип функций может иметь свой ожидаемый уровень доступности, NFV платформы должны поддерживать широкий диапазон механизмов отказоустойчивости. Эта гибкость позволяет операторам сетей оптимизировать свои NFV решения таким образом, чтобы удовлетворять любым требованиям доступности виртуальных сетевых функций.

Выводы по разделу 1

В данном разделе был проведен обзор задач и приложений где такое понятие как событие играет ключевую роль. Было показано что в настоящее время существует множество подходов для моделирования систем основанных на событиях. Такие концепции, как архитектура управляемая событиями и обработка сложных событий, широко используются для решения прикладных задач непрерывной обработки данных в различных предметных областях. Также такое направление, как машинное обучение, может быть использовано при проектировании и разработке систем основанных на обработке сложных событий. Также показано что системы обработки событий могут использоваться в задачах управления сетями.

РАЗДЕЛ 2
МАТЕМАТИЧЕСКИЕ МОДЕЛИ ОБРАБОТКИ СЛОЖНЫХ СОБЫТИЙ

Этот раздел посвящен построению математической модели системы обработки сложных событий. Такую модель будем называть машиной обработки сложных событий (CEP-машиной). Также в разделе будет показано, что эта модель тесно связанна с понятием предавтомата. Предлагаемая модель предоставляет строгие формулировки задач связанных с обработкой сложных событий. Показывается, что возможность обработки событий такими системами определяется неким компромисом между классом обрабатываемого потока событий и количеством сложных событий. Далее, в разделе решается задача синтеза CEP-машины, основанного на формальной спецификации. Часть раздела посвящена анализу систем обработки сложных событий. Устанавливается условие гарантирующее возможность алгоритмической реализации для CEP-машин.

2.1 Базовые понятия и обозначения

Пусть X и Y некоторые множества и f частичное отображение из X в Y тогда запись $f : X \dashrightarrow Y$ используется для обозначения того что f частичное отображение, в отличии от записи $X \to Y$, которая используется для обозначения всюду определенных отображений.

Пусть $f : X \dashrightarrow Y$ частичное отображение из множества X во множество Y, и пусть x некоторый элемент множества X тогда запись $f(x)\downarrow$ используется для обозначения того что x принадлежит области определения отображения f. Более того, если x принадлежит области определения

отображения f и известно что $f(x) = y$ тогда запись $f(x) \downarrow = y$ используется для обозначения этого факта. Подобным образом, запись $f(x) \uparrow$ используется для обозначения того, что x не принадлежит области определения отображения f. Запись Df используется для обозначения области определения f.

Пусть Σ конечный алфавит. Тогда запись Σ^* используется для обозначения свободного моноида порожденного Σ. Единица этого моноида обозначается как ε. Полугруппа всех непустых слов над алфавитом Σ обозначается как Σ^+. Следовательно, $\Sigma^* = \Sigma^+ \bigcup \{\varepsilon\}$.

Для любого $u \in \Sigma^+$ обозначим как $|u|$ длину слова u, например количество букв содержащихся в u, и зафиксируем что $|\varepsilon| = 0$.

Любое множество L состоящее из слов алфавита Σ называется префиксным если утверждения $uv \in L$ и $u \in L$ для $u, v \in \Sigma^*$ влекут равенство $v = \varepsilon$. Для любого $L \subset \Sigma^+$ обозначим как $C(L)$ такое множество

$$C(L) = \{w \in L \mid w = uv \text{ и } u \in L \text{ влечет } v = \varepsilon\}.$$

Легко можно заметить, что $L \subset \Sigma^+$ является префиксным тогда и только тогда, когда $C(L) = L$.

В дополнение к словам из некоторого алфавита, будем также рассматривать бесконечные последовательности букв этого алфавита. Следовательно, будем использовать обозначение Σ^ω для множества однонаправленных последовательностей букв принадлежащих алфавиту Σ. Для $\pi \in \Sigma^\omega$ и $n \in \mathbb{N}$ будем использовать запись $\pi_{[1..n]}$ для выделения слова длины n которое совпадает с началом последовательности π. Также обозначим $\pi_{(n..)}$ как последовательность принадлежащую Σ^ω которая определяется равенством $\pi = \pi_{[1..n]}\pi_{(n..)}$.

2.2 Сравнительный анализ обработки простых и сложных событий

В этой части рассматриваются особенности обработки сложных событий в сравнении с обработкой простых (атомарных) событий.

Определение 2.1 (Атомарные и Сложные События). Будем говорить, что событие является атомарным, если его не возможно представить как совокупность других событий имеющих смысл в интересующей предметной области.

В противоположность атомарному событию, будем говорить, что событие является сложным, если оно может быть представлено как совокупность других событий имеющих смысл в интересующей предметной области.

Пример 2.1. В языке Java различаются низкоуровневые и семантические события. Клавиатура, мышь и другие устройства ввода генерируют низкоуровневые события.

В противоположность низкоуровневым событиям, семантические события генерируются как результат последовательности низкоуровневых событий, например *SelectItemEvent* является результатом последовательности: *GetFocusEvent*, *DownButtonEvent*, и *UpButtonEvent*.

Определение 2.2 (Обработка простых (атомарных) событий). Обработка простых событий основывается на утверждении о том, что каждое событие напрямую связано с определенными измеряемыми изменениями некого условия и может быть обработано независимо [62].

В случае обработки простых событий каждое произошедшее событие немедленно вызывает соответствующие действия по его обработке.

Обработка простых событий широко используется для управления потоками реального времени тем самым уменьшая временные задержки и стоимость [62].

Как показано на рис. 2.1 функция-обработчик обрабатывает атомарные события как только получены уведомления об их наступлении.

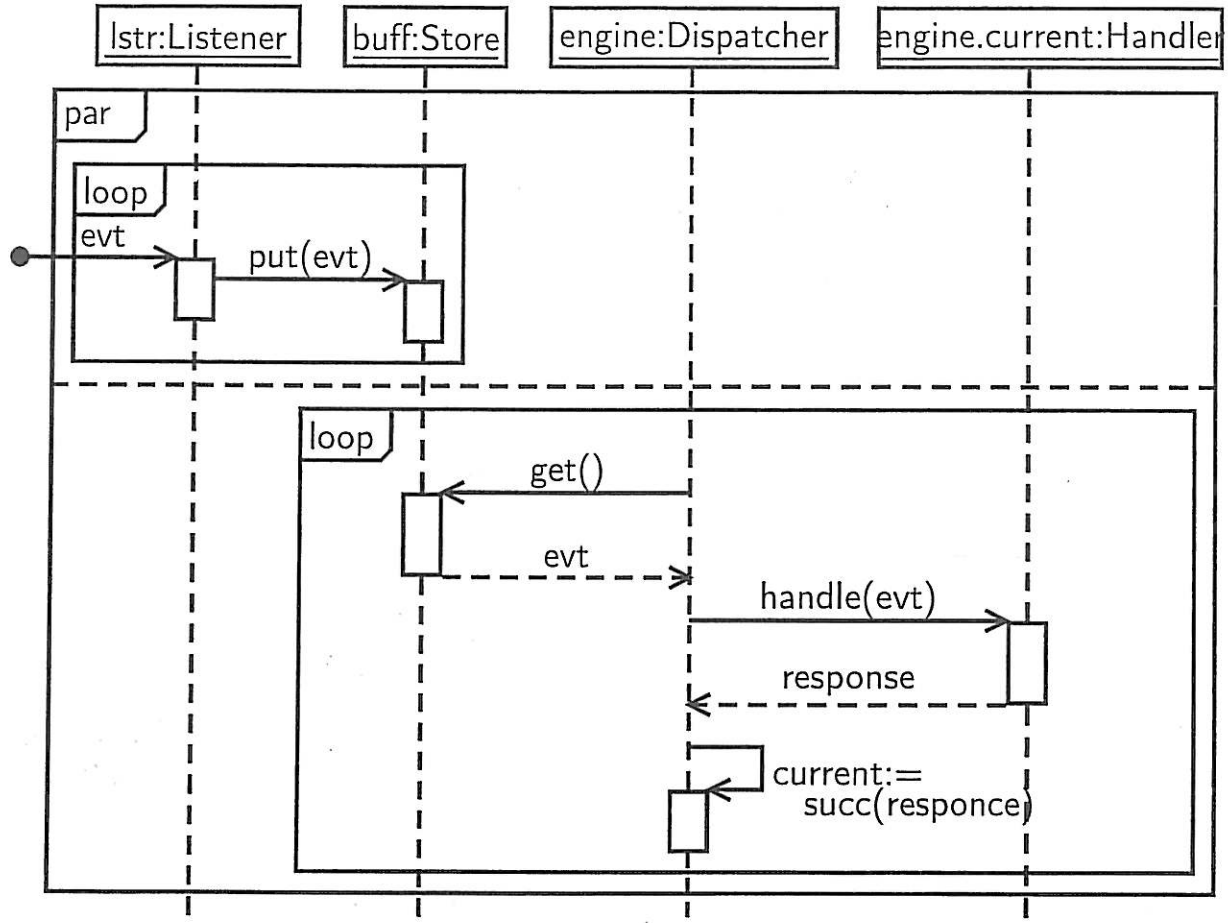

Рис. 2.1. Обработка простых событий

Определение 2.3 (Обработка сложных событий). Обработка сложных событий основывается на утверждении о том, что каждое атомарное событие содержит слишком мало информации для определения правильной реакции на него [62].

При обработке сложных событий оценивается сочетание событий, определяется их шаблон и затем выполняется соответствующее действие.

Сложные события могут иметь различные типы, а также могут происходить в течение длительного периода времени. Взаимозависимости между такими событиями могут также быть различными: причинными (causality), временными или пространственными. Обработка сложных событий требует использования сложных интерпретаторов событий, механизмов опре-

деления и распознавания шаблонов событий [62].

Как показано на рис. 2.2 обработчик сложных событий анализирует входной буфер атомарных событий для распознавания сложного события. Если обработчику удалось распознать сложное событие, то оно обрабатывается и удаляется из буфера. Следовательно, основным различием ме-

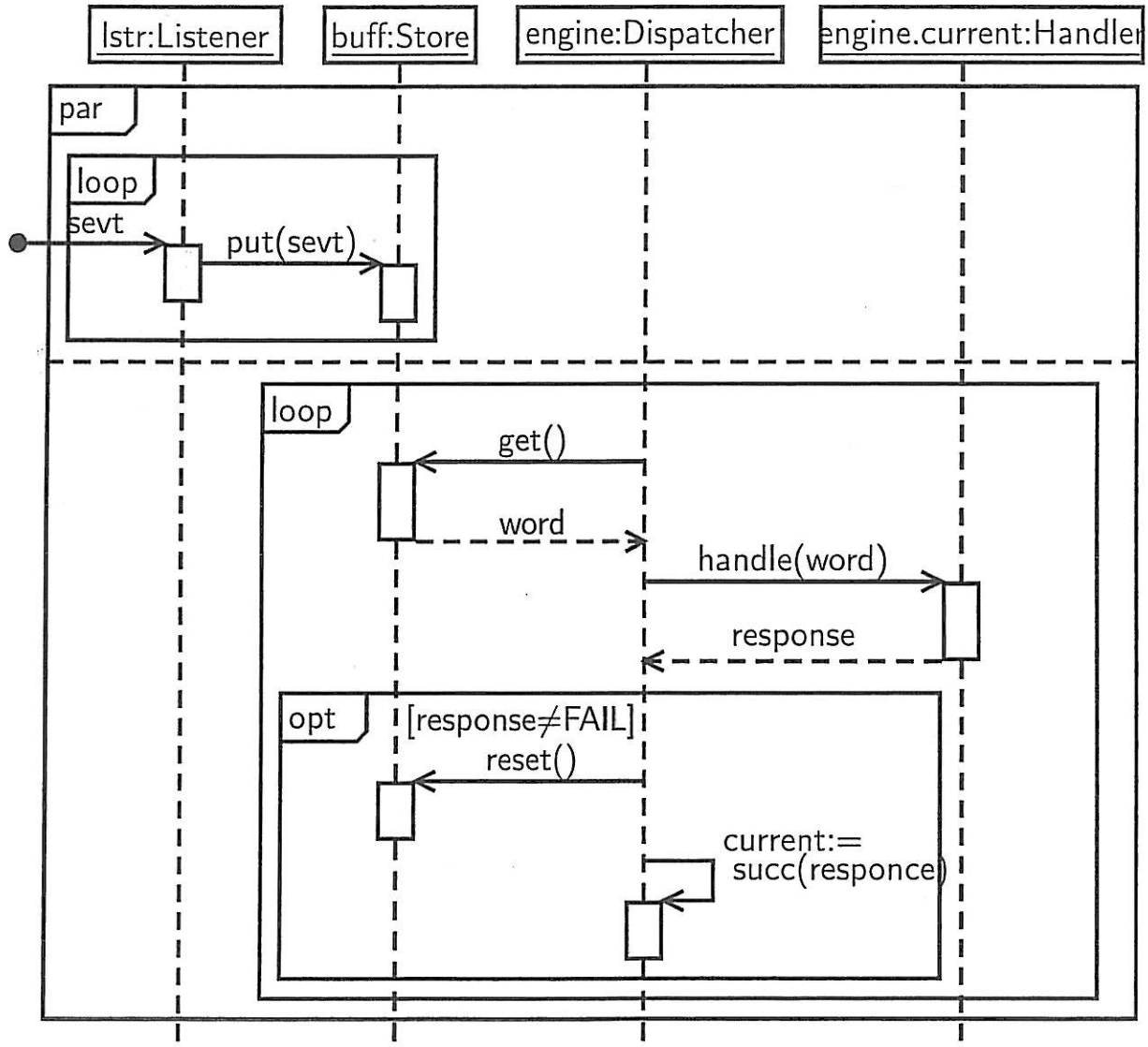

Рис. 2.2. Обработка сложных событий

жду обработкой простых и сложных событий является то, что в первом случае обрабатывающая система реагирует на произошедшее событие немедленно, в то время как во втором случае, обрабатывающая система анализирует буфер содержащий атомарные события, и пытается распознать

некоторые значимые сложные события и обработать их.

Факты приведенные выше и результаты полученные в работах [63, 64, 65] и [66] показывают, что предавтоматы могут широко использоваться для математического моделирования систем обработки сложных событий. Ниже в данном разделе будет доказан важный факт двойственности между предавтоматами и машинами обработки сложных событий [1].

2.3. Формальная модель системы обработки сложных событий

В этой части рассматривается математическая модель абстрактной машины обработки сложных событий. Для построения такой машины зафиксируем два конечных алфавита Σ и A для обозначения множеств атомарных событий и ответов системы соответственно.

Определение 2.4 (обработчик событий). Частично определенное отображение $h : \Sigma^+ \dashrightarrow A$ будем называть обработчиком событий (an Event Handler) если его область определения Dh является префиксным множеством.

Для пояснения необходимости выполнения условия $C(Dh) = Dh$, заметим, что процесс накопления последовательности атомарных событий, продолжающийся до тех пор пока последовательность не будет обработана, описывается этим условием.

Определение 2.5 (шаблон события). Пусть $h : \Sigma^+ \dashrightarrow A$ обработчик событий и $a \in A$ некоторый ответ системы, тогда префиксное множество $P_h(a) = \{w \in Dh \mid h(w) = a\}$ называется h,a-шаблоном (pattern).

Определение абстрактной машины обработки сложных событий должно пояснить неформальное описание, приведенное на рис. 2.2.

Определение 2.6 (CEP-машина). Абстрактная машина обработки сложных событий (ниже CEP-машина) это четверка $\mathscr{M} = (\Sigma, A, H, \delta)$ где Σ и A конечные алфавиты событий и ответов соответственно, H конечное множе-

ство обработчиков событий, и $\delta : A \to H$ всюду определенное отображение, которое называется отображением переходов.

Легко можно заметить, что каждый обработчик событий h в математической модели CEP-машины соответствует методу handle() класса *Handler* и отображение δ соответствует методу succ() класса Dispatcher (см. рис. 2.2).

Для определения поведения CEP-машины рассмотрим множество Σ^ω содержащее все бесконечные последовательности элементов принадлежащих Σ. Такие последовательности будем называть потоками событий.

Определим частичное отображение $T_h : \Sigma^\omega \dashrightarrow \mathbb{N}$ для любого обработчика $h \in H$ Задав такие условия

$$\begin{aligned} T_h(\pi) \downarrow = t \quad & \text{если } \pi_{[1..t]} \in Dh; \\ T_h(\pi) \uparrow \quad & \text{если } (\forall t \in \mathbb{N})\ h(\pi_{[1..t]}) \uparrow . \end{aligned} \tag{2.1}$$

Неформально, $T_h(\pi)$ первое время отклика обработчика h в процессе обработки потока событий π.

Определение 2.7 (Конфигурация). Пару $\langle h, \pi \rangle \in H \times \Sigma^\omega$ будем называть конфигурацией.

Определение 2.8 (Эволюционный оператор CEP-машины). Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина, тогда частичное отображение из множества конфигураций в себя $S : H \times \Sigma^\omega \dashrightarrow H \times \Sigma^\omega$ определяемое условиями

$$\begin{aligned} S\langle h, \pi \rangle \downarrow = \langle \delta(h(\pi_{[1..t]})), \pi_{(t..)} \rangle \quad & \text{если } T_h(\pi) \downarrow = t \\ S\langle h, \pi \rangle \uparrow \quad & \text{если } T_h(\pi) \uparrow \end{aligned}$$

будет называться эволюционным оператором (an evolutionary operator) CEP-машины.

Теперь для спецификации корректного поведения CEP-машины определим ее правильные сценарии работы, называемые потоками обработки.

Определение 2.9 (поток обработки). Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина и $\langle \langle h^{(t)}, \pi^{(t)} \rangle \mid\ t = 0, 1, \ldots \rangle$ последовательность конфигураций из $H \times \Sigma^\omega$ то-

гда она называется потоком обработки, если выполняется следующее равенство

$$\langle h^{(t+1)}, \pi^{(t+1)} \rangle = S\langle h^{(t)}, \pi^{(t)} \rangle \text{ для всех } t \geqslant 0.$$

2.4 Основные свойства CEP-машин

Простые, но очень важные свойства CEP-машин являются следствием существования некоторой естественной топологии на множестве Σ^ω.

Приведем некоторые теоретические сведения необходимые для дальнейшего изложения.

Определение 2.10 (Топология Тихонова). Пусть множество X такое что

$$X = \prod_{i \in I} X_i$$

это декартово произведение (как множеств) топологических пространств X_i, проиндексированых индексами $i \in I$. И $p_i : X \to X_i$ – проекция произведения на соответствующий сомножитель. Тихоновская топология на X – это наиболее грубая топология (то есть топология с наименьшим числом открытых множеств), для которой все проекции p_i непрерывны.

Открытые множества этой топологии – всевозможные объединения множеств вида $\prod_{i \in I} U_i$, где каждое U_i является открытым подмножеством X_i и $U_i \neq X_i$ только для конечного числа индексов.

В частности, открытые множества произведения конечного числа пространств – это просто объединение произведений открытых подмножеств исходных пространств. Произведение топологий на X – это топология порождаемая множествами вида $p_i^{-1}(U)$, где $i \in I$ и U является открытым подмножеством X_i. Другими словами, множества $\{p_i^{-1}(U)\}$ формируют предбазу топологии на X. Подмножество множества X является открытым тогда и только тогда, когда оно является (возможно бесконечным) объединением пересечений большого, но конечного числа множеств $p_i^{-1}(U)$ [67].

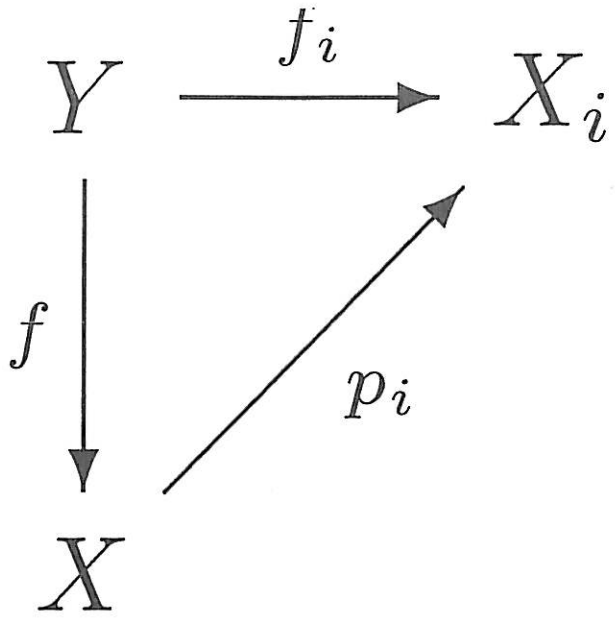

Рис. 2.3. Характеристическое свойство произведения пространств

Свойства тихоновских топологий. Топологическое пространство вместе с проекциями на каждую компоненту X_i может быть определено при помощи универсального свойства: если Y – произвольное топологическое пространство и для каждого $i \in I$ задано непрерывное отображение $Y \to X_i$, то существует единственное отображение $Y \to X$ такое что для каждого $i \in I$ диаграмма на рисунке 2.3 коммутативна [68].

Это показывает, что тихоновское произведение является произведением в категории топологических пространств. Из универсального свойства следует, что отображение $f: Y \to X$ непрерывно тогда и только тогда, когда непрерывно каждое отображение $f_i = p_i \circ f$, во многих ситуациях непрерывность f_i проверять проще. Проекции p_i являются не только непрерывными, но и открытыми отображениями (то есть каждое открытое множество произведения при проекции на компоненту переходит в открытое множество). Обратное, вообще говоря, неверно. Также проекции не обязательно являются замкнутыми отображениями. Топологию произведения иногда называют топологией поточечной сходимости. Причина этого сле-

дующая: последовательность элементов из произведения сходится тогда и только тогда, когда ее образ при проекции на каждую компоненту сходится.

Важной теоремой о произведении топологий является теорема Тихонова:

Теорема 2.1. Любое произведение компактных пространств является компактным.

Это легко показать для конечного числа произведений, в то время как общее утверждение эквивалентно аксиоме выбора [69, 70].

Теперь вернемся к изложению и рассмотрим для конечного алфавита Σ семейство подмножеств $\mathscr{B} = \left\{u \cdot \Sigma^{\omega} \mid u \in \Sigma^{+}\right\}$ на Σ^{ω}, которое содержит характеристические свойства базы топологии. Следовательно, будем рассматривать Σ^{ω} как топологическое пространство, а соответствующая топология является топологией Тихонова в пространстве последовательностей. Хорошо известно, что такое топологическое пространство является бикомпактным [71, 72].

Предложение 2.1. Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина, тогда для любого $h \in H$ отображение $T_h : \Sigma^{\omega} \dashrightarrow \mathbb{N}$, определенное в (2.1), является непрерывной функцией на множестве $\{\pi \in \Sigma^{\omega} \mid T_h(\pi)\downarrow\}$ в предположении, что топология на $\mathbb{N}$ является дискретной.

Доказательство. Легко можно заметить, что для любого $t \in \mathbb{N}$ равенство

$$T_h^{-1}(t) = \bigcup_{u \in Dh : |u| = t} u \cdot \Sigma^{\omega}$$

выполняется. Но, правая часть этого равенства является объединением множеств принадлежащих $\mathscr{B}$, следовательно, $T^{-1}(t)$ является открытым множеством. □

Следствие 2.1. Функция $T_h : \Sigma^{\omega} \dashrightarrow \mathbb{N}$ является кусочно-постоянной.

Следствие 2.2. Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина, S ее эволюционный оператор, а $D(S)$ область определения S, тогда

$$D(S) = \bigcup_{h \in H} D_h(S),$$

Где каждое множество $D_h(S) \subset \Sigma^\omega$ является открытым в топологии Тихонова на Σ^ω.

Теорема 2.2. Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина, S ее эволюционный оператор, а $h \in H$ некоторый обработчик событий, тогда $S\langle h, \pi\rangle \downarrow$ для всех $\pi \in \Sigma^\omega$ тогда и только тогда, когда Dh является конечным множеством и $\Sigma^\omega = \bigcup_{u \in Dh} u \cdot \Sigma^\omega$.

Доказательство. Действительно, предположим, что $S\langle h, \pi\rangle \downarrow$ для любого потока событий π, тогда утверждение $T_h(\pi) \downarrow$ выполняется для всех $\pi \in \Sigma^\omega$. Это значит что

$$\Sigma^\omega = \bigcup_{t=1}^{\infty} T^{-1}(t) = \bigcup_{t=1}^{\infty} \left(\bigcup_{u \in Dh : |u| = t} u \cdot \Sigma^\omega \right).$$

Далее, принимая во внимание предложение 2.1 и теорему Вейерштрасса об ограниченной возрастающей последовательности (Weierstrass Boundedness Theorem) можно заключить, что существует $M \in \mathbb{N}$ такое что $T_h(\pi) \leqslant M$ для всех $\pi \in \Sigma^\omega$. Следовательно, получаем

$$\Sigma^\omega = \bigcup_{t=1}^{M} \left(\bigcup_{u \in Dh : |u| = t} u \cdot \Sigma^\omega \right).$$

Таким образом, множество $D_0(h) = \{u \in Dh \mid 1 \leqslant |u| \leqslant M\}$ является конечным.

Пусть $D_0(h) = \{u_1, \ldots, u_m\}$ тогда

1. $u_i \in Dh$ для всех $i = 1, \ldots, m$ и
2. $\Sigma^\omega = \bigcup_{i=1}^{m} u_i \cdot \Sigma^\omega$.

Можно утверждать, что $u_i \cdot \Sigma^\omega \bigcap u_j \cdot \Sigma^\omega = \varnothing$ для $1 \leqslant i \neq j \leqslant m$. Действительно, предположим, что существует некоторый поток событий $\pi \in u_i \cdot \Sigma^\omega \bigcap u_j \cdot \Sigma^\omega$ тогда возможны три случая:

- $|u_i| = |u_j|$ и в этом случае имеем равенство $u_i = u_j$ которое противоречит предположению $i \neq j$;
- $|u_i| < |u_j|$ и u_i является префиксом u_j, что противоречит равенству $C(Dh) = Dh$;
- $|u_i| > |u_j|$ что также не возможно (причина та же что и в предыдущем случае).

Далее, пусть u произвольный элемент из Dh тогда принимая во внимание свойства 1 и 2 $D_0(h)$ приходим к утверждению: для каждого потока событий $\pi \in u \cdot \Sigma^\omega$ существует единственный $1 \leqslant i(\pi) \leqslant m$ такой что $\pi \in u_{i(\pi)} \cdot \Sigma^\omega$. Т.к. приведенное выше предположение $|u| \neq |u_{i(\pi)}|$ противоречит равенству $C(Dh) = Dh$ и равенство $|u| = |u_{i(\pi)}|$ означает что $u = u_{i(\pi)}$. Следовательно, $Dh = D_0(h)$ и прямое утверждение теоремы доказано. Обратное утверждение теоремы очевидно. □

Замечание 2.1. Теорема 2.2 не является сложной, но она обосновывает следующие важные утверждения: либо обработчик способен распознать бесконечное множество сложных событий, и в этом случае, обработка потоков событий невозможна, либо обработчик способен обработать любой поток событий, и в таком случае, возможно распознавание только конечного числа сложных событий.

2.5 CEP-машины и предавтоматы

Как было замечено выше, концепция CEP-машины тесно связана с концепцией предавтомата. Эта часть раздела посвящена описанию этой связи.

Определение 2.11 (см. [52]). Тройка (X, Σ, μ), где X некоторое множество, Σ конечный алфавит и $\mu : X \times \Sigma^* \dashrightarrow X$ частичное отображение, на-

зывается предавтоматом, если выполняются следующие условия:

1. равенство $\mu(x,\varepsilon)\downarrow = x$ выполняется для всех $x \in X$;
2. если утверждения $\mu(x,u)\downarrow$ и $\mu(\mu(x,u),v)\downarrow$ истинны для некоторых $x \in X$ и $u,v \in \Sigma^*$, тогда утверждение $\mu(x,uv)\downarrow = \mu(\mu(x,u),v)$ также является истинным;
3. если утверждения $\mu(x,u)\downarrow$ и $\mu(x,uv)\downarrow$ являются истинными для некоторых $x \in X$ и $u,v \in \Sigma^*$, тогда утверждение $\mu(\mu(x,u),v)\downarrow = \mu(x,uv)$ также истинно.

В этом контексте μ называется функцией переходов.

Теперь опишем способ ассоциации предавтомата и CEP-машины. Для CEP-машины $\mathscr{M} = (\Sigma, A, H, \delta)$ ассоциированный с ней предавтомат будет иметь вид тройки $\widehat{\mathscr{M}} = (H, \Sigma, \mu)$, такой, что частичное отображение $\mu : H \times \Sigma^* \dashrightarrow H$ удовлетворяет следующему условию

$$\begin{array}{l} \text{для любых } h \in H,\ w \in Dh, \text{ и } \pi \in \Sigma^\omega \\ \quad \text{конъюнкция } \mu(h,w)\downarrow \text{ и } S\langle h, w\cdot\pi\rangle \downarrow = \langle \mu(h,w), \pi\rangle \text{ истинна} \end{array} \tag{2.2}$$

где S эволюционный оператор CEP-машины $\mathscr{M}$.

Теорема 2.3. Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ CEP-машина и $\mu : H \times \Sigma^* \dashrightarrow H$ определяется следующим образом

1. $\mu(h,\varepsilon) = h$ для любого $h \in H$;
2. для любого $h \in H$, $u \in Dh$, и $v \in \Sigma^*$ истинность утверждения $\mu(\delta(h(u)),v)$ влечет истинность $\mu(h,uv)\downarrow$ и в таком случае $\mu(h,uv) = \mu(\delta(h(u)),v)$;
3. $\mu(h,w)\uparrow$ во всех остальных случаях,

тогда тройка $\widehat{\mathscr{M}} = (H, \Sigma, \mu)$ является предавтоматом и отображение μ удовлетворяет (2.2).

Для доказательства теоремы 2.3 необходима следующая лемма.

Лемма 2.1. Пусть μ определяется также как и в теореме 2.3, $h \in H$, а $w \in \Sigma^+$ некоторое слово такое что $\mu(h,w)\downarrow$, тогда существует единственная

последовательность переменных (alternating sequence) $h_0 \in H$, $u_0 \in \Sigma^+$, $h_1 \in H$, $u_1 \in \Sigma^+$, ..., $u_{n-1} \in \Sigma^+$, $h_n \in H$ такая, что

1. $w = u_0 \dots u_{n-1}$;
2. $u_i \in Dh_i$ для всех $i = 0, \dots, n-1$;
3. $h_0 = h$ и $h_{i+1} = \mu(h_i, u_i)$ для всех $i = 0, \dots, n-1$.

Доказательство. Для доказательства леммы рассмотрим следующий алгоритм.

1: Положим $h_0 \leftarrow h$, $w_0 \leftarrow w$, и $i \leftarrow 0$;
2: если $w_i \in Dh_i$, тогда положим $h_{i+1} \leftarrow \mu(h_i, w_i)$, $u_i \leftarrow w_i$, и останов;
3: выберем $u_i \in Dh_i$ и $w_{i+1} \in \Sigma^+$ т. о. чтобы выполнялось $w_i = u_i w_{i+1}$;
4: положим $h_{i+1} \leftarrow \mu(h_i, u_i)$ и $i \leftarrow i+1$;
5: переход к шагу 2.

Принимая во внимание, что $\mu(h, w)\downarrow$ и определение μ получаем для любого i либо $w_i \in Dh_i$ и алгоритм останавливается, либо w_i может быть представленно в соответствии с пунктом 3. В последнем случае определение μ гарантирует, что $\mu(h_i, w_i)\downarrow$. Более того, неравенство $|w_i| < |w_{i-1}|$ выполняется.

Следовательно, алгоритм останавливается т.к. каждый его шаг уменьшает длину слова w_i. Таким образом, после остановки алгоритма получаем требуемую последовательность. □

Следствие 2.3. Любое слово $w \in \Sigma^+$ такое что $\mu(h, w)\downarrow$ может единственным образом быть представленно в форме $w = uv$, где u слово удовлетворяющее условию $\mu(h, u)\downarrow$ и $v \in D(\mu(h, u))$.

Доказательство теоремы 2.3. Прежде всего, проверим что μ удовлетворяет условиям 1 – 3 определения 2.11. Верность условия 1 очевидна.
Для проверки условия 2 предположим, что $\mu(h, u)\downarrow$ и $\mu(\mu(h, u), v)\downarrow$. Используя лемму 2.1 можно построить последовательность переменных

$$h_0 = h, u_0, \dots, u_{m-1}, h_m = \mu(h, u), u_m, \dots, u_{m+n-1}, h_{m+n} = \mu(\mu(h, u), v)$$

так что последовательность $h_0 = h, u_0, \ldots, u_{m-1}, h_m = \mu(h,u)$ является последовательностью для $\mu(h,u)$ и $h_m = \mu(h,u), u_m, \ldots, u_{m+n-1}, h_{m+n} = \mu(\mu(h,u),v)$ является последовательностью для $\mu(\mu(h,u),v)$. Следовательно, $u = u_0 \ldots u_{m-1}$, $v = u_m \ldots u_{m+n-1}$, и $uv = u_0 \ldots u_{m-1} u_m \ldots u_{m+n-1}$. Очевидно, что условия 2 и 3 леммы 2.1 выполняются, следовательно, построенная последовательность является последовательностью для $\mu(h,uv)$. Это означает, что $\mu(h,uv) \downarrow =$ $\mu(\mu(h,u),v)$.

Проверка условия 3 аналогична проверке условия 2.

Верность (2.2) очевидна. □

Определение 2.12 (двойственный предавтомат для СЕР-машины). Пусть $\mathscr{M} = (\Sigma, A, H, \delta)$ СЕР-машина и $\widehat{\mathscr{M}} = (H, \Sigma, \mu)$ предавтомат определенный в теореме 2.3, тогда $\widehat{\mathscr{M}}$ называется двойственным предавтоматом для СЕР-машины $\mathscr{M}$.

Также возможно выполнить обратное построение и ассоциировать с любым предавтоматом $\mathscr{P}$ СЕР-машину $\mathscr{M}$ таким образом чтобы выполнялось равенство $\mathscr{P} = \widehat{\mathscr{M}}$.

Предложение 2.2. Предположим, что задан предавтомат $\mathscr{P} = (X, \Sigma, \mu)$ и определим четверку $\widehat{\mathscr{P}} = (\Sigma, X, H, \delta)$ так что

$H = \{h_x : \Sigma^+ \dashrightarrow X \mid x \in X\}$ где

$Dh_x = \{w \in \Sigma^+ \mid \mu(x,w) \downarrow$ и если $w = uv \wedge \mu(x,u) \downarrow$ тогда $v = \varepsilon\}$,

$h_x(w) = \mu(x,w)$ для $w \in Dh_x$;

$\delta(x) = h_x$.

Тогда $\widehat{\mathscr{P}}$ является СЕР-машиной и $\widehat{\widehat{\mathscr{P}}}$ является изоморфным к $\mathscr{P}$.

Доказательство. Пусть $\widehat{\mu} : H \times \Sigma^* \dashrightarrow H$ функция переходов для предавтомата $\widehat{\widehat{\mathscr{P}}}$. Тогда для $w \in Dh_x$ имеем $\widehat{\mu}(h_x, w) = \delta(h_x(w)) = h_{\mu(x,w)}$. Продолжив отображение $\widehat{\mu}(x,w)$ для $w \in \Sigma^*$ согласно условиям теоремы 2.3 получаем $\widehat{\mu}(h_x, w) \downarrow$ тогда и только тогда, когда $\mu(x,w) \downarrow$ и в этом случае

$\widehat{\mu}(h_x, w) = h_{\mu(x,w)}$. Следовательно, отображение такое что $x \mapsto h_x$ является изоморфизмом предавтоматов [52] □

2.6 Синтез CEP-машин с заданным поведением

Реальная инженерная практика требует улучшения методов разработки, в частности, для синтеза систем обработки сложных событий. Ключевым понятием для такой задачи синтеза является понятие "поведение CEP-машины". В этой части раздела предлагается обобщение для понятия "поведение", которое было определено для автоматов, и решает соответствующую задачу синтеза.

В этом контексте, первой задачей является задание способа описания поведения CEP-машины. Можно утверждать, что описание поведения CEP-машины и задание отображения, ассоциирующего входной поток событий с последовательностью ответов CEP-машины, являются решениями одной и той же задачи. Эти рассуждения приводят к следующим определениям.

Определение 2.13 (инициальная CEP-машина). Пятерка $\mathscr{M} = (\Sigma, A, H, h_*, \delta)$ где $h_* \in H$, называется инициальной CEP-машиной, если четверка (Σ, A, H, δ) является CEP-машиной.

Другими словами, инициальная CEP-машина это CEP-машина с отмеченным обработчиком событий – инициальным обработчиком. Этот обработчик устанавливается активным при инициализации CEP-машины.

Определение 2.14 (функция отклика CEP-машины). Для произвольной инициальной CEP-машины $\mathscr{M} = (\Sigma, A, H, h_*, \delta)$ определим частичное отображение $\beta : \Sigma^+ \dashrightarrow A$ следующим образом

1. область определения β задается равенством $D\beta = \{w \in \Sigma^+ \mid \mu(h_*, w)\downarrow\}$ где μ функция переходов предавтомата $\widehat{\mathscr{M}}$;
2. для $w \in D\beta$ используя следствие 2.3 представим $w = uv$ так что

$\mu(h_*, u)\downarrow$ и $v \in D(\mu(h_*, u))$ тогда определим $\beta(w) = \mu(h_*, u)(v)$.

Построенное отображение β называется функцией отклика СЕР-машины $\mathscr{M}$.

В следующем предложении доказывается основное свойство функции отклика СЕР-машины.

Предложение 2.3. Пусть $\mathscr{M} = (\Sigma, A, H, h_*, \delta)$ инициальная СЕР-машина и $\beta : \Sigma^+ \dashrightarrow A$ ее функция отклика, тогда выполняется следующее условие, для любых $u, v \in D\beta$ таких что $\beta(u) = \beta(v)$ и любого $w \in \Sigma^*$

$$uw \in D\beta \text{ влечет } vw \in D\beta \text{ и } \beta(uw) = \beta(vw). \tag{2.3}$$

Доказательство. Пусть $u, v \in D\beta$ и $\beta(u) = \beta(v)$. принимая во внимание последнее равенство и представление $\mu(h_*, u) = \delta(\beta(u))$ можно заключить, что $\mu(h_*, u) = \mu(h_*, v)$.

Если $uw \in D\beta$ тогда $\mu(h_*, uw)\downarrow$ и используя условие 3 определения 2.11 получаем что $\mu(\mu(h_*, u), w)\downarrow$ и, следовательно,

$$\mu(h_*, uw) = \mu(\mu(h_*, u), w) = \mu(\delta(\beta(u)), w) = \mu(\delta(\beta(v)), w) = \mu(\mu(h_*, v), w).$$

Следовательно, $\mu(\mu(h_*, v), w)\downarrow$ и используя условие 2 определения 2.11 получаем, что $\mu(h_*, vw)\downarrow$ то есть $vw \in D\beta$.

Далее, $\beta(u) = \beta(v)$ влечет $\mu(h_*, u) = \mu(h_*, v)$ и можно использовать декомпозицию из следствия 2.3 для $h = \mu(h_*, u) = \mu(h_*, v)$ и w. Пусть $w = w'w''$ соответствующая декомпозиция, тогда $\mu(h, w')\downarrow$ и $w'' \in D(\mu(h, w'))$. Определение 2.11 и равенство $\beta(u) = \beta(v)$ гарантируют, что утверждения $\mu(h_*, uw')\downarrow$ и $\mu(h_*, vw')\downarrow$ одновременно выполняются и равенство $\mu(h_*, uw') = \mu(h_*, vw')$ также выполняется. Следовательно, имеем

$$\beta(uw) = \mu(h_*, uw')(w'') = \mu(\mu(h_*, u), w')(w'') = \mu(h, w')(w'') = \\ \mu(\mu(h_*, v), w')(w'') = \mu(h_*, vw')(w'') = \beta(vw).$$

Таким образом, доказательство завершено. □

Обратное утверждение для предыдущего предложения также верно.

Теорема 2.4 (о синтезе СЕР-машины). Пусть Σ и A конечные алфавиты, а $\beta : \Sigma^+ \dashrightarrow A$ частичное отображение удовлетворяющее (2.3), тогда существует инициальная СЕР-машина $\mathscr{M} = (\Sigma, A, H, h_* \in H, \delta)$ чья функция отклика совпадает с β.

Доказательство. Проведем доказательство в два этапа. Во-первых, построим СЕР-машину $\mathscr{M}_\beta$ используя отображение β, а затем докажем, что ее функция отклика совпадает с β.

Для реализации первого этапа доказательства, рассмотрим на множестве $D\beta \subset \Sigma^*$ бинарное отношение $\equiv_\beta$ определенное следующим образом: $u \equiv_\beta v$ означает что $\beta(u) = \beta(v)$ выполняется. Очевидно, что это отношение является отношением эквивалентности.

Теперь рассмотрим $H = \{h_*\} \bigcup \{h_{[u]_\beta} \mid u \in D\beta\}$ где $[u]_\beta$ класс эквивалентности u по отношению к $\equiv_\beta$. Принимая во внимание, что $\beta|[u]_\beta$ является постоянным отображением для каждого $u \in D\beta$ можно заключить, что $|H| \leqslant |A| + 1$. Следовательно, H является конечным множеством.

Определим $h_*(w)\downarrow$ если $w \in C(D\beta)$ и в таком случае $h_*(w) = \beta(w)$. Определение $C(D\beta)$ гарантирует, что h_* является обработчиком событий.

Далее, для $u \in D\beta$ определим, что $h_{[u]_\beta}(w)\downarrow$ если $uw \in C(D\beta)$. Условие (2.3) гарантирует, что такое определение не зависит от выбора $u' \in [u]_\beta$. Более того, в рассматриваемом случае равенство $\beta(uw) = \beta(u'w)$ выполняется. Следовательно, выражение $h_{[u]_\beta}(w) = \beta(uw)$ определяет $h_{[u]_\beta}$ корректно. Чтобы проверить тот факт, что $h_{[u]_\beta}$ является обработчиком событий, предположим что $w'w'' \in Dh_{[u]_\beta}$ и $w' \in Dh_{[u]_\beta}$ тогда $uw'w'' \in C(D\beta)$ и $uw' \in C(D\beta)$ одновременно выполняются. Но, последняя конъюнкция означает что $w'' = \varepsilon$ по определению $C(D\beta)$. Поэтому, $h_{[u]_\beta}$ является обработчиком событий.

Чтобы определить $\delta : A \to H$ заметим, что для любого $a \in \beta(D\beta)$ равенство $\beta(u) = a$ единственным образом определяет $[u]_\beta$, следовательно, мо-

жно определить δ на $\beta(D\beta)$ следующим образом $\delta(a) = [u]_\beta$ где $\beta(u) = a$.
На множестве $A \setminus \beta(D\beta)$ можно определить отображение δ произвольно, например, задав $\delta|(A \setminus \beta(D\beta)) = h_*$.
Таким образом, получаем CEP-машину $\mathscr{M}_\beta = (\Sigma, A, H, \delta)$.

Для завершения доказательства необходимо показать, что функция отклика полученной CEP-машины $\mathscr{M}_\beta$ совпадает с β. Обозначим как $\widehat{\beta}$ функцию отклика CEP-машины $\mathscr{M}_\beta$ и также обозначим как μ функцию переходов предавтомата $\widehat{\mathscr{M}_\beta}$.
По определению $D(\widehat{\beta})$ совпадает со множеством $\{w \in \Sigma^+ \mid \mu(h_*, w)\downarrow\}$. Для такого слова w применим лемму 2.1 и построим последовательность $h_0 = h_*$, u_0, $h_1 = h_{[u_0]_\beta}$, $\ldots$, $h_{n-1} = h_{[u_0 \ldots u_{n-2}]_\beta}$, u_{n-1}, $h_n = h_{[u_{n-1}]_\beta}$ такую что $u_0 \ldots u_i \in Dh_i$ для $i = 0, \ldots, n-1$ и $w = u_0 \ldots u_{n-1}$.
Тогда, $\widehat{\beta}(w) = \beta(u_0 \ldots u_{n-2} u_{n-1}) = \beta(w)$ □

2.7 Обработка сложных событий и вычислимость

Выше рассматривались общие свойства CEP-машин и рассуждения строились подобным образом, как это делается в теории конечных автоматов [73]. Однако, задачи ассоциированые с CEP-машинами не могут быть решены теми же способами, что и задачи в теории конечных автоматов. Таким образом, задача вычислимости для CEP-машин требует дополнительного изучения.

Чтобы использовать описанный выше подход для решения задач проектирования программных систем, ограничим класс обработчиков событий, а именно, будем рассматривать только вычислимые обработчики событий. Такое ограничение является ключевым для возможности обработки потоков событий используя вычислительную систему. В самом общем виде схема обработки сложных событий представлена на рис. 2.2. Следует подчеркнуть, что в этой схеме есть потенциальная проблема. Она связана с

применением метода handle(). Этот метод последовательно применяет соответствующий обработчик событий к текущему буферу событий. Но, в общем случае, вычислимая функция не определяет данные, которые не принадлежат ее области определения. Следовательно, попытка вызова этого метода может привести к его бесконечному выполнению. В этой части будет показано, что обнаруженная аномалия может быть устранена. Приведем необходимые теоретические сведения. Напомним, что предикат $M(x)$ называется разрешимым, если его характеристическая функция, задаваемая формулой

$$c_M(x) = \begin{cases} 1, \text{ если } M(x) \text{ истинно,} \\ 0, \text{ если } M(x) \text{ ложно} \end{cases}$$

вычислима. Предикат $M(x)$ называется неразрешимым, если он не является разрешимым. Алгоритм, вычисляющий c_M, называется разрешающей процедурой для $M(x)$.

Теорема 2.5. Предикат $M(x)$ частично разрешим тогда и только тогда, когда существует вычислимая функция $g(x)$, такая, что $M(x)$ тогда и только тогда, когда $x \in Dom(g)$ [74].

Следующее свойство частично разрешимых предикатов показывает их связь с разрешимыми предикатами.

Теорема 2.6. Предикат $M(x)$ частично разрешим тогда и только тогда, когда существует разрешимый предикат $R(x,y)$, такой, что $M(x)$ тогда и только тогда, когда $\exists\, y\ R(x,y)$ [74].

Теорема 2.6 подсказывает важный способ интерпретации частично разрешимых предикатов. Она говорит о том, что частично разрешимые процедуры всегда могут быть представлены в виде процедуры неограниченного поиска числа y, обладающего некоторым разрешимым свойством $R(x,y)$. Такой поиск проще всего проводить, полагая последовательно $y = 0, 1, 2, \ldots$. Если и когда найдено такое y, для которого свойство $R(x,y)$ выполнено, то

происходит остановка. В противном случае поиск продолжается бесконечно долго [74].

Перейдем теперь к формулировке условия алгоритмической реализуемости CEP-машины.

Теорема 2.7 (о разрешимости задачи останова для обработчиков событий). Пусть Σ и A конечные алфавиты, $h : \Sigma^+ \dashrightarrow A$ вычислимый обработчик событий, а $\pi \in \Sigma^\omega$ поток событий тогда, предикат $M(\pi) \cong \exists n\, h(\pi_{[1..n]})\downarrow$ является разрешимым относительно $g(n) = \pi_{[1..n]}$.

Доказательство. В теории вычислимости доказано, что область определения вычислимой функции является полуразрешимым множеством. Следовательно, предикат $w \in D(h)$ является полуразрешимым. Такой предикат представляется как $\exists t\ R(w,t)$ для некоторого разрешимого предиката R (см. [74, стр. 114, Теорема 6.4]). В качестве R предиката можно выбрать, например, предикат "программа вычисляющая h выполняет не более чем t команд и останавливается когда на входе w".

Пусть $\gamma : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ отображение определяемое выражениями

$$\begin{aligned} \gamma_1(n) &= \mu k \left(n < \frac{k(k+1)}{2} \right) - 1 \\ \gamma_2(n) &= n - \gamma_1(n) \end{aligned},$$

где $\mu k\ (M(k,n))$ означает наименьшее значение k удовлетворяющее условию $M(k,n)$. Очевидно, что $\gamma = (\gamma_1, \gamma_2)$ является вычислимой биекцией. Теперь рассмотрим следующий алгоритм

1: положим $n \leftarrow 1$;

2: положим $m,t \leftarrow \gamma(n)$;

3: если $R(g(m),t)$ тогда вернуть $g(m)$ и останов;

4: положим $n \leftarrow n+1$;

5: перейти к пункту 2.

Принимая во внимание, что область определения h является префиксным множеством, можно легко проверить следующее утверждение: либо суще-

ствует m такое что $\pi_{[1..m]} \in D(h)$ и алгоритм обнаруживает это, либо алгоритм выполняется бесконечно. Для завершения доказательства достаточно преобразовать этот неформальный алгоритм в соответствующую машину Тьюринга. □

Следствие 2.4. Если все обработчики событий CEP-машины являются вычислимыми, то такая машина алгоритмически реализуема.

Доказательство. Действительно, можно найти $\pi_{[1..m]} \in D(h)$ используя алгоритм из теоремы 2.7, если такой поток $\pi_{[1..m]}$ существует,тогда можно вычислить $h(\pi_{[1..m]})$ □

Выводы по разделу 2

В этом разделе было проведено построение математической модели систем обработки сложных событий. Этот процесс основывался на известных примерах программных решений для таких систем и их обобщении. Таким образом, полученные результаты основываются на ранее проведенном изучении систем обработки сложных событий.

Обобщение инвариантов для таких примеров привело к формулировке математической модели, которая была названа CEP-машиной. Модель была описана с двух точек зрения: структурной и поведенческой. Для описания поведенческой части модели, в разделе было введено множество формальных понятий.

Основные свойства CEP-машин были изучены основываясь на этих формальных понятиях. Теорема 2.2 поясняет принципиальный результат этого изучения. Он заключается в том, что либо обработчик способен распознать бесконечное количество сложных событий и, в таком случае, обработка потоков событий невозможна, либо обработчик способен обработать любой поток событий и, в таком случае, возможно обнаружение только конечного числа сложных событий.

Двойственность между CEP-машинами и предавтоматами (см. теорему 2.3 и предложение 2.2) является еще одним важным результатом. Этот результат представляет способы изучения поведения CEP-машин.

Важной задачей для приложений является задача синтеза CEP-машины с заданным поведением, которое задается функцией отклика. Теорема 2.4 описывает и дает основания для решения этой задачи.

К сожалению, теорема 2.4 не устанавливает никаких универсальных свойств для построенной CEP-машины, в противоположность аналогичной конструкции для конечных автоматов. Таким образом, вопрос универсальности для такого построения CEP-машины остается открытым.

Далее, теорема 2.7 дает решение задачи об алгоритмической реализуемости CEP-машины. Эта теорема доказывает, что условие вычислимости обработчиков событий CEP-машины является достаточным для ее алгоритмической реализуемости.

Полученные в разделе результаты открывают направления для дальнейшего исследования. Следующее утверждение "Если функция отклика является вычислимой, то соответствующая CEP-машина алгоритмически реализуема" является примером открытой задачи.

Следует также отметить другое направление анализа CEP-машин. Необходимость этого направления следует из свойств CEP- машин отмеченных выше: если CEP-машина не является тривиальной, тогда существуют потоки событий, которые могут быть обработаны этой машиной. Следовательно, изучение таких аномалий является важной задачей. Такое изучение могло бы проводиться в вероятносном контексте и предоставляло бы значимые оценки для аномалий поведения CEP-машин.

РАЗДЕЛ 3
СИСТЕМЫ ОБРАБОТКИ РЕГУЛЯРНЫХ СОБЫТИЙ И МАШИННОЕ ОБУЧЕНИЕ

Этот раздел посвящен специальному классу сложных событий, которые могут быть описаны регулярным языком. Для обработки таких событий определяется специальный класс регулярных CEP-машин. Изучаются их свойства. А также показывается, что такие машины могут быть преобразованы в конечный детерминированый автомат. Используя регулярные машины обработки сложных событий, предлагается метод машинного обучения для компонентов системы основанной на архитектуре управляемой событиями. Также детально описывается процедура верификации правильности этого метода.

3.1 Базовые обозначения и определения

В этой части раздела мы зафиксируем базовые обозначения и приведем необходимые определения. Далее, будем использовать следующие обозначения:

$f\colon X \dashrightarrow Y$	обозначает, что f является частичным отображением из X в Y;
$f(x)\uparrow$	обозначает, что отображение $f(x)$ не определено для элемента x из множества X;
$f(x)\downarrow$	обозначает, что отображение $f(x)$ определено для элемента x из множества X;

$f(x)\downarrow = y$	обозначает, что $f(x)\downarrow$ и $y = f(x)$ для элемента y из множества Y;
ε	обозначает пустую (нулевой длины) последовательность;
X^+	обозначает множество всех не пустых конечных последовательностей состоящих из элементов множества X;
X^*	обозначает множество $\{\varepsilon\} \bigcup X^+$;
X^ω	обозначает множество всех бесконечных последовательностей состоящих из элементов множества X;
X^∞	обозначает множество $X^* \bigcup X^\omega$;
$\|x\|$	обозначает длину конечной последовательности x;
$x[0]$	обозначает первый элемент конечной или бесконечной последовательности x;
$x[1:]$	обозначает последовательность получаемую путем удаления первого элемента последовательности x.

Теперь приведем определения необходимые для описания регулярной CEP-машины:

Определение 3.1. Определим конечное множество X называемое алфавитом, а его элементы будем называть символами. В этом контексте будем называть множество $L \subseteq X^*$ языком порожденным X.

Последовательности символов представляют собой потоки элементарных сообщений информирующие о произошедших элементарных событиях. Некоторые конечные последовательности символов содержат в себе информацию о сложных событиях. Ниже будем называть такие последова-

тельности символов событиями. В противоположность, другие конечные последовательности символов не содержат в себе сложных событий. Такие последовательности будем называть словами.

Множества сложных событий должны удовлетворять некоторым условиям. Одним из наиболее важных является условие которое гласит что любой поток элементарных событий единственным способом разделяется на серии сложных событий при последовательном просмотре потока слева направо. Это условие приводит к следующему определению.

Определение 3.2. Пусть L некоторый язык с алфавитом X тогда L является префиксным множеством если для любого $u \in L$ и $v \in X^*$ предположение $uv \in L$ влечет равенство $v = \varepsilon$.

Зафиксируем что далее, всякое множество сложных событий является префиксным. Теперь перейдем к описанию обработчиков потоков таких событий.

3.2 Регулярные обработчики событий

В этой части раздела рассматривается и изучается некоторый класс обработчиков сложных событий тесно связанных с конечными автоматами [2]. Такие обработчики сложных событий будем называть регулярными. Для описания регулярных обработчиков в начале приведем необходимые теоретические понятия и определения.

Определение 3.3 (Конечный автомат). Детерминированный конечный автомат это пятерка $(X, Z, z_0, \delta, Z_{acc})$ где:

- X – входной алфавит. Т.е. конечное непустое множество входных символов.
- Z – конечное не пустое множество состояний автомата.
- $z_0 \in Z$ – начальное состояние.
- δ – функция переходов: $\delta\colon Z \times X \to Z$ В случае недетермини-

рованого конечного автомата функция переходов будет иметь вид: $\delta\colon\ Z\times\ X\to\ \mathscr{P}(Z)$, т.е., δ возвращает множество состояний.

— $Z_{acc}\ \subset\ Z$ – (возможно пустое) множество допускающих состояний [75, 76, 77].

В дальнейшем будем использовать только детерминированые конечные автоматы (ДКА).

Определение 3.4 (Регулярное множество). Множество слов (язык) $L\subseteq X^*$ допускаемое конечным автоматом является регулярным. Т.е. множество слов L регулярно если существует допускающий его конечный автомат.

Заметим, что задача определения языка допускаемого заданным конечным автоматом является примером алгебраической задачи нахождения пути в графе. Которая в свою очередь является обобщением задачи нахождения кратчайшего пути в графе, ребра которого взвешены элементами произвольного полукольца. [78, 79, 80]

Теперь рассмотрим специальный класс обработчиков сложных событий.

Определение 3.5 (регулярный обработчик). Обработчик $h\colon X^+ \dashrightarrow Y$ называется регулярным если существуют

— некоторое конечное множество Z с отмеченным элементом $z_0\in Z$ и

— некоторое отображение $\delta\colon Z\times X\to Z\bigcup Y$

такие что для любого $u\in X^+$ и $y\in Y$ условие $h(u)\downarrow=y$ выполняется тогда и только тогда, когда существуют $z_1,\ldots,z_{|u|-1}\in Z$ такие что

$$\begin{aligned} z_{i+1} &= \delta(z_i,u[i]) \quad \text{для} \quad 0\leqslant i<|u|-1 \quad \text{и} \\ y &= \delta(z_{|u|-1},u[|u|-1]). \end{aligned}$$

Из определения следует что область определения регулярного обработчика Dh является регулярным множеством.

Пример 3.1. Рассмотрим конечный автомат $(X,Z,\delta\colon Z\times X\to Z)$ и зафиксируем некоторое начальное состояние $z_0\in Z$, а также множество допуска-

ющих состояний $Z_{acc} \subset Z$. Наш выбор должен удовлетворять следующим условиям:

1. $z_0 \notin Z_{acc}$;
2. $\left(\bigcup_{a \in X} \delta(Z_{acc}, a) \right) \bigcap Z_{acc} = \varnothing$;
3. любое состояние из множества Z_{acc} недостижимо из $\bigcup_{a \in X} \delta(Z_{acc}, a)$.

Далее, пусть отображение $\beta \colon Z_{acc} \to Y$ и определим $h(u) = y \in Y$ если $\delta^*(z_0, u) \in Z_{acc}$ и $\beta(\delta^*(z_0, u)) = y$.

Мы утверждаем что h является обработчиком.

Доказательство. Действительно, рассмотрим множество слов $L = \{w | \delta^*(z_0, w) \in Z_{acc}\}$. Легко видить, что множество L является префиксным. В самом деле, пусть некоторые слова $u, w = uv \in L$, тогда слово w может быть принято автоматом только если $v = \varepsilon$. Следовательно, $\forall\, w \in L$, $\beta(\delta^*(z_0, w)) \in Y$ и $Dh \subseteq L$. Таким образом, Dh является префиксным множеством , и следовательно, h – обработчик (см. определение 2.4). □

3.3 Свойства регулярных обработчиков

В данной части раздела рассмотрим основные свойства регулярных обработчиков.

Определение 3.6 (Реализация обработчика). Пусть $h \colon X^+ \dashrightarrow Y$ – обработчик такой что существует конечный автомат $(X, Z, \delta \colon Z \times X \to X)$, z_0, Z_{acc}, β как в примере 3.1, и удовлетворяется уравнение $h(\cdot) = \beta(\delta^*(z_0, \cdot))$ тогда $((X, Z, \delta), z_0, Z_{acc}, \beta)$ называется реализацией обработчика h.

Принципиальный результат рассуждений приведенных выше таков.

Теорема 3.1. Обработчик $h \colon X^+ \dashrightarrow Y$ является регулярным тогда и только тогда когда существует его реализация.

Доказательство. Для начала покажем что если обработчик является регулярным, тогда существует его реализация. Действительно, если неко-

торый обработчик $h \in H$ является регулярным, тогда его область определения Dh это регулярное множество. Следовательно, существует автомат $(X, Z, \delta\colon Z \times X \to Z)$ такой что $\forall w \in Dh$, $\delta^*(z_0, w) \in Z_{acc}$ и можно определить отображение $\beta\colon Z_{acc} \to Y$ такое что $\beta(\delta^*(z_0, w)) = h(w) = y \in Y$. Таким образом, существует реализация регулярного обработчика h такая что $((X, Z, \delta), z_0, Z_{acc}, \beta)$.

Далее, покажем что если для некоторого обработчика h существует реализация $((X, Z, \delta), z_0, Z_{acc}, \beta)$ тогда такой обработчик h является регулярным. В самом деле, если $\forall w \in Dh$ уравнение $\beta(\delta^*(z_0, w)) = h(w)$ выполняется, то Dh является регулярным множеством и следовательно h – регулярный обработчик. □

3.4 Регулярные машины обработки сложных событий

В этой части раздела рассматривается некоторый класс машин обработки сложных событий, у которых все обработчики являются регулярными. Для описания таких машин приведем необходимые определения.

Определение 3.7 (Регулярная машина обработки сложных событий). Любая регулярная CEP-машина это пятерка $M = (X, Y, H, h_0, \alpha)$ состоящая из таких компонентов:

— X – конечное множество (алфавит) атомарных событий;

— Y – конечное множество (алфавит) ответов машины соответственно;

— H – конечное множество регулярных обработчиков;

— $h_0 \in H$ – начальный обработчик;

— $\alpha\colon Y \to H$ – функция отклика.

Алгоритм 3.1 описывает поведение любой регулярной CEP-машины [5]. Заметим, что в общем случае шаг 5 алгоритма может выполняться бесконечно см. раздел 2, часть 2.7. Но в случае регулярных CEP-машин алгоритм гарантировано не зависнет на этом шаге.

Алгоритм 3.1. Operational model of a Regular CEP-machine

```
1  def run(M, s):
       Require: the studied Regular CEP-machine M = (X, Y, H, h_0, α) and
                some stream of elementary events s ∈ X^ω
       Ensure : printing of the corresponding response stream
2      handler, buff = h_0, [ ]
3      while True:
4          new_event, s = s[0], s[1 :]
5          buff.append(new_event)
6          if handler(buff) ↑: continue
7          else:
8              response = handler(buff)
9              print(response)   # printing of the current response
10             handler, buff = α(response), [ ]
```

3.5 Основные свойства регулярных машин

Для изучения свойств регулярных CEP-машин напомним некоторые понятия связанные с концепцией частичного действия на моноидах [52].

Определение 3.8. [81] Пусть M – моноид с единицей e, Z – некоторое множество. Частичное действие M на Z является частичным отображением $f\colon Z \times M \dashrightarrow Z$ если: $f(z,a) = z_a$ такое что $\forall\, z \in Z\ f(z,e) = z$,
$\forall\, a,b \in M\ \forall\, z \in Z\ f(z,a) \neq \varnothing$ и $f(f(z,a),b) \neq \varnothing$ следует $f(z,ab) \neq \varnothing$ и $f(f(z,a),b) = f(z,ab)$
$\forall\, a,b \in M\ \forall\, z \in Z\ f(z,a) \neq \varnothing$ и $f(z,ab) \neq \varnothing$ следует $f(f(z,a),b) \neq \varnothing$ и $f(f(z,a),b) = f(z,ab)$.

Частичные действия иллюстрируются следующей ситуацией. Предположим что полное действие моноида M заданно на множестве Y и $Z \subset Y$. Тогда ограничение действия на множество Z является частичным действием.

Обратно, пусть $Z \times M \dashrightarrow Z$ –частичное действие и $Y \supset Z$.

Определение 3.9 (Глобализация). Полное действие $Y \times M \to Y$ называется глобализацией если его ограничение на Z совпадает с заданным частичным действием.

Следующая конструкция дает глобализацию для любого частичного действия $Z \times M \dashrightarrow Z$. Определим на множестве $Z \times M$ отношение $\vdash$: $f(z,ab) \vdash f(f(z,a),b) \Longleftrightarrow f(z,a) \neq \varnothing$. Пусть $\backsimeq$ является эквиваленцией порожденной $\vdash$ и $Y = (Z \times M) \setminus \backsimeq$. Обозначим как $[z,a]$ $\backsimeq$-класс содержащий пару (z,a). Получаем что множество $[z,a]b = [z,ab]$ для $[z,a] \in Y$, $b \in M$. Конструкция приведенная выше определяет полное действие на Y.

Теорема 3.2. [81] Определенное выше действие $Y \times M \to Y$ является глобализацией $Z \times M \dashrightarrow Z$; отображение $\alpha : Z \to Y : z \mapsto [z,e]$ – инъективный морфизм, и любой морфизм частичного действия M на Z в полном действии M может быть профакторизован по α.

Таким образом для некоторого предавтомата (Z, X, μ) можно построить глобализацию пополнив множество его состояний Z новыми состояниями. Теперь используя концепцию глобализации докажем следующую теорему.

Теорема 3.3 (Преобразование регулярной машины обработки сложных событий). Регулярную машину обработки сложных событий $\mathscr{M} = (X, Y, H, h_0, \alpha)$ можно преобразовать в конечный автомат $(X, Z, \delta : Z \times X \to Z)$, который является глобализацией ее двойственного предавтомата $\widehat{\mathscr{M}} = (Z_H, X, \mu : Z_H \times X^* \dashrightarrow Z_H)$.

Доказательство. Для доказательства того что регулярная машина обработки сложных событий $\mathscr{M}$ может быть преобразована в конечный автомат, для начала построим двойственный ей предавтомат $\widehat{\mathscr{M}}$, а затем для полученного предавтомата построим его глобализацию.

1. Для построения двойственного предавтомата $\widehat{\mathscr{M}} = (Z_H, X, \mu : Z_H \times X^* \dashrightarrow Z_H)$ определим полное отображение $\varphi : H \to Z_H$ такое что для любых $h \in H$ $\varphi(h) = z_h \in Z_H$ и выполняется следующее уравнение:

$$\forall w \in Dh,\ \varphi(\alpha(h(w))) = \mu(z_h, w)$$

И используя теорему 2.3 можно показать что такой предавтомат $\widehat{\mathscr{M}}$ является двойственным для регулярной машины обработки сложных событий $\mathscr{M}$.

2. Далее, построим глобализацию $(X, Z, \delta : Z \times X \to Z)$ для предавтомата $\widehat{\mathscr{M}}$. Для этого заметим что у каждого регулярного обработчика $h \in H$ есть реализация $((X, Z, \delta), z_0, Z_{acc}, \beta)_h$. Так что для всех обработчиков $h \in H$ обозначим начальное состояние как z_0 в их реализации $z_0 = z_h \in Z_H$ и $\mu(z_h, w) \downarrow = \delta^*(z_0, w) \in Z_{acc}$ для любых $w \in Dh$. Следовательно, $(X, Z, \delta) = \bigcup_{h \in H} (X, Z, \delta)_h$.

Более точно:

$$- Z = \bigcup_{h \in H} (Z \backslash Z_{acc})_h$$

$-\ \delta = \bigcup_{h \in H} \delta_h$.

Также легко заметить что $Z_H \subseteq Z$. Таким образом, автомат (X, Z, δ) является глобализацией для предавтомата $\widehat{\mathscr{M}} = (Z_H, X, \mu)$.

Очевидно что обратное утверждение "конечный автомат может быть преобразован в регулярную машину обработки сложных событий"также истинно. Поскольку всякий автомат может быть представлен как предавтомат [52] двойственный к регулярной машине обработки сложных событий. □

3.6 Метод обучения регулярной CEP-машины

В этой части рассматривается метод обучения регулярной CEP-машины представленный в [4, 5]. Заметим, что задачу обучения регулярной CEP-машины можно разделить на независимые подзадачи обучения ее обработчиков. Поскольку каждый регулярный обработчик имеет только один возможный ответ “accepted”, ниже будем называть такой обработчик регулярным акцептором. Тогда задача обучения регулярного акцептора может быть сформулирована следующим образом.

Задача 3.1. Пусть $E = \{u_1, \ldots, u_M\} \subset X^+$ это конечное префиксное множество событий, а $C = \{v_1, \ldots, v_N\} \subset X^+$ – конечное множество слов такие что $E \bigcap C = \varnothing$ тогда будем интерпретировать элементы множества E как примеры, а элементы множества C как контрпримеры;
необходимо найти регулярный акцептор $h\colon X^+ \dashrightarrow \{accepted\}$ такой что выполняются следующие условия:

1. $h(u_i)\downarrow = accepted$ для всех $0 \leqslant i < M$;
2. $h(v_i)\uparrow$ для всех $0 \leqslant i < N$; и
3. регулярный акцептор является минимальным. А именно, соответствующее множество Z имеет наименьшее количество элементов среди всех возможных.

Теперь рассмотрим взаимосвязь между регулярными акцепторами и де-

терминированными конечными автоматами. В частности, рассмотрим для любого регулярного акцептора $h : X^+ \dashrightarrow Y$ реализованного тройкой (Z, z_0, δ) автомат $(Q = Z \bigcup Y \bigcup \{q_{trap}\}, X, \overline{\delta} : Q \times X \to Q, q_0 = z_0, Q_{accept} = Y)$ где $q_{trap} \notin Z \bigcup Y$ и $\overline{\delta}(y,x) = q_{trap}$; $\overline{\delta}(q_{trap},x) = q_{trap}$ для любых $x \in X$ и $y \in Y$ тогда регулярный язык допускаемый данным автоматом совпадает с множеством событий допускаемых регулярным акцептором. Общая теория автоматов гарантирует что построенный автомат может быть единственным способом минимизирован не изменяя допускаемый язык. Заметим, что соответствующий минимальный автомат имеет только одно недопускающее состояние q такое что $\delta(q,x) = q$ для любого $x \in X$. Такое состояние называется ловушка и обозначается как `trap`. Далее, потребуем $\delta(q,x) = \texttt{trap}$ для любого допускающего состояния q и любого $x \in X$.

Теперь опишем предлагаемый метод обучения. Общий вид метода построения акцептора представлен на рисунке 3.1 в виде UML диаграммы активности. Более детальное описание метода представлено алгоритмом 3.2. Начиная с минимального акцептора построенного для множества примеров, данный метод определяет набор перенаправлений для переходов акцептора ведущих в состояние-ловушку. Алгоритм использует две функции: `init` – генерирует начальный акцептор используя множество примеров E; и `modify` – модифицирует акцептор таким образом что акцептор "обучается" новому событию. Эти функции приводятся отдельно.

Для завершения спецификации метода обучения приведем алгоритмы для функций `init` (см. шаг 3 алгоритма 3.2) и `modify` (см. шаг 7 алгоритма 3.2).

Для генерации минимального начального акцептора с помощью функции `init` будем использовать следующий алгоритм:

1. Состояния акцептора определяются рекурсивно как специальные множества слов;
2. выбираем множество E в качестве z_0 и добавляем его в Z;

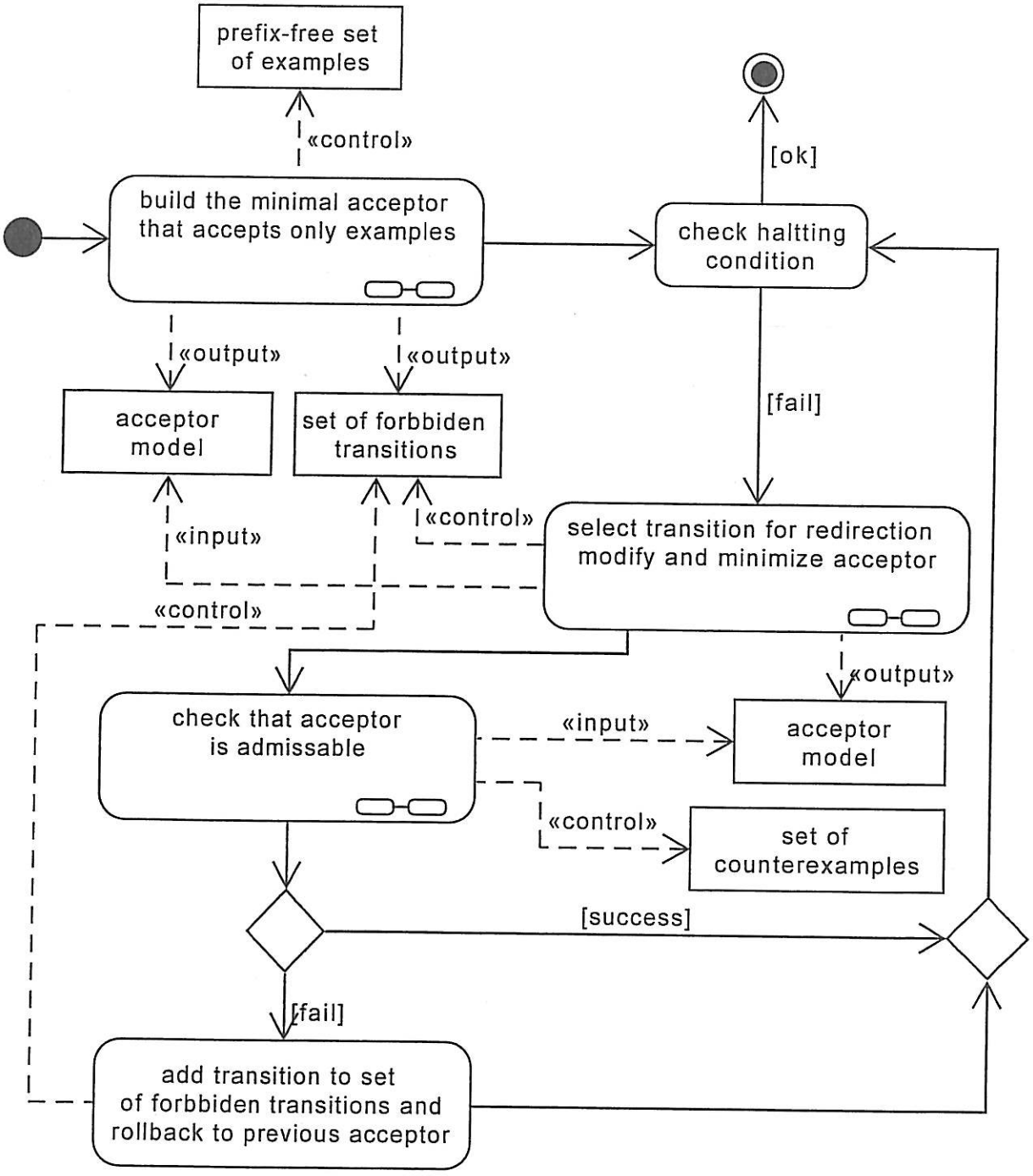

Рис. 3.1. Общая схема метода обучения регулярного акцептора

3. выбираем в качестве пустого множества `trap`;
4. если для $x \in X$ в E не найдется слово с первым символом x тогда присвоить $\delta(z_0, x) = \texttt{trap}$ иначе множество $\{u \in X^* \mid xu \in E\}$ добавить в Z;
5. рекурсивно повторять предыдущее действие для всех элементов множества Z пока Z не стабилизируется;
6. обозначить множество $\{\varepsilon\}$ как `accepted`.

Полученный таким образом акцептор присваивается как результат функции `init`.

Функция `modify` использует следующий алгоритм:

1. Зафиксировать $z = z_0$ множество допустимых событий и новое слово $w \in X^*$ которое необходимо выучить.
2. пока $w \neq \varepsilon$ выполнять:
3. добавить событие w если его еще нет в z
4. $x, w = w[0], w[1:]$
5. если существует переход x для пары $(z, z_x \in Z)$ тогда $z = z_x$. где $z_x = \{u \in X^+ | \forall\, v \in z : v = xu\}$ множество суффиксов полученное из множества z удалением символа x.
6. иначе если существует множество суффиксов $z_w \in Z$ такое что $w \in z_w$ тогда для пары (z, z_w) если не в `frozen_transitions` добавить переход обозначенный как x и зафиксировать $z = z_w$
7. иначе добавить новое состояние $z_{new} \in Z : z_{new} = \{w\}$ и для пары (z, z_{new}) добавить переход обозначенный как x и зафиксировать $z = z_{new}$
8. готово.

Для выбора перехода который будет перенаправлен заметим, что у минимального регулярного акцептора есть только одно состояние – аттрактор, т.е. любой переход выходящий из аттрактора переводит в тот же аттрактор. Более того, если акцептор допускает конечный язык тогда аттрактор га-

рантировано существует. Далее, для минимизации полученного акцептора будем использовать стандартный алгоритм Хопкрофта (Hopcroft's algorithm) [82].

Алгоритм Хопкрофта. Алгоритм Хопкрофта, объединяющий избыточные состояния детерминированного конечного автомата, основывается на уточнении наборов состояний (partition refinement), разделяя состояния ДКА на группы в зависимости от их поведения. Эти группы представляют собой классы эквивалентности отношения Майхилла-Нерода (Myhill–Nerode equivalence relation), в которых каждые два состояния являются эквивалентными если они генерируют одинаковое поведение на всех входных последовательностях символов. А именно, для каждых двух состояний p_1 и p_2 принадлежащих одному классу эквивалентности P, и для каждого входного слова w, переходы определяемые словом w всегда должны переводить состояния p_1 и p_2 в равные состояния, которые одновременно должны быть либо допускающими, либо не допускающими. Другими словами, w не должно переводить p_1 в допускающее состояние, а состояние p_2 – в не допускающее состояние, и наоборот. Псевдокод ниже описывает алгоритм:

1. $P := \{Z_{acc}, Z \backslash Z_{acc}\}$;
2. $W := \{Z_{acc}\}$;
3. while (W is not empty) do
4. choose and remove a set A from W

   for each c in X do
5. let Z_A be the set of states for which a transition on c leads to a state in A

   for each set Y_A in P for which $Z_A \cap Y_A$ is nonempty and $Y_A \backslash Z_A$ is nonempty do
6. replace Y_A in P by the two sets $Z_A \cap Y_A$ and $Y_A \backslash Z_A$

7. if Y_A is in W replace Y_A in W by the same two sets
8. else if $|Z_A \cap Y_A| \leqslant |Y_A \backslash Z_A|$ add $Z_A \cap Y_A$ to W
9. else add $Y_A \backslash Z_A$ to W
10. end for each cycle;
11. end for each cycle;
12. end while cycle;

Алгоритм начинает работу с грубого набора состояний: каждая пара состояний, эквивалентная согласно отношению эквивалентности Майхилла-Нерода, принадлежит одному и тому же множеству в наборе, однако не эквивалентные пары состояний могут также принадлежать данному множеству. Постепенно алгоритм уточняет наборы множеств состояний, увеличивая их количество, путем разделения на каждом шаге множеств состояний на пары подмножеств, которые являются обязательно неэквивалентными. Начальный набор состоит из двух подмножеств состояний, которые, очевидно, приводят к различному поведению автомата, то есть к допускающим и недопускающим состояниям соответственно. Затем, алгоритм периодически выбирает множество A из текущего набора и входного символа c, а затем, разделяет каждое множество набора на два (возможно пустых) подмножества: на подмножество состояний приводящих в множество состояний A по входному символу c, и на подмножество состояний не приводящих в A соответственно. Поскольку, про множество A уже известно, что оно приводит автомат к отличному от других множеств набора поведению, то и подмножества ведущие в A также генерируют поведение автомата отличное от подмножеств не ведущих в A. Когда алгоритму не удается разделить множества в наборе состояний, он останавливается.

Лемма 3.1. Пусть задан зафиксированный символ c и класс эквивалентности Y который разделяется на классы эквивалентности B и C, тогда достаточно только одного класса эквивалентности B или C для уточнения

всего набора состояний [83].

Пример 3.2. Предположим, что имеется класс эквивалентности Y, который разделяется на классы эквивалентности B и C. Также, предположим, что имеются классы эквивалентности D, E и F; D и E содержат состояния из которых имеются переходы в B по символу c, тогда как F содержит состояния имеющие переходы в C по символу c. По лемме 3.1, можно равносильно выбрать B или C как генерирующий отличное поведение. Для определенности выберем B. Тогда состояния из D и E разделяются своими переходами на B. Однако, F, который не имеет переходов в B, не будет разделен в текущей итерации алгоритма; он будет уточнен на следующих итерациях.

После использования алгоритма Хопкрофта для группирования состояний входного ДКА в классы эквивалентности, минимальный ДКА может быть сконструирован путем формирования одного состояния для каждого класса эквивалентности. Если S – это множество состояний из P, s – состояние из S, а c – входной символ, тогда переход в минимальном ДКА из состояния для S, по входному символу c, идет во множество, содержащее состояние в которое переходил бы исходный ДКА из состояния s по входному символу c. Начальным состоянием минимального ДКА является состояние содержащее начальное состояние исходного ДКА, а допускающие состояния минимального ДКА – это те, чьи элементы являются допустимыми состояниями в исходном ДКА.

3.7 Верификация правильности метода обучения

В этой части раздела в деталях представлен компьютерный эксперимент, разработанный для верификации правильности метода обучения регулярного акцептора. Схема эксперимента представленна UML диаграмой на рис. 3.2. Алгоритм 3.3 описывает пошаговое выполнение эксперимента.

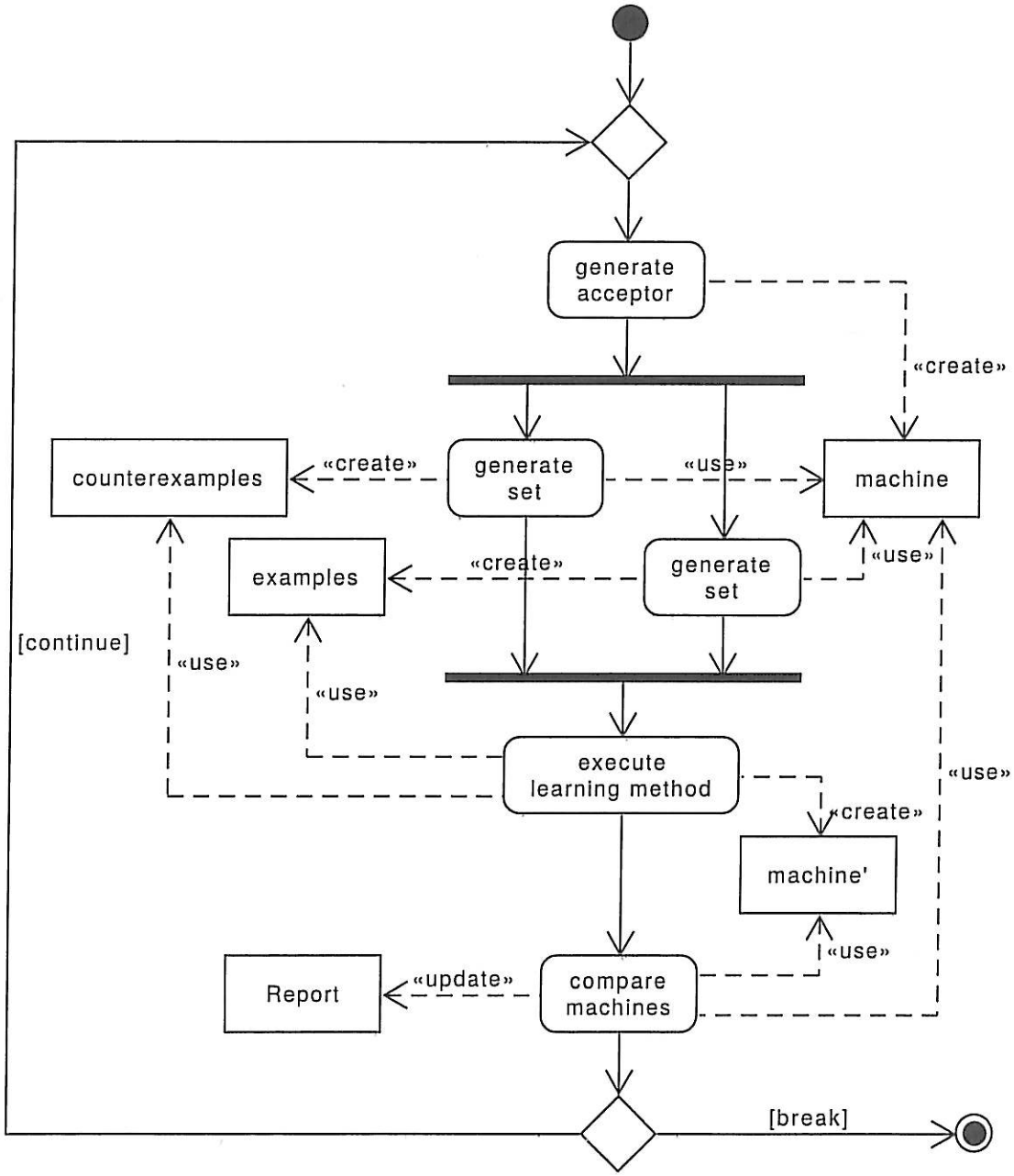

Рис. 3.2. Общая схема эксперимента верификации обучающего метода

Ключевой точкой верификации является генерация случайным способом акцептора `test_acceptor`. Затем этот акцептор используется для задания множеств примеров E и контрпримеров C. Для генерации такого акцептора `test_acceptor` сначала строится синтаксическое дерево, которое затем преобразовывается в регулярный акцептор.

3.8 Имплементация эксперимента

Для имплементации приведенной схемы эксперимента используется язык программирования Python, а также библиотеки "scipy" и "numpy" [84]. В частности, выбор всех случайных значений осуществляется с помощью стандартной функции `random.choice` из библиотеки "numpy".

Для случайной генерации регулярных выражений используется следующая схема.

Структуры данных: Для генерации регулярных выражений будем использовать две структуры данных, а именно,

1. будем использовать следующий словарь (в терминах языка Python) для хранения константных данных

```
model_frame = {
    'tokens': (  # this tuple contains a list of
                 # used tokens
        . . . . . .
    ),
    'functors': (  # this tuple contains operations
                   # and their arities
        ('star', 1),           # Kleene's star
        ('concatenation', 2)),
        ('union', 2)
    )
```

```
}
```

2. Для представления синтаксического дерева регулярного выражения будем использовать такую рекурсивную структуру :

```
expression = token
           | (functor, expression, ...)
```

где количество выражений после функтора равняется арности этого функтора. например, только одно выражение следует после функтора "`star`".

Легко видеть, что выражение представлятся деревом. Каждый лист этого дерева представляется как токен, а каждый внутренний узел как функтор. Более того, количество потомков каждого внутреннего узла равняется арности функтора помечающего этот узел.

Схема построения случайного синтаксического дерева: рекурсивная структура дерева, которое представляет некоторое регулярное выражение, указывает способ построения такого случайного синтаксического дерева описаный алгоритмом 3.4. Для принятия решения о том является ли корень текущего поддерева внутренним узлом или листом (шаг 4 алгоритма 3.4) используется функция $p(n)$, которая определяет условную вероятность того отмечать ли текущий узел как внутренний в зависимости от его глубины n

$$p(n) = \begin{cases} 1 - \frac{1}{2}\left(\frac{n}{\mu}\right)^2 & \text{если } 0 \leqslant n < \mu \\ \frac{1}{2}\exp\left(1 - \frac{n}{\mu}\right) & \text{если } n \geqslant \mu \end{cases} . \tag{3.1}$$

Приведенный метод построения предоставляет хорошо сбалансированные синтаксические деревья, но данный метод не контролирует наличие дисбаланса при обозначении узлов. Следовательно, для исправления возможного дисбаланса в узлах предлагается следующий способ выполнения шагов 7 и 11 алгоритма 3.4: используется стандартная функция `random.choice`,

находящаяся в пакете "random" библиотеки "numpy", для выбора элемента из списка $[i_1, i_2, \ldots, i_k]$ при условии что вероятность выбора i_j обратно пропорциональна количеству уже сделанных выборов данного элемента. Такая поправка гарантирует "равномерность" распределения при обозначении узлов как внутренних либо листьев.

Теперь, можно использовать следующую процедуру $TransformSyntaxTreeToFSM(tree)$ для преобразования синтаксического дерева в детерминированый конечный автомат. А затем полученный автомат может быть преобразован в регулярный акцептор путем объединения всех допустимых состояний.

1. if $SyntaxTreeNode.GetType() == leaf$ then:
) $symbol = SyntaxTreeNode.GetValue()$;
) $FSM.AddStartState(q_0)$;
) $FSM.AddTerminalState(q_{symbol})$;
) $FSM.AddArrow(q_0, symbol, q_{symbol})$;
) Return FSM;
2. else if $SyntaxTreeNode.GetType ==' star'$ then:
) $childNode = SyntaxTreeNode.GetChild()$;
) $FSM = TransformSyntaxTreeToFSM(childNode)$;
) for each terminal arrow $arrow \in FSM.GetAllTerminalArrows()$:
) $arrow.replaceEndStateTo(q_0)$;
) end for each loop;
) Return FSM;
3. else if $SyntaxTreeNode.GetType ==' union'$ then:
) $childNode_0 = SyntaxTreeNode.GetChild(0)$;
) $childNode_1 = SyntaxTreeNode.GetChild(1)$;
) $subFSM_0 = TransformSyntaxTreeToFSM(childNode_0)$;
) $subFSM_1 = TransformSyntaxTreeToFSM(childNode_1)$;
) $FSM = Merge(subFSM_0, subFSM_1)$;

) Return FSM;

4. else if $SyntaxTreeNode.GetType ==' concatenation'$ then:

) $childNode_0 = SyntaxTreeNode.GetChild(0)$;

) $childNode_1 = SyntaxTreeNode.GetChild(1)$;

) $subFSM_0 = TransformSyntaxTreeToFSM(childNode_0)$;

) $subFSM_1 = TransformSyntaxTreeToFSM(childNode_1)$;

) $FSM = Concatenate(subFSM_0, subFSM_1)$;

) Return FSM;

Заметим, что операция $Merge()$ объединяет два конечных автомата начиная с начального состояния удаляя при этом повторяющиеся переходы. Тогда как операция $Concatenate()$ просто объединяет все терминальные состояния первого автомата с начальным состоянием второго.

Результаты более чем 10 000 экспериментов показывают что для случайным образом сгенерированого акцептора и полученных множеств E и C, представленный обучающий метод восстанавливает данный акцептор используя множества примеров и контрпримеров.

Следовательно, можно предположить что рассмотренный обучающий метод точен для регулярных акцепторов. Последнее предположение может рассматриваться как свидетельство правильности представленного обучающего метода.

Выводы по разделу 3

В разделе рассматривались системы обработки сложных событий, у которых потоки событий можно описать регулярным языком. В части 3.2 рассматривался некоторый специальный класс обработчиков сложных событий. Такие обработчики событий были названы регулярными. Теорема 3.1 доказывает что регулярный обработчик может быть реализован с помощью конечного автомата.

В части 3.4 представлен важный класс CEP-машин, названных регулярными. Приведен алгоритм описывающий поведение таких машин. Теорема 3.3 доказывает что регулярная CEP-машина может быть преобразована в автомат, который является глобализацией двойственного предавтомата этой регулярной CEP-машины.

Полученные результаты могут быть полезны в моделировании и разработке систем обработки сложных событий, способных распознавать такие события, которые можно описать регулярным языком.

Далее, в разделе рассматривался метод машинного обучения для регулярных обработчиков с одним допускающим состоянием (регулярных акцепторов). Также была детально описана процедура верификации правильности метода. Было показано что этот метод может быть использован при проектировании и разработке систем основанных на архитектуре управляемой событиями. Преимуществом метода является то что анализ всех возможных комбинаций событий может быть заменен обучением на этапе выполнения. Компьютерный эксперимент основанный на процедуре верификации правильности показывает что обучающий метод восстанавливает тестируюмую CEP-машину. Данные результаты были также опубликованы в источниках [2, 5, 6].

Алгоритм 3.2. Specification of the learning method

```
1  def learning_method(E,C):
       Require: the finite prefix-free set of events E
                the finite set of words that are not events C
       Ensure : the required acceptor
2      do that:
3          initiate the learning process by applying function init to the set
           E and denoting the result by acceptor
       # initialize the set of transitions that cannot be
         redirected
4      frozen_transitions = set()
5      while halting condition is not fulfilled:
6          do that:
7              modify acceptor by redirecting a transition leading into the
               trap and minimize the resulting acceptor if the redirected
               transition is not in frozen_transitions
8          do that:
9              check that acceptor is admissible in the sense that it does
               not accept any word from C
10         if the checking is successful: continue
11         else:
12             do that:
13                 roll-back the modification and add the redirected
                   transition into frozen_transitions
```

Алгоритм 3.3. The schema of the computer experiment

```
for _ in range(given_number_of_experiments):
    do that:
        generate randomly a regular acceptor test_acceptor
    do that:
        generate randomly sets E and C using test_acceptor
    acceptor' = learning_method(E,C)
    do that:
        compare acceptor' and test_acceptor
```

Алгоритм 3.4. A tree random generation

```
def create_tree(depth = 0):
    global model_frame
    do that:
        decide whether the tree root is an internal node or a leaf
    if the tree root is a leaf:
        do that:
            choose randomly token
        return token
    else:
        do that:
            choose randomly functor
        arity,temp = model_frame['functors'][functor][1],[functor]
        for _ in range(arity): temp.append(create_tree(h+1))
        return tuple(temp)
```

РАЗДЕЛ 4
ИСПОЛЬЗОВАНИЕ МОДЕЛИ РЕГУЛЯРНОЙ СЕР-МАШИНЫ В ЗАДАЧАХ УПРАВЛЕНИЯ ТРАФИКОМ В РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

В данном разделе рассматриваются задачи управления трафиком в распределенных информационных системах (РИС). Существует множество подходов к управлению и оптимизации трафика. В частности, формирование трафика (traffic shaping) является одним из нескольких методов управления полосой пропускания в таких системах [85]. В данном разделе показывается, что результаты полученные в разделах 2 и 3 могут широко использоваться при проектировании систем формирования и оптимизации трафика (traffic shapers). В частности, применение данных подходов в оптимизации архитектуры виртуальных маршрутизаторов дает существенный рост производительности.

4.1. Базовые понятия и обозначения

Приведем необходимые понятия и обозначения, используемые в данном разделе. Как и в предыдущем разделе, определим алфавит X как множество символов. В данном контексте каждый символ $x \in X$ будем интерпретировать как атомарный пакет данных передаваемый по сети. Например, такой пакет может представлять собой фрагмент аудио, видео либо любых других данных. Таким образом, можно определить множество X как объединение:

$$X = \bigcup_{i=1}^{n} \{A^i \bigcup V^i \bigcup D^i \bigcup M^i\}$$

Где :

— A^i множество типов аудиопакетов генерируемых i-м пользователем сети.

— Аналогично, V^i множество типов видеопакетов.

— D^i множество пакетов других данных, например, таких как сообщения электронной почты или веб-контент.

— И наконец, M^i множество служебных сообщений. Такие сообщения могут содержать диагностическую информацию о состоянии узла, либо управляющие команды другим узлам сети.

Замечание 4.1. Заметим, что количество типов данных может быть выбрано произвольно и зависит от требований к качеству передачи для тех или других данных. Так, например, тип пакетов D^i может в свою очередь быть представлен как объединение множеств пакетов представляющих полезный трафик и нежелательный (спам). В нашем случае в дальнейшем в качестве примера будет рассматриваться некоторая сеть мобильной связи в которой качество передачи видео и аудио данных играет ключевую роль, в то время как, скорость передачи других данных может ограничиваться с целью максимального улучшения передачи изображения и звука. Другим критерием качества обслуживания может быть зависимость от типа тарифного плана клиента. Также заметим, что в данном контексте пользователем может быть как некоторый узел сети (базовая станция), так и некоторое приложение на мобильном устройстве генерирующее пакеты.

X^+ – множество конечных, непустых последовательностей пакетов. ε – пустая последовательность. Тогда $X^* = X^+ \bigcup \varepsilon$. Далее, аналогично обозначению в разделе 3, обозначим как X^ω множество бесконечных потоков пакетов данных. Тогда весь трафик можно обозначить как $X^\infty = X^* \bigcup X^\omega$. Теперь рассмотрим существующие подходы к управлению РИС и оптимизации трафика.

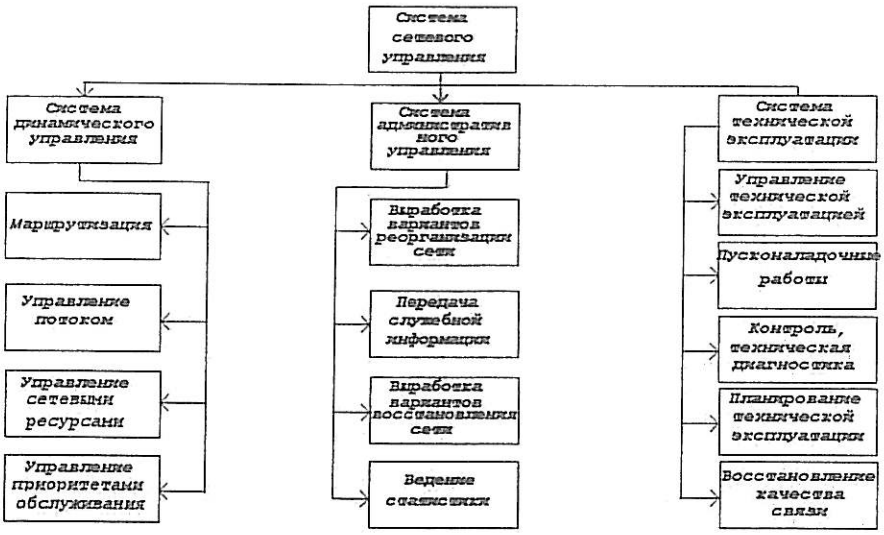

Рис. 4.1. Структура системы сетевого управления

4.2 Анализ методов сетевого управления

Характеристика средств управления распределенными информационными системами. Производительность распределенных информационных систем, построенных на принципах New Generation Networks (NGN), во многом зависит от эффективности реализации функций системы сетевого управления (ССУ). В сетях следующего поколения система сетевого управления осуществляет управление на каждом из трех уровней (транспортном, коммутации и передачи информации) в подсистемах административного управления, технической эксплуатации и динамического управления (рис. 4.1) [86].

Система динамического управления обеспечивает распределение потоков информации с целью наилучшего использования ресурсов сети при удовлетворении требований абонентов. Она основана на иерархической структуре протоколов. Эта система осуществляет управление потоками пакетов как находящихся в РИС на обслуживании путем выбора допустимых маршрутов передачи сообщений, так и на этапе ввода в РИС за счет запрета

доступа. Технической базой системы динамического управления являются узлы коммутации.

Система административного управления предназначена для выбора вариантов реорганизации структуры РИС в случае массовых разрушений объектов РИС, отображения ее состояния на рабочих местах операторов, ведения статистики передачи служебной информации. Кроме того, она обеспечивает отображение состояния элементов и всей сети, а также принятие решения на устранение возникающих неисправностей. В процессе набора статистики должны быть получены такие интегральные вероятностно-временные характеристики, как вероятность и время доставки сообщений, а также данные об использовании каналов и центров коммутации в различное время. Данная информация необходима для принятия решения о реорганизации РИС [87]. В случае выхода из строя элементов РИС требуется определить степень разрушения и возможности восстановления элементов, возможность получения дополнительных ресурсов из каналов и трактов первичной передачи, использования резервных узлов коммутации и линий связи. В случае недостатков ресурсов сети должна быть предусмотрена возможность принудительного отключения некоторых пользователей. Техническую базу системы административного управления составляет совокупность пунктов управления сетью и элементами сети, совмещенных с узлами связи. Автоматизированная система технической эксплуатации (АСТЭ) представляет собой совокупность методов управления, программных и технических средств, людских и материальных ресурсов, осуществляющих техническую эксплуатацию по принятым показателям качества управления. Основной целью АСТЭ является минимизация трудовых затрат при обеспечении необходимого качества обслуживания. Ее основные функции заключаются в организации пуско-наладочных и ремонтных работ, управлении технической эксплуатацией, в контроле и диагностике технического состояния сети, восстановлении качества связи, планировании техническо-

го обслуживания [87]. Ввиду ограниченности сетевых ресурсов (буферного пространства на узлах сети, пропускной способности трактов передачи, вычислительных мощностей и времени на принятие управленческих решений) для реализации перечисленных преимуществ мультисервисная РИС в рамках ССУ должна обладать эффективными средствами и способами управления сетевыми ресурсами. Кроме того, эффективное распределение сетевых ресурсов между большим числом различных типов информационных потоков, предъявляющих различные требования по качеству обслуживания, является очень сложной задачей [88]. Следовательно, эффективность распределенных информационных систем, построенных на принципах NGN, во многом зависит от эффективности решения задач управления сетевыми ресурсами, которые в первую очередь направлены на обеспечение Quality of Service (QoS) [88]. Средства управления сетевыми ресурсами должны принимать самое активное участие в процессе функционирования уровня транспорта и уровня доступа [88, 89].

Анализ средств управления трафиком. Как было показано ранее, система динамического управления предназначена для решения задач маршрутизации, управления потоками, управления сетевыми ресурсами и управления приоритетами обслуживания. Рассмотрим подробнее средства и способы решения этих задач.

В настоящее время наиболее распространенными используемыми средствами управления трафиком являются протоколы динамической маршрутизации, такие как: RIP (Routing Information Protocol), IGRP (Interior Gateway Routing Protocol), OSPF (Open Shortest Path First), IS-IS (Intermediate System to Intermediate System), BGP (Border Gateway Protocol), EGP (Exterior Gateway Protocol) в IP сетях [90] и PNNI (Private Network to Network Interface), IISP (Interim Inter-Switch Protocol) в сетях ATM [91].

Широкое использование этих протоколов маршрутизации обусловле-

но их высокой оперативностью и надежностью, относительной простотой реализации. Главным свойством протокола маршрутизации при решении задач по обеспечению гарантированного качества обслуживания является поддержка множества метрик, которые напрямую или косвенно связаны со скоростными, временными и вероятностными показателями качества обслуживания. С целью реализации многопутевых стратегий маршрутизации ряд существующих протоколов маршрутизации IGRP, EIGRP, IS-IS, OSPF и PNNI дополнены процедурами балансировки нагрузки, а также другими методами управления потоками на узлах РИС. Одним из мощных средств обеспечения эффективного использования сетевых ресурсов является применение технологии Traffic Engineering (TE). Под TE понимаются методы и механизмы достижения сбалансированности загрузки всех ресурсов сети за счет рационального выбора путей прохождения трафика через РИС [87].

Анализ алгоритмов управления потоками. В большинстве РИС возникают ситуации, когда поступающая извне нагрузка больше той, которая может быть обслужена даже при оптимальной маршрутизации [92]. Тогда, если не предпринять никаких мер по ограничению поступающего трафика, размеры очередей быстро увеличиваются, что приводит к увеличению задержки передачи пакетов, часть пакетов может быть потеряна. Поэтому возникает необходимость использования дополнительных способов, позволяющих управлять потоками в сети. Главными целями управления потоками являются [93]:

— установление подходящего компромисса между ограничением пользователей и удержанием средней задержки пакета на приемлемом уровне;

— соблюдение справедливости по отношению ко всем пользователям, когда часть поступающего трафика не допускается в сеть.

В основном вопрос управления потоком возникает, когда имеется ограничение на скорость передачи между двумя точками вследствие ограничен-

ной пропускной способности линии передачи или узла коммутации. Таким образом, схема управления потоком может потребоваться на участках передачи между двумя пользователями (транспортный уровень), между пользователем и входной точкой подсети (сетевой уровень) или между двумя узлами подсети (сетевой уровень). Способы управления потоками по месту применения можно разбить на две группы: базирующихся в маршрутизаторах и базирующихся на конечных узлах. В первом случае каждый маршрутизатор принимает самостоятельное решение о том, какой пакет продвигать дальше, а какой отбросить. Во втором случае решение о продвижении пакета принимается на конечном узле с учетом состояния сети. Другой способ классификации способов управления потоками позволяет разбить их на две группы [92]: с использованием резервирования и с использованием обратной связи. В сетях, использующих резервирование, узел-отправитель запрашивает у сети необходимые для передачи трафика ресурсы. Каждый маршрутизатор проверяет возможность резервирования необходимых ресурсов. Если необходимых ресурсов недостаточно, то входной поток отбрасывается. В системах, использующих обратную связь, узел отправляет информацию без предварительного резервирования необходимых ресурсов. В дальнейшем узел-отправитель при помощи обратной связи получает информацию о состоянии сети. На основании этой информации узел-отправитель управляет скоростью отправки данных [92]. Механизмы, использующие обратную связь, в свою очередь, можно разделить на механизмы с явной и неявной обратной связью. Неявная обратная связь подразумевает, что источник нагрузки получает информацию через узел-приемник и процесс мониторинга параметров передачи нагрузки осуществляется обоими узлами, и они же ответственны за определение состояния сети, и, соответственно, за определение скорости передачи [92]. Механизм явной обратной связи должен явно информировать источник нагрузки о состоянии сети. Существует два типа явной обратной связи: уведомление

о перегрузке и индикация скорости. Возможности механизма сильно ограничены тем, что его информация переноситься, как правило, в заголовках пакетов, а их размер и количество используемых бит ограничено. Явная обратная связь может быть как бинарной – уведомление о перегрузке, так и многозначной. Обычно количество значений ограничено, т. е. существует возможность сообщить источнику перегрузки об определенном уровне перегрузки. Примером этого механизма для протоколов TCP/IP является флаг ECN и сообщения ICMP Source Quench. Другой способ классификации способов управления потоками позволяет разделить механизмы на две группы [92, 94]: с использованием окон и управлением скоростью передачи. В первом случае, получатель указывает количество данных, которое отправитель может отправить. Во втором случае, отправитель указывает скорость, с которой отправитель должен отправлять свои данные. Большинство современных протоколов используют оконное управление потоками [93, 94]. Различают оконное управление от конца до конца и поузловое оконное управление. Эти методы позволяют регулировать интенсивность потока пакетов в зависимости от загруженности сети в целом, отдельных ее участков или сетевых устройств. Главная трудность таких способов управления потоком состоит в том, что они не могут хорошо справляться с быстрыми изменениями предлагаемой нагрузки различных сеансов [92]. Для решения этой проблемы используют управление потоком с временным окном. Суть этого способа состоит в том, чтобы ограничить максимальную скорость передачи трафика (traffic shaping) и сгладить входной поток (traffic policing). Существуют также методы явного уведомления источника трафика о перегрузке сети, когда источнику отправляется уведомление о перегрузке с требованием уменьшить интенсивность входного потока [92].

4.3 Задача формирования трафика

В данной части приводятся определения и базовые понятия необходимые для постановки задачи формирования трафика в распределенных информационных системах.

Основные способы формирования трафика. В начале приведем определение термина "формирование трафика".

Определение 4.1 (Формирование трафика). Термин формирование трафика (также известный как формирование пакетов) – это техника управления трафиком в сетях, которая заключается в задержке некоторых или всех датаграмм (datagrams) для приведения их в оптимальное состояние удовлетворяющее возможностям передающего канала.[95, 96] Формирование трафика используется для оптимизации или гарантирования производительности, улучшения скорости передачи, или увеличения используемой полосы пропускания для некоторых типов пакетов путем задержки отправки пакетов других типов.

Часто техника формирования трафика используется вместе с техникой мониторинга трафика (traffic policing), которая заключается в проверке соответствия передаваемых данных неким условиям (политикам). Данные не удовлетворяющее условиям обычно удаляются или маркеруются специальным образом [97]. Существует несколько основных подходов к формированию трафика:

— В общем случае, формирование трафика основывается на анализе приложений, которые его генерируют [98]. В случае формирования трафика генерируемого приложениями, в начале используются инструменты определения типов этих приложений (fingerprinting tools), а затем трафик формируется согласно заданным правилам для определенных типов. В некоторых случаях использование формирования трафика, генерируемого приложениями, является противоречивым. Например, изменение ширины по-

лосы пропускания для P2P приложений. Многие прикладные протоколы используют шифрование чтобы обойти процедуру формирования трафика, т.к. в таком случае не возможно определить тип данных. В нашем случае, данная проблема может быть решена путем привязки генерируемых пакетов к IMEI мобильного устройства либо к идентификатору базовой станции во всей сети либо в ее сегменте. Или такой трафик может быть помечен как нежелательный, в следствие чего скорость доставки таких данных будет минимальной.

— Другой тип формирования трафика основывается на анализе маршрута данных. Т.е. проводится анализ передающих каналов для определения оптимальной пропускной способности. Анализ формирования трафика в узле основанного на маршрутизации обычно проводится с использованием информации о загруженности и пропускной способности предыдущего или следующего канала связи, через которые проходит данный трафик [99]. Формирование пакетов обычно применяется в передающих каналах сети для контроля трафика входящего в сеть, но также может применяться источниками трафика (например, компьютером или сетевой картой[100]) или любым другим элементом сети. Также техника формирования трафика широко используется интернет провайдерами при проектировании и отладке сетей. Интернет провайдеры часто используют формирование трафика в локальных сетях как один из нескольких инструментов управления трафиком [101]. Некоторые провайдеры могут использовать формирование трафика для блокирования peer-to-peer файлообменных сетей, таких как BitTorrent [102].

— Другой способ формирования трафика это ограничение приложениями генерируемого ими трафика. Такие приложения генерируют трафик который никогда не превосходит некоторое максимальное значение. Например, медиа приложения, генерирующие аудио и видео трафик, не могут передавать по сети больше данных в единицу времени чем позволяет скорость их

кодирования [103]. Однако, ключевым недостатком такого подхода является то, что, как правило, операционная система, в которой выполняется такое приложение, является многозадачной, что в результате приводит к тому что одновременно могут быть запущены несколько экземпляров данного приложения. Что в свою очередь приведет к генерации избыточного трафика и перегрузке сети. Другой пример – это TCP Nice, модифицированная версия TCP протокола, разработанная исследователями Техасского университета в Остине, которая позволяет приложениям требовать чтобы некоторые TCP соединения управлялись операционной системой таким образом, чтобы минимизировать использование ресурсов. Т.е. система должна генерировать оптимизированные потоки трафика ("nice"flows). Такие оптимизированные потоки трафика почти не пересекаются с основными потоками (non-nice flows), в тоже время они используют большую часть свободной пропускающей полосы сети [104].

Способы имплементации механизмов формирования трафика. Утилиты формирования трафика работают по принципу задержки измеряемого трафика так, чтобы каждый пакет удовлетворял соответствующим ограничениям налогаемым на трафик. Измерение трафика может быть реализовано с помощью, например, алгоритмов текущего ведра [105, 106] или ведра токенов [107].

Определение 4.2 (Алгоритм текущего ведра). "Текущее ведро"это алгоритм который может использоваться для определения того соответствует ли некоторая последовательность дискретных событий заданным ограничениям на их среднее либо пиковое значение или частоту появления [108, 109]. Алгоритм текущего ведра используется в компьютерных сетях с коммутацей пакетов и в телекоммуникационных сетях для проверки того что передача данных в форме пакетов удовлетворяет заданным ограничениям на ширину полосы пропускания и пакетирования (степень нерав-

номерности или изменение свойств потока трафика) [110]. Также данный алгоритм может использоваться как механизм планирования для определения времени передачи данных, которое будет соответствовать ширине полосы пропускания и пропускной способности сети [111].

Измеряемые пакеты или кадры затем добавляются в очередь (FIFO buffer), для каждого класса данных формируемую отдельно, до тех пор когда появится возможность передать накопленные данные в соответствии с ограничениями на передачу таких данных. Передача данных может произойти:

— немедленно, если условия на передачу такого трафика выполняются на момент его поступления;

— после некоторой задержки, данные ожидают в очереди согласно заданного расписания отправки;

— никогда, в случае переполнения очереди.

Проблема переполнения при формировании трафика. Все реализации утилит формирования трафика имеют ограниченный объем памяти для входящих данных и должны уметь обрабатывать событие переполнения очереди. Простым и стандартным подходом является удаление входящего трафика в момент переполнения очереди (удаляется хвост очереди). Более сложные реализации могут применять специальные алгоритмы удаления избыточного трафика такие как алгоритм произвольного раннего обнаружения (Random Early Discard, RED) [112]. Существует несколько модификаций алгоритма произвольного раннего обнаружения. Каждая модификация имеет свои достоинства и недостатки [113, 114, 115]. Приведем наиболее распространенные модификации:

— WRED (Weighted random early detection): В алгоритме взвешенного произвольного раннего обнаружения можно определить различные вероятности удаления пакетов для различных данных с заданным приоритетом (IP

precedence, DSCP) либо очередей [116].

— ARED (Adaptive/Active Random Early Detection): алгоритм адаптивного или активного произвольного раннего обнаружения [117] управляет интенсивностью удаления пакетов в зависимости от длины очереди. Если средняя длина очереди колеблется вокруг некоторого минимального порогового значения, то это значет что раннее обнаружение слишком активно. С другой стороны, если средняя длина очереди колеблется вокруг максимального порогового значения, то это означает что раннее обнаружение слишком пассивно. Алгоритм изменяет вероятность удаления пакетов в зависимости от уровня своей активности [118].

— RRED (Robust random early detection): алгоритм надежного произвольного раннего обнаружения был разработан для улучшения пропускной способности TCP во время атак отказа в обслуживании (Denial-of-Service, DoS) [119].

Другим элементарным способом обработки переполнения является пересылка не сформированного трафика для которого не хватило места в очереди. Очевидно, что такой подход приведет к уменьшению скорости передачи приоритетных данных, что в свою очередь может повлеять на качество обслуживания. С другой стороны, такой подход уменьшает количество повторных передач данных, что в свою очередь помогает уменьшить пиковую нагрузку.

Способы классификации трафика при его формировании. При использовании простых схем формирования трафика входящие данные обычно формируются однородно согласно определенной для них максимальной скорости передачи. Более сложные системы формирования трафика вначале классифицируют входные данные.

Определение 4.3 (Классификация трафика). Классификация трафика это автоматизированный процесс выделения различных типов (классов)

данных из поступающего в сеть трафика согласно множества различных параметров [120]. Каждый полученный класс данных может обрабатываться различным способом что позволяет предоставлять различным пользователям персональные сервисы.

В самом простом случае трафик может классифицироваться по номеру порта или типу протокола. Различные классы данных могут затем подвергнуться дополнительному формированию для получения ожидаемого эффекта. Планировщик сети также может обрабатывать различные классы пакетов различными способами. Классифицируя потоки трафика согласно типа протокола, можно применять предопределенную политику управления потоками гарантирующую определенный уровень качества передачи данных, например, для таких сервисов как VoIP или Media Streaming [121], по отношению к другим типам данных. Такой механизм управления может применяться на входе в сеть с детальностью позволяющей разделить трафик на независимые потоки, которые затем буферизируются, анализируются и формируются раздельно [122]. Приведем наиболее распространенные способы классификации трафика:

— классификация трафика по номеру порта: быстрая, потребляет мало вычислительных ресурсов, поддерживается большинством сетевых устроиств, не зависит от содержимого передаваемых данных, полезна для приложений и сервисов использующих фиксированые номера портов, легко блокируемая путем изменения номера порта в системе.

— Глубокое инспектирование пакетов: проверяет фактическое содержимое пакетов, определяет тип приложений и сервисов в независимости от номера порта на котором они работают, не поддерживает множество приложений, не производительное, требует больших вычислительных ресурсов, требует частых обновлений механизмов распознавания приложений, во многих случаях шифрование делает использование данного способа классификации не

возможным. Примером обнаружения приложений является отслеживание некоторых специальных шаблонов данных передаваемых Bit Torrent приложениями на этапе установки соединения [123]. Сравнительный анализ систем глубокого инспектирования пакетов представлен в [124].

— Статистическая классификация: опирается на статистический анализ частоты появления тех или иных последовательностей байтов, размера пакетов и время их прихода [125]. Часто использует алгоритмы машинного обучения. Может обнаруживать классы ранее неизвестных приложений.

Теперь рассмотрим основные классы трафика:

— Чувствительный трафик (Sensitive Traffic): время доставки данных такого типа является критичным. Примерами такого трафика является VoIP, сетевые игры, видео конференции и веб-данные. Схемы управления трафиком обычно конфигурируются так, чтобы гарантировать качество доставки класса таких данных или по крайней мере передавать такие данные с более высоким приоритетом.

— Полезный трафик (Best Effort Traffic): другие данные не являющиеся вредоносными. Это трафик который считается нечувствительным к метрикам измерения качества сервисов (неустойчивость, потеря пакетов, задержка). Типичными примерами являются peer-to-peer и почтовые приложения.

— Нежелательный трафик (Undesired traffic): категория данных этого типа обычно ограничивается трафиком генерируемым спамом, вирусными приложениями типа "червь"(worm), сетями ботов (botnets) и другими хакерскими атаками. В таких случаях, механизмы идентифицирующие данные такого класса, позволяют оператору сети либо полностью блокировать такой трафик, либо серьезно затрудняют его.

Результаты исследования, представленные в [126], показывают, что в интернете растет доля P2P трафика. Следовательно, решение задач анализа и классификации трафика является одним из ключевых условий улучшения скорости передачи данных, что в свою очередь приводит к улучшению

качества обслуживания.

4.4 Метод управления трафиком с помощью CEP-машины

В данной части приводится способ применения регулярной CEP-машины на этапе проектирования системы анализа, оптимизации и управления трафиком в узле сети. В частности, таким узлом сети может быть любой экземпляр виртуальной сетевой функции (VNF). Например, виртуальный сетевой коммутатор см. пример имплементации сети в части 4.5.

Общий вид системы управления трафиком. В общем случае система управления трафиком может состоять из таких компонентов:

1. System Traffic Monitor Plane: отслеживает входящий трафик на предмет важных изменений (событий) в его содержимом, объем и другие возможные аномалии.

2. System Control Plane: компонент принятия решений. Основывается на событиях обнаруженных System Traffic Monitor Plane модулем, устанавливает методы мониторинга, а также управления трафиком.

3. System Traffic Management Plane: управляет трафиком согласно правилам определенным в System Control Plane [3].

Схема взаимодействия компонентов системы представлена на рисунке 4.2. На приведенной схеме X^{ω} обозначает входящий в узел сети трафик, тогда как R^{ω} – выходящий соответственно. В начале входящий трафик X^{ω} анализируется System Traffic Monitor Plane компонентом на предмет возможных событий либо аномалий. Затем System Traffic Monitor Plane модуль перенаправляет данный трафик в System Traffic Management Plane где с помощью заданных System Control Plane модулем правил он оптимизируется и формируется выходящий трафик R^{ω}. Если компонент System Traffic Monitor Plane обнаруживает некоторое событие или аномалию во входящем трафике X^{ω}, то сообщение об этом $y \in Y$ отправляется модулю System

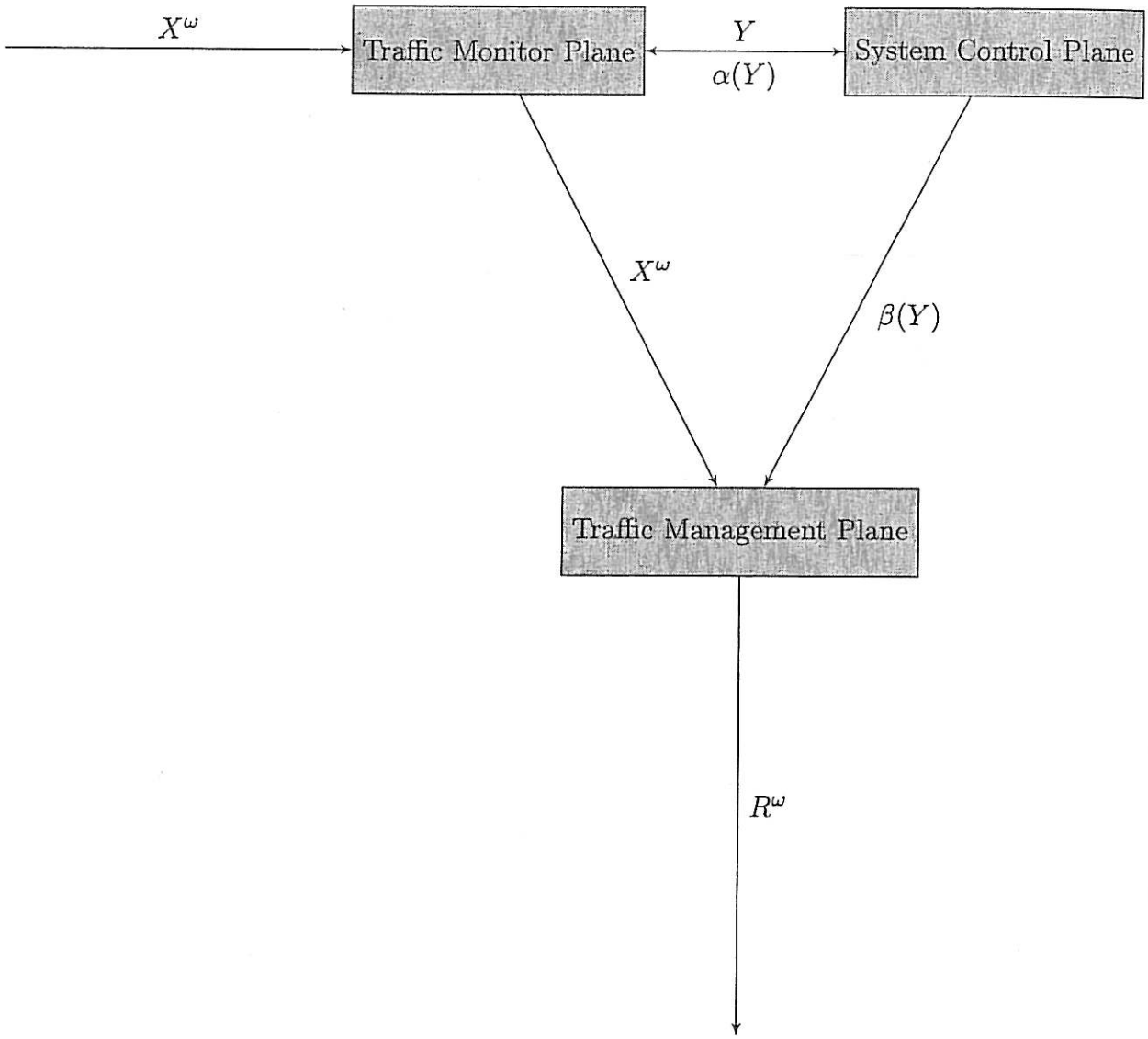

Рис. 4.2. Схема взаимодействия компонентов системы управления трафиком

Control Plane. Далее, System Control Plane модуль обрабатывает возникшее событие и отправляет управляющие сигналы $\alpha(y)$ и $\beta(y)$, $y \in Y$ компонентам System Traffic Monitor Plane и System Traffic Management Plane соответственно. Управление может заключаться в изменении метода мониторинга входящего трафика X^{ω}, либо изменении метода оптимизации и формирования выходящего трафика R^{ω}. Например, может быть запущен механизм масштабирования, увеличивающий объем доступных виртуальных вычислительных ресурсов, либо активизирующий дополнительные экземпляры виртуальных сетевых функций см. часть 4.5.

Замечание 4.2. Заметим что на физическом уровне входящий трафик X^{ω} может приходить по нескольким каналам связи. Аналогичное утверждение также верно для выходящего трафика R^{ω}. Более того, компонент System Traffic Management Plane должен управлять распределением выходящего трафика по доступным выходным каналам.

Взаимосвязь системы управления трафиком и регулярной CEP-машины. Напомним что регулярная CEP-машина определяется пятеркой $M = (X, Y, H, h_0, \alpha)$ см. определение 3.7. В данном контексте множество X является входным алфавитом пакетов, тогда как множество Y – выходной алфавит событий генерируемых как реакция на обнаруженные события или аномалии во входном трафике. Тогда любой регулярный обработчик (частичное отображение) $h \in H$ такой что $h\colon X^{+} \dashrightarrow Y$ можно интерпретировать как алгоритм анализа входного трафика предназначенный для обнаружения определенного множества аномалий. А функция отклика $\alpha\colon Y \to H$ определяет новый алгоритм анализа входного трафика, который должен использоваться в случае обнаружения той или иной аномалии на предыдущем шаге. Например, при обнаружении во входном трафике множества подозрительных пакетов которые могут содержать некую вредоносную информацию, может быть активирован алгоритм определения

генератора такой информации и его блокирования. Другим примером может являться обнаружение последовательности пакетов содержащих информацию о статусе других узлов сети на основании которой данный узел может определить загруженность всей сети и активировать соответствующий метод формирования пользовательского трафика. Таким образом, можно оптимизировать управление балансом загруженности всей сети в автоматическом режиме.

Замечание 4.3. Заметим, что как правило, каждый отдельный пакет $x \in X$ несет в себе слишком мало информации для эффективной идентификации аномальных изменений характера трафика. Следовательно, необходимо рассматривать комбинации пакетов для получения объективной картины происходящего в сети.

Использование машинного обучения при проектировании компонента анализа входного трафика. Решение задачи анализа входящего трафика является одним из ключевых пунктов при проектировании системы управления трафиком. Для решения данной задачи можно использовать метод рассмотренный в части 3.6. Для этого необходимо определить достаточно большое множество примеров событий E – некий набор хорошо известных проблем трафика которые могут привести к сбою или перегрузке сети. А также необходимо определить множество контрпримеров C – набор сценариев правильной работы сети.

Замечание 4.4. Заметим что множество контрпримеров должно быть максимальным для уменьшения проблемы ложного обнаружения аномалий в трафике.

Для иллюстрации метода приведем пример определения множеств E и C.

Пример 4.1. Рассмотрим телекоммуникационную сеть состоящую из n узлов (базовых станций), возможно виртуальных. И пусть множество па-

кетов характеризуется следующим образом:

$$X = \bigcup_{i=1}^{n} \{A^i \bigcup V^i \bigcup D^i \bigcup M^i\}$$

Где A^i, V^i, D^i множества аудио, видео и других веб-данных сгенерированных базовыми станциями соответственно. Тогда как M^i – множество служебных пакетов, которые могут содержать диагностическую информацию о состоянии станции, уровне загруженности, планируемых действиях (выключение или перезапуск системы, появление в сети нового узла и т.д.). Тогда начальное множество отслеживаемых событий E в общем случае можно задать следующим образом:

$$E = \{(a^* v^* d^*)^* m^+ (a^* v^* d^*)^* | \forall\, a \in A^i, v \in V^i, d \in D^i, m \in M^i\}$$

Т.е. системе необходимо отслеживать все слова содержащие в себе служебные пакеты и реагировать на те или иные события. Тогда множество контрпримеров можно задать как:

$$C = \{(a^* v^* d^*)^* | \forall\, a \in A^i, v \in V^i, d \in D^i\}$$

Где $i = 1 \ldots n$, * – звездочка Клини, а $x^+ = x^* \backslash \{\varepsilon\}$. Т.е. система не меняет режим работы при прохождении пользовательского трафика.

Другим примером может быть обнаружение ситуации когда возникает необходимость ограничения трафика генерируемого пользователями имеющими более низкий приоритет обслуживания. А также улучшение пропускной способности сети для видео трафика.

Пример 4.2. Пусть пользователь k имеет более высокий приоритет передачи данных. И также необходимо обеспечить высокую пропускную способность для видео трафика. Тогда система должна реагировать на ситуации когда входной трафик содержит слишком много пакетов с более низким приоритетом. Следовательно, множество примеров E можно задать следующим образом:

$E = \{x_i^+((v^k)^*(a^k)^*(d^k)^*)^+x_j^+ | \forall\, x_i, x_j \in X', v^k \in V^k, a^k \in A^k, d^k \in D^k\} \bigcup$
$\bigcup \{x_l^+ v^+ x_m^+ | \forall\, x_l, x_m \in X'', v \in V\}$
Где $X' = X \backslash (A^k \bigcup V^k \bigcup D^k)$ и $X'' = X \backslash V$. Следовательно, система не должна реагировать на ситуации когда в трафике отсутствуют пакеты узла k. Т.е. узел k не генерирует трафик. Либо в сети преобладает видео трафик или трафик генерируемый узлом k. Тогда множество контрпримеров C можно определить следующим образом:

$$C = \{x^* | \forall\, x \in X'\} \bigcup \{(x^k)^* | \forall\, x^k \in X^k\} \bigcup \{v^* | \forall\, v \in V\}$$

Где $X^k = A^k \bigcup V^k \bigcup D^k$.

Заметим что в случае ложного обнаружения, обнаруженное событие можно добавить во множество контрпримеров C для исключения ситуации обнаружения такого события в будущем.

4.5 Пример имплементации

Теперь приведем пример имплементации телекоммуникационной сети, использующей представленную выше систему управления трафиком.

Облачные коммуникации. В настоящее время все больше компаний-операторов телекоммуникационных сетей используют облачные технологии, перенося всю инфраструктуру в облако. Что приводит к понятию "облачные коммуникации".

Определение 4.4. Облачные коммуникации – это технология передачи голоса и других данных через интернет, базирующаяся на том что телекоммуникационные приложения, модули коммутации и сохранения информации размещены в облаке, и доступны через интернет в любой точке мира. А само облако управляется и организуется компанией-владельцем облака .

Более широким термином является понятие облачных сервисов, относящееся в основном к сервисам хранения данных в дата-центрах, досту-

пных через интернет. Не так давно, такие сервисы были ориентированы на данные, но с развитием VoIP (протокол передачи голоса через интернет), голос стал частью облачных технологий [127]. Компании-поставщики облачных коммуникаций предоставляют услуги и приложения передачи голоса и данных, размещая их на своих серверах, которые сами поддерживают и обслуживают, тогда как клиентам предоставляется доступ к "облаку". Оплачивая только услуги либо использование приложений, клиенты имеют более выгодное, надежное и защищенное коммуникационное окружение, избавляя себя от затрат на прокладку сетей и обслуживание сетевого оборудования [128]. Таким образом, используя услуги облачной коммуникации, компании могут сократить расходы без ущерба качеству предоставляемых услуг [129]. Успех Google и других компаний-поставщиков облачных услуг показал, что платформа основанная на облачных технологиях может быть на столько же эффективна, на сколько может быть эффективна и любая другая программная платформа, но гараздо менее затратна. Голосовые услуги предоставляемые "облаком"повышают ценность hosted telephony, поскольку клиенты могут равноценно использовать облачные решения и не зависить от производительности своего сетевого оборудования. Также такой подход расширяет географию использования, поскольку не зависит от местных операторов связи. Облачные коммуникации являются привлекательной технологией, поскольку "облако"сейчас может быть платформой для голосовых, видео и других данных. В настоящее время существуют три основные тенденции в корпоративной коммуникации принуждающие пользователей использовать облачные технологии и позволяющие им получать доступ к облаку используя множество устройств.

— Первой тенденцией является увеличение распределенной деятельности в компаниях имеющих множество филиалов либо позволяющих работать удаленно, что делает использование корпоративных сетей громоздким, не эффективным и дорогим.

— Второе, все большему количеству коммуникационных устройств требуется доступ к корпоративным сетям.

— Третье, корпоративные дата-центры размещаются в одном месте и управляются удаленно [130].

Теперь рассмотрим наиболее распространенный способ имплементации сети в облаке.

Виртуализация сетевых функций.

Определение 4.5. Виртуализация сетевых функций (Network functions virtualization, NFV) – это концепция сетевой архитектуры, которая использует технологию виртуализации для программного моделирования всех классов функций сетевого узла, создавая составные модули, которые можно объединять в последовательности с целью создания коммуникационных сервисов.

Виртуализация сетевых функций основывается на уже ставших традиционными способах серверной виртуализации, используемых в корпоративных информационных технологиях. Виртуальная сетевая функция (virtualized network function, VNF) может состоять из одной или нескольких виртуальных машин управляемых различным программным обеспечением и выполняющих различные процессы. Вместо использования специального сетевого оборудования предназначенного для каждой определенной сетевой функции, виртуальная сетевая функция может выполняться на стандартных высокопроизводительных серверах, коммутаторах и устройствах хранения данных, или даже на инфраструктуре облачных вычислений. На пример, виртуальный пограничный контроллер сессий (Session border controller, SBC) [131] может быть развернут для защиты сети без затрат на доставку и установку аппаратных модулей. Другими примерами NFV могут быть виртуальные балансировщики сети, брандмауеры, устройства обнаружения проникновения и WAN ускорители [132, 57].

В октябре 2012 года исследовательская группа "Network Functions Virtualisa представила спецификацию (белую книгу) по теме "software-defined networking (SDN) и OpenFlow "[134]. В состав группы разработчиков, подразделение European Telecommunications Standards Institute (ETSI), входили представители как европейской телекоммуникационной отрасли так и представители из других стран [135, 136]. Позднее группа дополнила спецификацию детальными материалами содержащими определения стандартной терминологии [137], а также примеры использования виртуализации сетевых функций, которые могут использоваться производителями и операторами как справочные материалы при разработке сетей использующих виртуализацию сетевых функций.

Парадигма виртуализации сетевых функций состоит из трех основных компонентов: [59]

1. Виртуальные сетевые функции (VNF) – программные реализации сетевых функций которые могут быть развернуты на виртуальной сетевой инфраструктуре (network functions virtualization infrastructure, NFVI).

2. Виртуальная сетевая инфраструктура (NFVI) – совокупность всех программных и аппаратных компонентов формирующих среду где размещаются виртуальные сетевые функции. Инфраструктура виртуализации сетевых функций может располагаться в нескольких местах. Сетевое соединение между различными компонентами NFV также рассматривается как часть инфраструктуры виртуализации сетевых функций.

3. Архитектурная парадигма управления и распределения виртуализации сетевых функций (Network functions virtualization management and orchestration architectural framework, NFV-MANO) – это набор всех функциональных блоков, репозиториев данных используемых этими блоками, а также множества ссылок и интерфейсов через которые данные функциональные блоки обмениваются информацией с целью управления и распределения инфраструктурой виртуализации сетевых функций и виртуаль-

ными сетевыми функциями.

Составным блоком для инфраструктуры NFV и NFV-MANO является платформа виртуализации сетевых функций. В случае инфраструктуры NFV, платформа состоит из виртуальной и физической обработки, а также ресурсов хранения данных и программного обеспечения для виртуализации. А в случае NFV-MANO, платформа состоит из менеджеров виртуальных сетевых функций и инфраструктуры NFV, а также программного обеспечения виртуализации выполняющегося на аппаратном контроллере. Платформа виртуализации сетевых функций реализует функции операторского класса, использующиеся для управления и мониторинга компонентов платформы, восстановления после сбоев, а также предоставления эффективной защиты. Все эти функции необходимы в публичной телекоммуникационной сети (public carrier network).

Поставщик услуг, следующий NFV дизайну, реализует одну или более виртуальных сетевых функций. Виртуальная сетевая функция сама по себе автоматически не предоставляет полезный продукт либо услугу доступную клиентам. Для построения более сложных сервисов, используется понятие цепочек сервисов, в которых последовательно вызываются несколько виртуальных сетевых функций для предоставления услуги. Другой аспект реализации виртуализации сетевых функций – это процесс управления (an orchestration process). Для построения высоконадежных и масштабируемых сервисов, NFV требует чтобы сеть могла создавать экземпляры виртуальных сетевых функций, проводить их мониторинг, восстанавливать, а также (самое главное для поставщиков услуг) взымать плату за предоставленные услуги. Также важно, чтобы уровень управления (orchestration layer) был способен управлять виртуальными сетевыми функциями внезависимости от лежащей в основе технологии используемой в VNF. Например, уровень управления должен быть способен управлять виртуальной сетевой функцией реализующей SBC, разработанной компанией X и выполняющейся

на VMware vSphere, также успешно, как и виртуальной сетевой функцией реализующей IMS, разработанной компанией Y, выполняющейся на KVM.

В процессе проектирования и разработки программного обеспечения предоставляющего виртуальные сетевые функции, разработчики могут структурировать это программное обеспечение в программные компоненты (с точки зрения реализации программной архитектуры), а затем упаковывать эти компоненты в один или несколько образов (с точки зрения архитектуры развертывания программного обеспечения). Эти определенные разработчиками программные компоненты называются VNF компонентами (VNFC). Виртуальные сетевые функции реализованы с помощью одного или нескольких VNF компонентов и предполагается, без потери общности, что VNFC экземпляры соответствуют VM образам как один к одному.

VNF компоненты в общем должны масштабироваться как путем увеличения производительности одного компонента виртуальной функции, так и путем добавления дополнительных экземпляров данного компонента функции для паралельного выполнения. Гибко выделяя вычислительные ресурсы (возможно виртуальные) каждому экземпляру VNFC, уровень управления сетью может увеличивать производительность данного VNF компонента для предоставления ожидаемой производительности в рамках одной функции или платформы. Аналогично, уровень управления сетью может увеличить производительность VNFC путем активации нескольких экземпляров данного VNF компонента на нескольких платформах, и следовательно, достичь необходимой результативности, определенной в архитектурной спецификации, при этом не влияя на стабильность работы других виртуальных функций.

Определяемые программно сети. Сети определяемые программно (softwar defined networking, SDN) – концепция связанная с виртуализацией сетевых функций, но относящаяся к другой предметной области. По существу, про-

граммное определение сети – это подход к созданию передающего данные сетевого оборудования и программного обеспечения, который разделяет и обобщает элементы этих систем. Разделение и обобщение элементов достигается путем отделения уровня управления (control plane) от уровня данных (data plane), так, чтобы уровень управления оставался централизованным, а компоненты передачи данных были распределены. Уровень управления взаимодействует с другими уровнями в двух направлениях:

— в первом случае, уровень управления предоставляет общий абстрактный вид сети программному обеспечению более высокого уровня использующему его прикладной программный интерфейс (application programming interface, API);

— во втором случае, уровень управления программирует поведение передающего уровня, используя прикладной программный интерфейс на уровне распределенного физического сетевого оборудования формирующего физическую сеть.

Следовательно, виртуализация сетевых функций не зависит от программно определяемой сети либо от ее концепции. Существует возможность полноценной реализации виртуальной сетевой функции (VNF), как полностью независимой сущности, используя существующие парадигмы сетевого и программного распределения. Однако, существуют неоспоримые преимущества использования концепции SDN при реализации и управлении инфраструктуры NFV, вчастности при рассмотрении способов управления и распределения виртуальных сетевых функций. Инфраструктура NFV требует централизованной системы управления и распределения ресурсов, которая принимает запросы от оператора ассоциированного с VNF. Далее, на основании полученных запросов, система управления выполняет соответствующие вычисления, конфигурирует хранилище данных и сетевые параметры необходимые для запуска соответствующей виртуальной сетевой функции. После запуска данной виртуальной сетевой функции, необходи-

мо вести мониторинг объема потребляемых ею вычислительных ресурсов, а также ее настройка в случае необходимости [138]. Вся такая функциональность может быть реализована с помощью использования концепции программно определяемой сети, и в таком случае, виртуализация сетевых функций может рассматриваться как один из главных примеров использования в сфере поставки телекоммуникационных услуг. Также очевидно, что множество примеров использования программно определяемой сети могут включать в себя концепции представленные в описании виртуализации сетевых функций. Примером может быть централизованный контроллер, управляющий распределенными функциями передачи данных, которые могли бы на самом деле быть также визуализированы на существующем оборудовании обработки и маршрутизации данных.

4.6 Эффективность метода

В данной части приводится сравнительный анализ переработанной и, широко используемой в настоящее время, оригинальной версии производительности виртуального сетевого комутатора Open Virtual Switch (OVS). Переработанная версия спроектирована с использованием математических моделей представленных в разделах 2 и 3. В частности, для улучшения качества передачи звука, а также изображения, в переработанную версию виртуального маршрутизатора была добавлена подсистема формирования трафика представленная в части 4.4. Для определения эффективности метода было проведено множество тестов, использующихся для оценки качества обслуживания (Quality of service).

Определение 4.6. Качество обслуживания (QoS) – это набор обобщенных показателей производительности сети. В частности, это производительность сети оцениваемая пользователями. Для количественного измерения качества обслуживания, обычно рассматривается несколько характе-

ристик, таких как: частота ошибок, скорость передачи данных, пропускная способность, задержки передачи, доступность, фазовое дрожание цифрового сигнала данных (jitter) и т.д.

В сфере компьютерных сетей, а также других телекоммуникационных сетей с коммутацией пакетов, качество обслуживания скорее относится к приоритезации трафика и механизмам контроля резервирования ресурсов, чем к достижению фиксированных показателей качества обслуживания. Качество обслуживания это возможность сети предоставлять различные приоритеты различным приложениям, пользователям, либо потокам данных, а также гарантирование заданной производительности тем или иным потокам данных. Также, качество обслуживания является важным, в частности, для передачи трафика с особыми требованиями.

Теперь приведем критерии оценки качества обслуживания в тестируемой сети. Пример критериев оценки качества можно найти в источнике [139]. Множество ситуаций может возникнуть при передаче пакетов данных по сети между отправителем и получателем, приводящих к следующим проблемам с точки зрения отправителя и получателя:

— Низкая пропускная способность (low throughput) По причине изменения загруженности сети, генерируемой различными пользователями, которые используют общие сетевые ресурсы, скорость передачи (максимальная пропускная способность), которая может быть предоставлена некоторому потоку данных, может быть слишком низкой для сервисов мультимедиа реального времени, если все потоки данных имеют одинаковый приоритет планирования.

Проведенные тесты показывают, что использование переработанного виртуального коммутатора уменьшает вероятность возникновение данной аномалии на 10%.

— Потеря пакетов (dropped packets). Маршрутизаторы (коммутаторы) мо-

гут иметь проблемы с доставкой (терять) некоторые пакеты в случае нарушения их целосности, либо пакеты могут приходить в тот момент когда внутренняя очередь маршрутизатора уже переполнена. В такой ситуации приложение-получатель может запросить повторную передачу данной информации, что в свою очередь может приводить к серьезным задержкам передачи данных в целом.

Согласно результатам тестирования, потеря пакетов сократилась на 15,4%.

— Задержка (latency). Доставка каждого пакета адресату может занимать много времени по причине долгого ожидания пересылки в длинных очередях, либо пакет пересылается по более длинному маршруту во избежание перегруженности сети. Данная аномалия отличается от проблемы пропускной способности, поскольку задержка может накапливаться во времени, даже если пропускная способность почти удовлетворяет требованиям. В некоторых случаях, чрезмерная задержка может привести к невозможности использования некоторых приложений, таких как передача голоса (VoIP). Результаты тестирования показывают уменьшение задержек при передаче на 6,9%.

— Фазовое дрожание цифрового сигнала данных (jitter). Пакеты доставляются от отправителя получателю с различной задержкой. Задержка пакетов варируется в зависимости от положения пакетов в очередях маршрутизаторов по пути между отправителем и получателем. Данное положение в очередях может изменяться случайным образом. Такая вариация задержек известна как фазовое дрожание цифрового сигнала данных и может серьезно влиять на качество передачи аудио и видео данных.

Результаты тестирования показывают уменьшение проявления данной аномалии на 5.3%.

— Нарушение порядка доставки пакетов (out-of-order delivery). Когда набор связанных между собой пакетов направляется по сети, различные пакеты могут идти по различным маршрутам, что приводит к различным

задержкам доставки. В результате пакеты приходят получателю в последовательности, отличной от той в которой они были отправлены. Эта проблема требует специальных дополнительных протоколов упорядочивания полученных пакетов для восстановления исходного порядка в котором они были отправлены. Что особенно важно в случае видео и VoIP потоков где качество сильно зависит от величины задержки и отсутствия упорядоченности данных.

Из результатов проведенного тестирования видно что проявление данной аномалии уменьшилось на 19,4%.

Выводы по разделу 4

В данном разделе были представлены примеры использования регулярных CEP-машин при проектировании систем оптимизации трафика в распределенных информационных системах.

В части 4.1 были представлены базовые обозначения использующиеся в данном разделе.

Часть 4.2 была посвящена основным методам управления сетями.

Далее, в части 4.3 рассматривались задачи формирования и оптимизации трафика. Были рассмотрены основные проблемы возникающие при решении таких задач.

Затем, в части 4.4 был приведен подход к решению задачи формирования и оптимизации трафика с помощью регулярных CEP-машин. Было показано что метод машинного обучения детально описанный в части 3.6 значительно упрощает процесс анализа возникающих в сетях аномалий. В частности, примеры 4.1 и 4.2 приводят способ раннего обнаружения возможности перегрузки сети, что помогает в ее автоматической балансировке и тем самым улучшает производительность, поскольку, значительно уменьшается вероятность потери пакетов и их повторной передачи. Что в свою

очередь является ключевым фактором при обеспечении качества передачи мультимедиа данных. Также пример 4.2 показывает возможность управления отдельными потоками данных имеющих более высокий приоритет.

Далее, часть 4.5 содержит пример имплементации современной телекоммуникационной сети. Показывается что сочетание использования систем оптимизации трафика основанных на регулярных CEP-машинах, а также "облачных"технологий может значительно улучшить ключевые показатели производительности сети. Поскольку, системы оптимизации трафика, основанные на регулярных CEP-машинах, позволяют эффективно обнаруживать возможные аномалии при передаче по сети трафика. А "облачные"технологии позволяют гибко реагировать на те или иные аномалии путем увеличения количества доступных вычислительных ресурсов на уже действующих узлах сети, либо активизации дополнительных узлов для расширения передающих возможностей. Также в части отмечается экономическая выгода применения "облачных"технологий.

И наконец, в части 4.6 представлены критерии оценки эффективности метода. Приведены результаты тестирования производительности сети по основным показателям качества обслуживания (QOS). Полученные результаты показывают хороший прирост производительности сети, в частности, при передаче мультимедиа потоков данных. Что позволяет утверждать, что предложенный метод является достаточно эффективным при проектировании систем оптимизации и формирования трафика.

ВЫВОДЫ

В работе решена важная научно-прикладная задача – совершенствование моделей и развитие методов статического анализа для событийно-управляемых систем обработки информации. Был проведен детальный обзор задач и приложений в которых термин "событие"играет ключевую роль. Было показано что в настоящее время существует множество моделей описывающих системы основанные на событиях. Анализ аномалий в таких моделях привел к формулировке задач решаемых в работе. В частности, была обозначена тенденция показывающая, что такие концепции как архитектура управляемая событиями и обработка сложных событий могут широко использоваться для решения прикладных задач непрерывной обработки данных в различных предметных областях. В ходе исследования было обнаружено, что такое направление как машинное обучение может быть использовано при проектировании и разработке систем основанных на обработке сложных событий. Также был предложен подход использования системы обработки событий при решении задач оптимизации и управления телекоммуникационными сетями.

Обобщение примеров систем обработки сложных событий привело к формулировке математической модели, которая была названа CEP-машиной (Complex Event Processing machine). Модель описана с двух точек зрения: структурной и поведенческой. Для описания поведенческой части модели, в работе был введен набор формальных понятий.

Теорема 2.2 доказывает принципиальный результат изучения свойств поведения CEP-машин. Он заключается в том, что либо обработчик событий способен распознать бесконечное количество сложных событий и в таком случае, обработка потоков событий невозможна, либо обработчик

способен обработать любой поток событий, и в таком случае, возможно обнаружение только конечного числа сложных событий.

Также в работе, впервые, был установлен очень важный результат, подтверждающий двойственность между CEP-машинами и предавтоматами (см. теорему 2.3 и предложение 2.2). Этот результат дает возможности изучать поведение CEP-машин.

В работе было показано, что важной задачей для приложений является задача синтеза CEP-машины с заданным поведением, которое задается функцией отклика. Теорема 2.4 описывает и дает основания для решения этой задачи.

В работе решен вопрос алгоритмической реализуемости CEP-машин. Теорема 2.7 дает решение задачи об алгоритмической реализуемости CEP-машины. Эта теорема доказывает, что условие вычислимости обработчиков событий CEP-машины является достаточным для ее алгоритмической реализуемости.

Затем, в работе рассматривались системы обработки сложных событий, у которых потоки событий можно описать регулярным языком. Впервые был предложен специальный класс обработчиков сложных событий. Такие обработчики событий были названы регулярными. В теореме 3.1 было доказано, что регулярный обработчик может быть реализован с помощью конечного автомата.

Основываясь на понятии регулярного обработчика, в работе представлен важный класс CEP-машин, названных регулярными. Приведен алгоритм описывающий поведение таких машин. Теорема 3.3 доказывает что регулярная CEP-машина может быть преобразована в детерминированый конечный автомат, который является глобализацией двойственного предавтомата этой регулярной CEP-машины.

Полученные результаты являются важными и полезными в моделировании и разработке систем обработки сложных событий, способных распо-

знавать такие события, которые можно описать регулярным языком.

Дальнейшее исследование полученых результатов привело к разработке метода машинного обучения для регулярных обработчиков с одним допускающим состоянием (регулярных акцепторов). Также была разработана и детально описана процедура верификации правильности метода. Было показано, что этот метод может быть использован при проектировании и разработке систем основанных на архитектуре управляемой событиями. Преимуществом метода является то, что анализ всех возможных комбинаций событий может быть заменен обучением на этапе выполнения. В рамках исследования разработан и проведен компьютерный эксперимент, основанный на процедуре верификации, который показал, что обучающий метод восстанавливает тестируюмую CEP-машину.

В работе представлены примеры использования регулярных CEP-машин при проектировании систем оптимизации трафика в сетях. Проведен детальный анализ задач формирования и оптимизации трафика. Рассмотрены основные проблемы возникающие при решении таких задач, что привело к формулировке концепции использования регулярных CEP-машин при проектировании таких систем.

Также работа содержит пример имплементации современной телекоммуникационной сети. Показано, что сочетание использования систем оптимизации трафика, основанных на регулярных CEP-машинах, а также "облачных"технологий может значительно улучшить ключевые показатели производительности сети. Поскольку, системы оптимизации трафика, основанные на регулярных CEP-машинах, позволяют эффективно обнаруживать возможные аномалии при передаче по сети трафика. А "облачные"технологии позволяют гибко реагировать на те или иные аномалии путем увеличения количества доступных вычислительных ресурсов на уже действующих узлах сети, либо активизации дополнительных узлов для расширения передающих возможностей.

В заключительной части работы представлены критерии оценки эффективности метода. Приведены результаты тестирования производительности сети по основным показателям качества обслуживания (QOS). Полученные результаты показывают хороший прирост производительности сети, в частности, при передаче мультимедиа потоков данных. Что позволяет утверждать, что предложенный метод является достаточно эффективным при проектировании систем оптимизации и формирования трафика.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Zholtkevych G., Novikov B., Dorozhinsky V. Pre-automata and complex event processing // ICTERI 2014. CCIS / Ed. by V. Ermolayev, et al.— Switzerland: Springer International Publishing, 2014. — Vol. 469. — Pp. 100-116.
2. Dorozhinsky V. Regular complex event processing machines // Information processing systems. — 2015. — no. 8. — Pp. 82-86.
3. Дорожинский В., Жолткевич Г. Регулярная обработка событий в управлении телекоммуникационными сетями // Системи управління, навігації та зв'язку. — 2015. — Т. 35, № 3. — С. 70-76.
4. Dorozhinsky V., Khadikov A., Zholtkevych N. Machine learning methods and "real-time" economics // ICTERI 2016.— CEUR-WS.org, 2016.— Vol. 1614. — Pp. 469-476.
5. Zholtkevych G., Dorozhinsky V., Khadikov A. Real-time event processing and machine learning methods // Weapons systems and military equipment. — 2016. — no. 2. — Pp. 79-83.
6. Zholtkevych G., Dorozhinsky V., Khadikov A. Regular event processing and machine learning correctness verification // Information processing systems. — 2016. — Vol. 146, no. 9. — Pp. 162-166.
7. **Federal standard 1037c data stream. http://www.its.bldrdoc.gov/fs-1037/dir-010/_1451.htm.**
8. **Srfi 41: Streams.— srfi.schemers.org. http://srfi.schemers.org/srfi-41/srfi-41.html.**

9. Gehani N. H., Jagadish H. V., Shmueli O. Composit event specification in active databases: Model & implementation // Proceedings of the 18th International Conference on Very Large Data Bases (VLDB'92).— Vancouver, Canada: Morgan Kaufmann, 1992. — Pp. 327-338.
10. Ceri S., Pelagatti G. Distributed Databases - Principles and Systems.— New York, San Francisco, Washington, D.C.: McGraw-Hill Inc., 1984.
11. Codd E. F. A relational model of data for large shared data banks // CACM.— 1970. —Vol. 13, no. 6.
12. Souleiman H., Riain S., Curry E. Approximate semantic matching of heterogeneous events // In 6th ACM International Conference on Distributed Event-Based Systems (DEBS 2012).— Berlin, Germany: ACM, 2012.— Pp. 252-263.
13. Hanson J. Event-driven services in SOA // Javaworld. — 2005. — no. 1.
14. Mani-Chandy K. Event-Driven Applications: Costs, Benefits and Design Approaches. — USA: California Institute of Technology, 2006.
15. Sliwa C. Event-driven architecture poised for wide adoption // Computerworld. — 2003. — no. 5.
16. Mersel J. Program interrupt on the univac scientific computer // Proceedings of the Western Joint Computer Conference (WJCC'56).— San Francisco, CA, USA: ACM, 1956. — Pp. 52-53.
17. Rosen S. Electronic computers: A historical survey // ACM Computing Surveys. — 1969. — Vol. 1, no. 1. — Pp. 7-36.
18. **Red hat enterprise realtime reference guide.** — www.redhat.com. **https: //access.redhat.com/documentation/en-US/Red_Hat_Enterprise_ MRG/1.3/html/Realtime_Reference_Guide/chap-Realtime_ Reference_Guide-Hardware_interrupts.html#cp-main.**

19. Corbet J., Rubini A., Kroah-Hartman G. Linux Device Drivers, Third Edition. — O'Reilly Media, 2005. — 269 pp.
20. Krasner G. E., Pope S. T. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80 // Journal of Object Oriented Programming. — 1988. — Vol. 1.3. — Pp. 26-49.
21. Linton M. A., Vlissides J. M., Calder P. R. Composing user interfaces with interviews // Tech. rep. CSL-TR-88-369. — Stanford, CA, USA: Stanford University, 1988.
22. Dogac A. METU object-oriented DBMS / A. Dogac, C. Ozkan, B. Arpinar et al. // Advances in Object-Oriented Database Systems. LNCS / Ed. by A. Dogac, M. T. Ozsu, A. Biliris, T. Sellis. — Berlin, Heidelberg: SpringerVerlag, 1995. — Vol. 978. — Pp. 14-27.
23. Dittrich K. R., Gatziu S., Geppert A. The active database management system manifesto: A rulebase of ADBMS features // In: Proceedings of the 2nd International Workshop on Rules in Database Systems (RIDS '95). Lecture Notes in Computer Science. / Ed. by T. Sellis. — Glyfada, Athens, Greece: Springer, 1995. — Vol. 985. — Pp. 3-20.
24. McCarthy D., Dayal U. The architecture of an active database management system / Ed. by J. Clifford, B. Lindsay, D. Maier. — Portland, OR, USA: ACM Press, 1989. — Pp. 215-224.
25. Paton N. W., Diaz O. Active database systems // ACM Computing Surveys. — 1999. — Vol. 31, no. 1. — Pp. 63-103.
26. Cole R., Graefe G. Optimization of dynamic query evaluation plans // Management of Data. — Minneapolis, MI: In: Proc. ACM SIGMOD Conf., 1994. — Pp. 150-160.

27. Ives Z. An Adaptive Query Execution System for data integration / Z. Ives, D. Florescu, M. Friedman et al. // Proc. ACM SIGMOD Int. Conf. on Management of Data. — ACM, 1999. — Pp. 299-310.
28. Eugster P. T. The many faces of publish/subscribe / P. T. Eugster, P. A. Felber, R. Guerraoui, A. M. Kermarrec // ACM Computing Surveys. — 2003. — Vol. 35, no. 2. — Pp. 114-131.
29. Muhl G. Large-scale content-based publish/subscribe systems // Ph.D. Thesis. — Darmstadt, Germany: Technische Universitat Darmstadt, 2002.
30. Oki B. The information bus: An architecture for extensible distributed systems / B. Oki, M. Pfluegl, A. Siegl, D. Skeen // Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP'93) / Ed. by A. P. Black, B. Liskov. — Asheville, NC, USA: ACM Press, 1993. — Pp. 58-68.
31. Parnas D. L. On the criteria to be used in decomposing systems into modules // Communications of the ACM.— ACM, 1972.— Vol. 15.— Pp. 1053-1058.
32. Booch G. Object-Oriented Analysis and Design with Applications. 2nd. — Redwood City, CA, USA: Benjamin Cummings Publishing, 1994.
33. Szyperski C. A. Component Software: Beyond Object-Oriented Programming. — Boston, MA, USA: Addison-Wesley, 1998.
34. Luckham D. C. Event Processing for Business: Organizing the Real-Time Enterprise. — Hoboken New, Jersey: John Wiley & Sons Inc., 2012.
35. Bates J. Secrets revealed: Trading tools uncover hidden opportunities // Global Trading.— 2012. **http://fixglobal.com/home/secrets-revealed-trading-tools-uncover-hidden-opportunities/.**
36. Martin-Flatin J. P., Jakobson G., Lewis L. Event correlation in integrated management: Lessons learned and outlook // Network and Syst. Management. — 2007. — Vol. 17, no. 4.

37. Schmerken I. Deciphering the myths around complex event processing // Wall Street & Technology. — 2008.

38. Cugola G., Margara A. Processing flows of information: From data stream to complex event processing // CSUR, ACM Press. — 2012.— Vol. 44, no. 3.

39. Babcock B., Datar M., Motwani R. Load shedding for aggregation queries over data streams //In ICDE'04: Proc. 20-th Int. Conf. Data Eng.— Washington: IEEE CS, 2004. — Pp. 350-361.

40. Mit/brown/brandeis "aurora" stream processing project. — cs.brown.edu. **http://cs.brown.edu/research/aurora/.**

41. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. — Addison-Wesley, 2002.

42. Odysseus - an open source framework for event processing engines based on java. — Germany: uni-oldenburg.de. **http://odysseus.informatik.uni-oldenburg.de/index.php?id=1&L=2.**

43. Adi A. Complex Event Processing for financial services / A. Adi, D. Botzer, G. Nechushtai, G. Sharo //In Proceedings of the IEEE Services Computing Workshops. — IEEE, 2006. — Vol. SCW'06, 0-7695-2681-0/06.

44. Widder A. An approach for automatic fraud detection in the insurance domain / A. Widder, R. Ammon, G. Hagemann, D. Schonfeld //In AAAI Symposia, Spring, SS 09-05-017. — AAAI, 2009.

45. Wang D., Rundensteiner E. A. Active complex event processing over event streams // Proceedings of the VLDB Endowment. — Seattle, Washington: Ellison III, Richard T., 2011.

46. **Crosman P., Ravenpack A. To feed news into trading algos // Wall Street & Technology.— 2009. http://www.wallstreetandtech.com/ data-management/aleri-ravenpack-to-feed-news-into-tradin/ 217500395.**

47. Eckert M. A CEP Babelfish: languages for complex event processing and querying surveyed / M. Eckert, F. Bry, S. Brodt et al. // Reasoning in Event-Based Distributed Systems, SCI / Ed. by S. Helmer, et al. — Berlin, Heidelberg: Springer-Verlag, 2011. —Vol. 347. — Pp. 47-70.
48. Stanley H. E. Interview on econophysics // "IIM Kozhikode Society & Management Review".— 2013.— Vol. 2, no. 2.— Pp. 73-78. **http:// www.saha.ac.in/cmp/camcs/Stanley-interview.pdf**.
49. Etzion O., Niblett P. Event Processing in Action. — Manning Publications, 2010.
50. Belzer J., Holzman A. G., Kent A. Encyclopedia of Computer Science and Technology. — USA: CRC Press, 1975. — Vol. 25.
51. Koshy T. Discrete Mathematics With Applications.— Academic Press, 2004.
52. Dokuchaev M., Novikov B., Zholtkevych G. Partial actions and automata // Alg. Discr. Math. — 2011. — Vol. 2, no. 11.
53. Zholtkevych G. Realisation of synchronous and asynchronous black boxes using machines // ICTERI 2015. CCIS / Ed. by V. Yakovyna, et al. — Switzerland: Springer International Publishing, 2016.— Vol. 594.— Pp. 124-139.
54. Kohavi R., Provost F. Glossary of terms // Machine Learning. — 1998. — Vol. 30. — Pp. 271-274.
55. Mannila H. Data mining: machine learning, statistics, and databases // Int'l Conf. Scientific and Statistical Database Management. — USA: IEEE Computer Society, 1996.
56. Friedman J. Data mining and statistics: What's the connection? // Computing Science and Statistics. — 1998. — Vol. 21, no. 29. — Pp. 3-9.
57. **Network functions virtualisation (NFV); use cases.— ISG web portal. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01. 01_60/gs_NFV001v010101p.pdf.**

58. Lemke A. 5 must-have attributes of an NFV platform.— insight.nokia.com, 2014. **http://insight.nokia.com/ 5-must-have-attributes-nfv-platform#main-content.**

59. Network-functions virtualization (NFV) proofs of concept // Framework, GS NFV-PER 002 v1.1.1.— ISG web portal, 2013. **http://www.etsi. org/technologies-clusters/technologies/nfv.**

60. **Wilson C. New NFV forum focused on interoperability // Light Reading. — 2015. http://www.lightreading.com/nfv/nfv-strategies/ new-nfv-forum-focused-on-interoperability/d/d-id/710874.**

61. Open virtual switch project.— www.openvswitch.org. **http://www. openvswitch.org/.**

62. Michelson B. Event-driven architecture overview // Patricia Seybold Group. — 2006.

63. Novikov B., Perepelytsya I., Zholtkevych G. Pre-automata as mathematical models of event flows recognisers // Proc. 7th Int. Conf. ICTERI 2011 / Ed. by V. Ermolayev, et al. — CEURS-WS, 2011. — Pp. 41-50.

64. Perepelytsya I., Zholtkevych G. On some class of mathematical models for static analysis of critical-mission asynchronous systems // Syst. ozbr. ta viysk. tehn. — 2011. — Vol. 27, no. 3. — Pp. 60-63.

65. Perepelytsya I., Zholtkevych G. Hierarchic decomposition of pre-machines as models of software system components // Syst. upravl. navig. i zv'i- azku. — 2011. — Vol. 20, no. 4. — Pp. 233-238.

66. Mikhailova I., Novikov B., Zholtkevych G. Protoautomata as models of systems with data accumulation // Proc. 9th Int. Conf. ICTERI 2013 / Ed. by V. Ermolayev, et al. — CEUR-WS, 2013. — Pp. 582-589.

67. Willard S. General Topology. — Mineola, NY, USA: Dover Publications, 2004.

68. Munkres J. Topology. — 2 edition. — Upper Saddle River, NJ: Prentice Hall, 2000.
69. Kelley J. L. Convergence in topology // Duke Mathematical Journal. — 1950. — Vol. 17, no. 3. — Pp. 277-283.
70. Kelley J. L. The tychonoff product theorem implies the axiom of choice // Fundamenta Mathematica. — 1950. — Vol. 37. — Pp. 75-76.
71. Chernoff P. N. A simple proof of tychonoff's theorem via nets // American Mathematical Monthly. — 1992. —Vol. 99, no. 10. — Pp. 932-934.
72. Johnstone P. T. Tychonoff's theorem without the axiom of choice // Fundamenta Mathematica. — 1981. — Vol. 113. — Pp. 21-35.
73. Eilenberg S. Automata, Languages, and Machines. — New York and London: Academic Press, 1974. — Vol. A.
74. Cutland N. Computability. An introduction to recursive function theory. — London: Cambridge Univ. Press, 1980.
75. Aho A. V., Sethi R., Ullman J. D. Compilers: Principles, Techniques, and Tools (1st ed.). —Addison-Wesley ISBN 9780201100884, 1986.
76. Anderson J. A., Head T. J. Automata theory with modern applications. — Cambridge University Press. ISBN 978-0-521-84887-9, 2006.
77. Black P. Finite state machine // Dictionary of Algorithms and Data Structures. — USA: U.S. National Institute of Standards and Technology, 2008.
78. Pouly M., Kohlas J. Generic Inference: A Unifying Theory for Automated Reasoning. — John Wiley & Sons, 2011.
79. Storer J. A. An Introduction to Data Structures and Algorithms. — Springer Science & Business Media, 2001. — 337 pp.
80. Hopcroft J., Motwani R., Ullman J. Introduction to Automata Theory, Languages, and Computation, 2nd ed. — Addison-Wesley, 2001.

81. Megrelishvili M., Schroder L. Globalization of confluent partial actions on topological and metric spaces // Topology and its Appl. — 2004. — Vol. 145. — Pp. 119-145.
82. Hopcroft J. An *n*log*n* algorithm for minimizing states in a finite automaton // In: Proc. Internat. Sympos., Technion, Haifa. Theory of machines and computations. — New York: Academic Press, 1971. — Pp. 189-196.
83. Knuutila T. Re-describing an algorithm by hopcroft // Theoretical Computer Science. — 2001. — Vol. 250, no. 1-2. — Pp. 333-363.
84. Scientific computing tools for python. — scipy.org. **http://scipy.org/.**
85. Traffic shaping // ATM Forum Traffic Management Specification, Version 4.0. — ATM. — Vol. Approved Specification 0056.00. **https://www.broadband-forum.org/ftp/pub/approved-specs/ af-tm-0056.000.pdf.**
86. Арипов М. Н. Проектирование и техническая эксплуатация сетей передачи дискретных сообщений. — М. : Радио и связь, 1988. — 285 с.
87. Лосев Ю. И., Шматков С. И., Руккас К. М. Методы и модели обмена информацией в распределенных адаптивных вычислительных сетях с временной параметризацией параллельных процессов.— Х.: ХНУ имени В. Н. Каразина, 2011. — 204 с.
88. Абдельхамид З., Звягольская Г. В., Селевко С. Н. Методы адаптации и управления в задачах профилирования трафика // Радиотехника: Всеукр. межведомств. науч.-техн. сб. — 2004. — Т. 138. — С. 130-133.
89. Назаров А. Н., Разживин И. А., Симонов М. В. АТМ: принципы и технические решения создания сетей. — М. : Горячая линия - Телеком, 2002. — 408 с.
90. Абдельхамид З., Звягольская Г. В., Селевко С. Н. Разработка математической модели состояния нестационарной телекоммуникационной

системы // Радиотехника: Всеукр. межведомств. науч.-техн. сб. — 2005. — Т. 142. — С. 80-82.

91. Хемби С., Мак-Ферсон Д. Принципы маршрутизации в Internet. Пер. с англ. — 2 изд. — М. : Издательский дом «Вильямс», 2001. — 448 с.
92. Кучерявый Е. А. Управление трафиком и качество обслуживания в сети Интернет. — СПб. : Наука и техника, 2004. — 336 с.
93. Олифер В. Г., Олифер Н. А. Новые технологии и оборудование IP- сетей. — СПб. : BHV-СПб, 2000. — 512 с.
94. Остерлох Х. Маршрутизация в IP-сетях. Принципы, протоколы, настройка. — СПб. : BHV-СПб, 2002. — 512 с.
95. Internet Standard definition of shaper // IETF RFC 2475. An Architecture for Differentiated Services. — IETF. **https://tools.ietf.org/html/rfc2475#section-2.3.3.3.**
96. ITU-T, traffic control and congestion control in B-ISDN // Recommendation I.371, Annex A. — International Telecommunication Union, 2004. **http://www.itu.int/rec/T-REC-I.371-200403-I/en.**
97. Comparing traffic policing and traffic shaping for bandwidth limiting // Cisco Tech Notes.— Cisco Systems, 2005.— Vol. 19645. **http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html.**
98. Dischinger M. Detecting bittorrent blocking / M. Dischinger, A. Mislove, A. Haeberlen, K. Gummadi // IMC '08 Proceedings of the 8th ACM SIGCOMM conference on Internet measurement. — New York, NY, USA: ACM, 2008. — Pp. 3-8.
99. Zhang Z. Y., Mao M., Zhang M. Ascertaining the reality of network neutrality violation in backbone isps // HotNets. — USA: Association for Computing

Machinery, Inc., 2008. **https://www.microsoft.com/en-us/research/publication/ascertaining-the-reality-of-network-neutrality-violation-in-backb**

100. Pratt I., Fraser K. a user-accessible gigabit ethernet interface // Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies Proceedings. — Computer Laboratory, Cambridge University, 2001. — Vol. 1. — Pp. 67-76.
101. Review of the internet traffic management practices of internet service providers // Canadian telecomms regulator CRC ruling relating to traffic shaping usage by domestic ISPs. — Canada: Telecom. Reg. Policy CRTC 2009-657, 2009. **http://www.crtc.gc.ca/eng/archive/2009/ 2009-657.htm.**
102. **How to bypass internet censorship.— FLOSS Manuals. http://en.flossmanuals.net/bypassing-censorship/ch007_ chapter-2-censorship/.**
103. Helzer J., Xu L. Congestion control for multimedia streaming with self-limiting sources. — www.ieee-icnp.org. **http://www. ieee-icnp.org/ 2005/Posters/Helzer.pdf.**
104. Venkataramani A., Kokku R., Dahlin M. Tcp nice: Self-tuning network support for background applications.— citeseerx.ist.psu.edu. **http:// citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.5905.**
105. Turner J. New directions in communications (or which way to the information age?) // Communications Magazine, IEEE. — 1986. — Vol. 24, no. 10. — P. 815.
106. Tanenbaum A. S. Computer Networks, Fourth Edition.— 4 edition.— USA: Prentice Hall PTR, 2003.
107. Linux HTB home page. — luxik.cdi.cz. **http://luxik.cdi.cz/~devik/ qos/htb/.**
108. The user network interface (UNI) v3.1 // ATM Forum. — Prentice Hall PTR, 1995.
109. McDysan D. E., Spohn D. L. ATM: Theory and Application. on computer communications. — USA: McGraw-Hill, 1995.

110. ITU-T, B-ISDN ATM layer cell transfer performance // Recommendation I.356, Section 6.5.2.2 Cell delay variation between two MPs. — International Telecommunication Union, 2000.

111. **Intel server board s5400sf // Technical Product Specification.— Intel, 2007. http://download.intel.com/support/motherboards/server/ s5400sf/sb/s5400sf_tps_rev2_01.pdf.**

112. Floyd S., Jacobson V. Random early detection (RED) gateways for congestion avoidance // IEEE/ACM Transactions on Networking. — 1993.— Vol. 4, no. 1. — Pp. 397-413.

113. Clark D. D., Wroclawski J. An approach to service allocation in the internet // IETF. — tools.ietf.org, 1997.— P. 12. **https://tools.ietf.org/ html/draft-clark-diff-svc-alloc-00.**

114. Gettys J. RED in a different light. — gettys.wordpress.com, 2010. **https: //gettys.wordpress.com/2010/12/17/red-in-a-different-light/.**

115. Jacobson V., Nichols K., Poduri K. RED in a different light.— citeseerx.ist.psu.edu, 1999. **http: //citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.22.9406.**

116. Chao H. J. Quality of service control in high speed networks.— 605 Third Avenue, New York, NY: John Wiley & Sons Inc. doi:10.1002/0471224391.fmatter_indsub. ISBN: 0-471-00397-2, 2002.

117. Floyd S., Gummadi R., Shenker S. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management.— citeseer.ist.psu.edu, 2001. **http://citeseer.ist.psu.edu/ floyd01adaptive.html.**

118. Srikant R. The Mathematics of Internet Congestion Control. — Boston, MA, USA: Birkhuser, 2004.

119. ChangwangZ. RRED: robust RED algorithm to counter low-rate denial- of-service attacks / Z. Changwang, Y. Jianping, C. Zhiping, C. Weifeng // IEEE Communications Letters. — 2010. — Vol. 14, no. 5. — Pp. 489-491.

120. Ietf definition of classifier // IETF RFC 2475. An Architecture for Differentiated Services.— IETF. **https://tools.ietf.org/html/rfc2475# section-2.3.1.**

121. Suppliers' information note for the bt network bt wholesale // BT IP- stream Advanced Services - End User Speed Control and Downstream Quality of Service.— www1c.btwebworld.com, 2007. **http://www1c. btwebworld.com/sinet/450v1p2.pdf.**

122. Ferguson P., Huston G. Quality of Service: Delivering QoS on the Internet and in Corporate Networks. — John Wiley & Sons, Inc., 1998.

123. **Bittorrent protocol // wiki.theory.org. https://wiki.theory.org/ BitTorrentSpecification.**

124. Bujlow T., Carela-Espanol V., Barlet-Ros P. Independent comparison of popular dpi tools for traffic classification // Computer **Networks.— 2014. http://tomasz.bujlow.com/publications/2014_ journal_elsevier_comnet_independent_comparison.htm.**

125. Hjelmvik E., John W. Statistical protocol identification with spid: Preliminary results // Proceedings of SNCNW. — spid.sourceforge.net, 2009. **http://spid.sourceforge.net/sncnw09-hjelmvik_john-CR.pdf.**

126. Leydon J. P2p swamps broadband networks.— UK: www.theregister.co.uk, 2002. **http://www.theregister.co.uk/2002/ 09/12/p2p_swamps_broadband_networks/.**

127. Arnold J. Why businesses need to think differently about cloud communications // Focus. — 2010. **http://www.comparebusinessproducts. com/ briefs/why-businesses-need-think-differently-about-cloud.**

128. Whitemore J. Unified communications for the enterprise: hosted voip or cloud communications // Smoothstone IP Communications. — 2009.

129. Morisy M. Cloud-based unified communications could reduce costs and hassle // Search Unified Communications. — 2009.

130. Thornycroft P. Moving communications into the cloud // Phone+. — 2010. — no. 4.

131. HautakorpiJ. Requirements from SIP (session initiation protocol) / J. Hautakorpi, G. Camarillo, R. Penfield et al. // Session Border Control Deployments. — IETF. RFC 5853., 2010.

132. Munch B., Neil R. Magic quadrant for wan optimization // Gartner, Inc. — 2015.

133. **Network functions virtualisation. — ISG web portal. https: //portal.etsi.org/tb.aspx?tbid=789&SubTB=789,795,796,801, 800,798,799,797,802.**

134. Network functions virtualisation: Introductory white paper. — ETSI, 2012. **https://portal.etsi.org/NFV/NFV_White_Paper.pdf**.

135. Latest agenda at SDN // OpenFlow World Congress. — Layer123.com, 2012. **http://web.archive.org/web/20121014053339/http: //www.layer123.com/sdn-agenda/.**

136. **Maistre R. Tier 1 carriers tackle telco SDN // Light Reading.— 2012. http://www.lightreading.com/ carrier-sdn/nfv-(network-functions-virtualization) /tier-1-carriers-tackle-telco-sdn/d/d-id/699161.**

137. Mulligan U. Etsi publishes first specifications for network functions virtualisation.— www.etsi.org.**http://www.etsi.org/news-events/news/ 700-2013-10-etsi-publishes-first-nfv-specifications.**

138. Madhusanka L. Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture. — UK: John Wiley, 2015. — 438 pp.

139. Peuhkuri M. IP Quality of Service. — Helsinki: Helsinki University of Technology, Laboratory of Telecommunications Technology, 1999.

ПРИЛОЖЕНИЯ

АКТЫ ОБ ИСПОЛЬЗОВАНИИ

AFORE
solutions international

Представництво «ЕФОР СОЛУШНЗ ІНТЕРНЕШНЛ ІНК.»
м. Харків, 61010, вулиця Нетеченська, 25, поверх 4.
код ЄДРПОУ 26548706 тел. 714-04-58

"ЗАТВЕРДЖУЮ"
Директор Представництва
"Ефор Солушнс Інтернешнл Інк"

Стадченко В.Г.

серпня 2016 р.

АКТ ПРО ВИКОРИСТАННЯ

результатів дисертаційної роботи Дорожинського В.В. "Математичні моделі для специфікації та аналізу компонентів керованих подіями систем"

Ми, що підписалися нижче:

Директор Представництва	Стадченко В.Г.
Заступник Директора Представництва	Доля Г.М.
Інженер-програміст	Руккас К.М.

склали цей акт про те, що у 2016 році в Представництві "Ефор Солушнс Інтернешнл Інк" під час виконання проекту "Transition" були використані результати дисертаційного дослідження аспіранта кафедри теоретичної та прикладної інформатики Харківського національного університету імені В.Н. Каразіна Дорожинського Володимира Володимировича " Математичні моделі для специфікації та аналізу компонентів керованих подіями систем".

Результати дисертаційної роботи використані шляхом залучення математичних моделей та методів аналізу, запропонованих у роботі, під час проектування та оптимізації архітектурних рішень програмних компонентів системи. Зокрема, використання математичної моделі компонента обробки складних подій та метода його синтезу дозволило значно спростити процес виявлення аномалій, що можуть виникати підчас передачі даних у хмарних телекомунікаційних мережах. За результатами дослідження і в процесі використання моделі обробки складних подій встановлено, що використання результатів дисертаційної роботи Дорожинського В.В. "Математичні моделі

для специфікації та аналізу компонентів керованих подіями систем" дозволило суттєво покращити якість передачі аудіо та відео даних за рахунок попереднього статичного аналізу архітектури програмної системи управління трафіком.

Даний акт не є підставою для висунення обопільних фінансових вимог.

Директор Представництва _____ Стадченко В.Г.

Заступник Директора Представництва _____ Доля Г.М.

Інженер-програміст _____ Руккас К.М.

ТОВ «КЛАУД ВОРКС»
61001, м. Харків, вул.Ганни 20А
тел: +3 80 (57) 768-07-42
Код ЄДРПОУ - 37462274
Р/р 26053210339527 в АТ «ПроКредит Банк», МФО 320984

" ЗАТВЕРДЖУЮ"
Директор компанії "Клауд Воркс"
Свербілов Д.С.

"10" серпня 2016

АКТ ПРО ВИКОРИСТАННЯ

результатів дисертаційної роботи Дорожинського В.В. "Математичні моделі для специфікації та аналізу компонентів керованих подіями систем"

ми, що підписалися нижче:

Свербілов Д.С.

Шишкіна А.О.

Зацко А.В.

склали цей акт про те, що у 2016 році в компанії "Клауд Воркс" були використані під час виконання проекту "Система управління логістики підприємства" результати дисертаційного дослідження аспіранта кафедри теоретичної та прикладної інформатики Харківського національного університету імені В.Н. Каразіна Дорожинського Володимира Володимировича " Математичні моделі для специфікації та аналізу компонентів керованих подіями систем".

Результати дисертаційної роботи використані шляхом залучення математичних моделей та методів аналізу, запропонованих у роботі, підчас проектування та оптимізації архітектурних рішень програмних компонентів системи. Зокрема, використання математичної моделі компонента обробки складних подій та метода його синтезу дозволило значно спростити процес

виявлення аномалій, що можуть виникати підчас передачі даних у віртуальних хмарних мережах. За результатами дослідження і в процесі використання моделі обробки складних подій встановлено, що використання результатів дисертаційної роботи Дорожинського В.В. "Математичні моделі для специфікації та аналізу компонентів керованих подіями систем" дозволило суттєво покращити якість передачі аудіо та відео даних за рахунок попереднього статичного аналізу архітектури програмної системи управління трафіком.

Даний акт не є підставою для висунення обопільних фінансових вимог.

Свербілов Д.С.

Шкуратов А.О.

Гецко А.О.