

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки	09.04.02 «Информационные системы и технологии»
Магистерская программа	Проектирование, разработка и эксплуатация информационных систем

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**Моделирование и обнаружение сложных событий в
системах потоковой обработки событий и исполнение
активных правил**

Студент	_____	Кривокоров Георгий Владимирович
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Шибанов С.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Рецензент	_____	_____
	(должность, место работы)	(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2023

Содержание

Введение.....	4
1 Аспекты обнаружения сложных составных событий в системах потоковой обработки данных	6
1.1 Основы потоковой обработки событий с применением активных правил..	6
1.1.1 Современные подходы к обработке событий	6
1.1.2 Потоковая обработка данных и событий.....	11
1.1.3 Интеграция потоковой обработки событий и механизма активных правил	13
1.2 Задача обнаружения сложных составных событий в системах потоковой обработки данных	15
1.2.1 Классификация событий	15
1.2.2 Общий сценарий и этапы обнаружения событий в системах потоковой обработки данных	17
1.2.3 Особенности обнаружения сложных составных событий.....	18
1.3 Обзор подходов к моделированию сложных составных событий.....	21
1.3.1 Алгебраическая временная модель сложных составных событий	21
1.3.2 Моделирование сложных составных событий на основе графов	23
1.3.3 Моделирование сложных составных событий на основе автоматов	25
1.3.4 Моделирование сложных составных событий на основе сетей Петри ...	26
1.4 Обзор реализаций систем обнаружения сложных составных событий .	27
Выводы	31
2 Модельные представления и алгоритмы обнаружения сложных составных событий	33
2.1 Модельное представление сложных составных событий в виде асинхронного процесса.....	33
2.2 Модельное представление сложных составных событий в виде раскрашенных условных сетей Петри	36
2.3 Алгоритм развертывания раскрашенной условной сети Петри для моделирования сложного составного события	39
2.4 Алгоритмы обнаружения сложных составных событий на раскрашенной условной сети Петри для различных режимов потребления.....	40
Выводы	46

3 Программные средства обнаружения сложных составных событий в составе сервиса потоковой обработки событий и исполнения активных правил ..	47
3.1 Сервис потоковой обработки событий и исполнения активных правил	47
3.2 Микросервис обнаружения сложных составных событий в составе сервиса ..	50
3.2.1 Архитектура микросервиса обнаружения сложных составных событий ..	50
3.2.2 Принцип обработки событий в микросервисе обнаружения сложных составных событий ..	52
3.2.3 Структура раскрашенной условной сети Петри в рамках микросервиса обнаружения сложных составных событий ..	54
Выводы ..	56
Заключение ..	58
Список использованных источников ..	60
Приложение А ..	63
Приложение Б ..	65

Введение

Актуальность. Системы потоковой обработки событий предназначены для обработки большого потока событий. Такие системы позволяют выявлять более существенные события, а также реагировать на них.

Понятие событие описывает какое-либо событие, которое может произойти в наблюдаемой системе или вне нее.

События, поступающие на вход таких систем потоковой обработки, можно разделить на элементарные (данные, полученные с датчиков, изменение данных, активность пользователей) и сложные. Сложные события – вычисляемые события, обнаруживаемые системой в результате обработки элементарных событий или других сложными событиями.

Сложные события в свою очередь можно разделить на агрегируемые и составные события.

Большинство систем обработки сложных событий, такие как Apache Flink, Арама, Kafka и т. п. строятся на обработке только агрегированных событий на основе запросов к потоку. Систем, обрабатывающие сложные составные события не обнаружено.

Сложные составные события задаются при помощи алгебры событий и определяются через временные отношения между события. При помощи моделирования на основе условных раскрашенных сетей Петри можно реализовать алгоритмы, позволяющие данные события обнаружить.

Таким образом, применяя модельные представления и алгоритмы появляется возможность реализовать программный компонент, позволяющий обеспечить надежность обнаружения сложных составных событий.

Объект исследования. Объектом исследования являются процессы обнаружения сложных составных событий в системах потоковой обработки событий и исполнение активных правил.

Предмет исследования. Предметом исследования являются модельные представления сложных составных событий, алгоритмы и программные средства обнаружения сложных составных событий в составе системы потоковой обработки событий и исполнение активных правил.

Цели и задачи. Целью работы является исследование модельных представлений и разработка алгоритмов и программных средств обнаружения сложных составных событий в системах потоковой обработки событий и исполнение активных правил.

Для достижения поставленной цели планируется решить ряд важных задач:

- 1) исследование и анализ проблем обнаружения сложных составных событий в системах потоковой обработки событий;
- 2) исследование структуры сложных составных событий и обоснование выбора модельного представления сложных составных событий;
- 3) алгоритмы отображения сложных составных событий в модельное представление на основе условных раскрашенных сетей Петри;
- 4) алгоритмы обнаружения сложных составных событий с учетом сценариев обработки входящих событий;
- 5) разработка архитектуры программных средств обнаружения сложных составных событий в составе сервиса потоковой обработки событий и исполнение активных правил.

1 Аспекты обнаружения сложных составных событий в системах потоковой обработки данных

1.1 Основы потоковой обработки событий с применением активных правил

1.1.1 Современные подходы к обработке событий

История обработки событий берет начало с 1960-х годов от использования событий в более ранних приложениях до использования событий в двадцать первом веке в коммуникациях, обработке информации и управлении предприятием.

Выделяет 5 временных этапа развития технологий обработки событий:

- 1) дискретное моделирование событий;
- 2) сети;
- 3) активные базы данных;
- 4) промежуточное ПО;
- 5) уровень стратегического управления.

Дискретное моделирование событий. Моделирование дискретных событий – применяется при проектировании, реинжиниринге и оптимизации промышленных и логистических систем, а также в большем числе организаций в сфере услуг.

Первые инструменты моделирования промышленных потоков дискретных событий появились в 1960-х годах с таким программным обеспечением, как Simula или Simgen. Также в 1970-х годах появилось программное обеспечение для моделирования, оснащенное упрощенным графическим интерфейсом и динамической анимацией.

В 1980-х годах появился рынок программного обеспечения для моделирования. Появились такие программы как SIMAN, Witness и Promodel, затем ExtendSim, разработавший свой дискретный событийный движок в 1990 году. В 1990-х годах массовое использование моделирования потоков появились в некоторых компаниях [1].

Сети. Технология обработки событий также была разработана как фундаментальная часть сети, начинается с ARPANET в 1969 году.

Акцент был сделан на использовании очень простых типов событий для построения связи между компьютерами по ненадежным линиям передачи. Связь была посредством пакетов. Пакет, представляет собой последовательность битов, которые кодируют небольшой объем данных вместе с источником и пунктом назначения пакета в сети. Компьютеры общаются, отправляя пакеты друг другу по линии передачи. Отправка и получение пакетов – это события.

Чтобы гарантировать, что пакет прибывает на другой конец, компьютеры используют протокол подтверждения получения каждого пакета или его повторной отправки, если подтверждение не получено в течение установленного времени. Если пакет потерян во время передачи, он будет повторно отправлен до тех пор, пока не будет доставлен. Протоколы связи – это методы обработки событий для обеспечения определенных свойств (например, надежности связи). Между отправителем и получателем может быть несколько «переходов», то есть промежуточных компьютеров, которые передают пакет дальше.

Таким образом, на самом низком уровне обработка событий в коммуникационной сети представляет собой обработку очень небольших сообщений.

Активные базы данных. В отличие от стандартных баз данных (БД) активная база данных (АБД) помимо выполнения действий пользователя, выполняет дополнительные действия в соответствии с правилами, заложенными в саму БД. Система управления активной базой данных (СУАБД) должна отслеживать возникновение событий, описанных в АБД, и осуществлять их обработку посредством запуска процедур, затрагивающих как саму АБД, так и ее окружение.

Для обеспечения реагирующего поведения СУАБД должна поддерживать соответствующие модели описания и выполнения правил. Классическая модель описания активных правил основывается на правилах ЕСА, которые содержат три компонента: событие (event), условие (condition) и действие (action) и

называются ЕСА-правилами (Event–Condition–Action). Компонент Событие (event) описывает какое-либо событие, которое может произойти в СУБД или вне нее. Компонент Условие (condition) проверяет контекст, при котором событие произошло. Действие (action) описывает процедуру, которая должна быть выполнена правилом, если соответствующее событие произошло и условие оказалось истинным. Действия могут менять структуру базы данных или набора правил, выполнять некоторый вызов поведения внутри базы данных или внешний вызов, информировать пользователя или системного администратора о какой-либо ситуации, прерывать транзакцию или принимать альтернативную линию поведения.

Классическая модель ЕСА-правил не учитывает, что в реальных приложениях поведение объекта часто зависит от его состояния (state). Состояние – это абстракция значений и связей объекта, которая определяет отклик объекта на получаемые события. В конкретном состоянии обрабатываются только те события, для которых явным образом описано поведение, любые другие события игнорируются. Это свойство состояний особенно важно – оно позволяет наделить объекты системы свойством изменчивости – способности изменять свое поведение в зависимости от возникающих событий [2].

Промежуточное ПО. Технология промежуточного программного обеспечения зародилась в конце 1980-х годов и бурно развивалась в различных направлениях в 1990-х и 2000-х годах. Это началось как попытка модульного разделения сетевых уровней и протоколов обмена сообщениями, которые становились важной инфраструктурой, соединяющей бизнес-приложения воедино. Идея заключалась в том, чтобы скрыть запутанные детали событий и различные протоколы передачи событий, с помощью которых сообщения передавались туда и обратно между пользователями или между пользователями и бизнес-приложениями. ПО промежуточного слоя пыталось скрыть события на уровне передачи сообщений, предоставляя функции связи более высокого

уровня в интерфейсе прикладного программирования или API. Пользователям не нужно ничего знать о том, как обрабатываются их сообщения и запросы.

Из Технологий Промежуточного ПО можно выделить: ESB, SOA, EDA.

ESB – это набор программного обеспечения, которое работает как единый центр для обмена сообщениями между информационными системами и приложениями. Сервисная шина позволяет легко настраивать маршруты движения сообщений, хранит историю сообщений, фиксирует путь каждого из них.

ESB – это не монолитная система, а слой в IT-инфраструктуре компании, который состоит из нескольких продуктов. Каждый из них берёт на себя одну или несколько функций сервисной шины.

Важные события, которые происходят с сообщениями и очередями, необходимо наблюдать и вовремя обрабатывать, реагируя на них. Компонент мониторинга позволяет задавать события, которые нуждаются в реакции, и сообщать о них поддержке.

SOA, или сервис-ориентированная архитектура, определяет способ многократного использования и взаимодействия программных компонентов через сервисные интерфейсы. Службы используют общие стандарты интерфейса и архитектурный шаблон, поэтому их можно быстро включать в новые приложения. Это снимает задачи с разработчика приложения, который ранее перерабатывал или дублировал существующую функциональность или должен был знать, как подключиться или обеспечить взаимодействие с существующими функциями [3].

Архитектура, управляемая событиями (EDA) – это архитектурная парадигма, в которой поведение определяется реакцией на события. В этой парадигме события подразумевают существенное изменение состояния. Это позволяет создавать системную архитектуру, в которой поток информации определяется событиями. В архитектуре, управляемой событиями, изменения состояния проходят через конечные точки, события и акторы эфемерны и не имеют состояния. Это создает отделение логики от данных, кеша и

состояния. Архитектуру, управляемую событиями, также иногда называют архитектурой, управляемой сообщениями, или архитектурой потоковой обработки, основанной на ее вариациях [4].

Уровень стратегического управления. Управление бизнес-процессами (BPM) использует методы для обнаружения, моделирования, анализа, измерения, улучшения и оптимизации бизнес-стратегии и процессов. Управление задачами фокусируется на отдельных задачах, тогда как BPM наблюдает за всем сквозным процессом.

Управление проектами относится к одноразовому объему работ, в то время как BPM фокусируется конкретно на повторяющихся процессах. Благодаря непрерывному реинжинирингу процессов организации могут оптимизировать свои рабочие процессы в целом, что приводит к повышению эффективности и экономии средств. Эта концепция не нова; Шесть сигм и принципы бережливого производства являются примерами методологий BPM.

Включая расширенную аналитику, мониторинг активности и возможности управления решениями, пакеты управления бизнес-процессами могут координировать людей, системы, информацию и материалы для достижения бизнес-результатов. В результате они помогают ускорить реализацию стратегий цифровой трансформации [5].

Комплексная обработка событий – это технология для агрегирования, обработки и анализа больших потоков данных, позволяющая получать информацию о событиях в реальном времени по мере их возникновения.

Комплексная обработка событий применяет знания предметной области из нескольких источников данных, чтобы понять, что происходит с точки зрения концепций высокого уровня и сложных событий. Комплексная обработка событий предназначена для вывода сложных событий из необработанных данных с использованием бизнес-шаблонов и концепций. Цель состоит в том, чтобы выявить значимые факты, которые можно использовать для принятия обоснованных решений.

Сложная обработка событий является обобщением традиционной потоковой обработки. Традиционная потоковая обработка связана с поиском низкоуровневых закономерностей в данных. Используя модели причинно-следственных связей и концептуальные иерархии, Комплексная обработка событий может делать высокоуровневые выводы о сложных событиях в бизнес-области.

Основное преимущество комплексной обработки событий заключается в том, что действия могут быть инициированы комбинацией событий, происходящих в разное время и в разных контекстах [6].

1.1.2 Потоковая обработка данных и событий

Обработка сложных событий заключается в обработке потока событий, происходящих на всех уровнях организации, при этом идентифицируются наиболее существенные события из потока событий, анализируется их влияние и в режиме реального времени предпринимаются соответствующие действия [7].

Система потоковой обработки событий должна обладать следующими характеристиками:

- постоянная обработка вновь поступающих данных;
- возможность принимать приблизительные результаты;
- возможность выбора приоритетных характеристик для QoS;
- использование доступных вычислительных ресурсов;
- обработка сложных событий.

Постоянная обработка вновь поступающих данных необходима для всех потоковых приложений. Это означает, что запрос выполняется длительное время и его необходимо многократно оценивать системой на предмет новых данных, пока запрос не будет завершен.

Приблизительные результаты приемлемы, когда точные результаты недоступны или, когда система потребляет слишком много критических ресурсов для вычисления точных результатов.

Многие приложения предъявляют очень специфические требования. Некоторые из общих требований включают: время отклика (также называемое задержкой кортежа), точность, пропускную способность и допустимое использование памяти. Поскольку эти требования не являются независимыми друг от друга (например, уменьшение задержки кортежа обычно увеличивает требования к памяти), выбор одного из них может существенно повлиять на другие. Следовательно, существуют компромиссы между этими метриками, которые необходимо сбалансировать.

Использование доступных ресурсов (например, циклов ЦП, памяти) для максимального увеличения их влияния на показатели критично для потоковых приложений. Система должна включать механизмы для оптимального использования ресурсов. Инструменты для планирования мощности также будут полезны для разработки потоковых приложений.

Обработка правил и уведомление являются другими важными требованиями многих потоковых приложений, которые обнаруживают события или условия и должны своевременно запускать правила, триггеры, действия при обнаружении аномальных или определяемых пользователем событий.

Невозможно удовлетворить вышеуказанные требования потоковых приложений, просто загружая потоки данных в современную СУБД и обрабатывая запросы. СУБД может быть полезна для обеспечения постоянного управления данными и поддержки управления транзакциями для некоторых потоковых приложений. Однако он не предназначен для получения постоянных результатов. Традиционная СУБД не поддерживает требования и предполагает, что все (а также точные) результаты должны быть сгенерированы для запроса. Поэтому для поддержки новых требований, взятых из большого числа появляющихся потоковых приложений, используется другая парадигма потоковой обработки.

1.1.3 Интеграция потоковой обработки событий и механизма активных правил

Активные правила моделируют и обеспечивают выполнение корпоративных требований, таких как мониторинг ситуации, с использованием событий, условий и действий. Эти правила называются активными, поскольку они делают лежащее в их основе приложение или систему, такую как система управления базами данных, активными (т. е. активно реагирующими на изменения, используя парадигму проталкивания). События играют критическую роль в активных правилах, поскольку они определяют и обнаруживают возникновение интереса в реальном мире и запускают связанные правила.

Традиционно активные правила использовались для различных приложений в различных областях, таких как мониторинг ситуации, моделирование рабочих процессов и системы управления реляционными и объектно-ориентированными базами данных.

Активные правила и непрерывные запросы к потоковым данным тесно связаны. Тем не менее, в отличие от традиционных баз данных, данные не хранятся постоянно, а передаются через систему с колеблющейся скоростью поступления. Таким образом, интеграция механизма активных правил и потоковой обработки событий требует точной синхронизации между запросами и правилами и тщательного учета задержки и использования ресурсов.

Одним из примеров интеграции потоковой обработки событий и активных правил является программное обеспечение «HyReminder» [8].

Технологии CEP используются для эффективного отслеживания шаблонов событий, таких как последовательность, когда медработник покидает палату пациента, не дезинфицирует руки, а затем вошел в палату другого пациента.

Такая последовательность действий будет считаться нарушением правил гигиены рук. Помимо обнаружения с ложных событий, система «HyReminder» требует возможности задавать логические правила, напоминающие медработникам о необходимости выполнения соответствующей надлежащей

гигиены при обнаружении неизбежного нарушения гигиены руки ли фактического наблюдаемого нарушения.

Ссылаясь на правила гигиены рук, применяемые в большинстве больниц США, создаются запросы и активные правила для «HyReminder», используя следующую методологию:

- 1) моделируется конкретная последовательность поведения медработников с помощью шаблонных запросов;
- 2) моделируется эффективность гигиены рук медработников тремя статусами, а именно соблюдением или «зеленым», предупреждением или «желтым» и нарушением или «красным»;
- 3) моделируется логика смены статуса медработника, а именно определенную последовательность действий, которая приводит работника в другой статус, как это предусмотрено в правилах гигиены рук, используя активные правила.

Внутренний интерфейс позволяет системным администраторам задавать и отлаживать запросы и правила. Интерфейс системного администрирования предлагает окно для редактирования запросов и правил в текстовом виде. Интерфейс также отображает входные потоки событий, выходные потоки для каждого запроса и производительность системы в реальном времени. Пользователи получают возможность отправлять свои собственные запросы и тестировать их как на текущих, так и на постоянных данных событий.

Медсестры в режиме реального времени могут наблюдать текущее состояние соблюдения гигиены каждым медработником в указанном отделении интенсивной терапии. Окно мониторинга на основе карты в хронологическом порядке отображает каждого работника как движущийся объект на карте отделения интенсивной терапии. Чтобы представить наиболее актуальный статус работника, определяемый активными правилами, определенными в системе, наблюдаемый объект обозначается разными цветами.

1.2 Задача обнаружения сложных составных событий в системах потоковой обработки данных

1.2.1 Классификация событий

Для обнаружения общих признаков событий, и для выработки алгоритмов обработки и выявления событий, события необходимо классифицировать.

События в системах потоковой обработки данных можно классифицировать по различным критериям. Классификация производится по следующим критериям: степень сложности, соответствие с шаблоном, степень предсказуемости, степень спланированности, степень формальности [9].

Классификация событий в системах потоковой обработки данных представлена в таблице 1.

Таблица 1 – Классификация событий

Вид события		Описание
Элементарное		Событие, которое невозможно разложить на более простое.
Сложное	Агрегированное	Событие, строящееся на основе запросов к потоку событий
	Составное	Событие, строящееся на основе алгебры событий
Временное	Одноразовое	Выполняется только один раз.
	Периодическое	Повторяется в регулярном интервале.
Явно заданное		Событие, которое явно описано пользователем.
Стандартное		Событие, которое имеет стандартную структуру.
Нестандартное		Событие, которое имеет отклонения от стандартной структуры события.

Продолжение таблицы 1

Вид события	Описание
Ожидаемое	Событие, которое ожидает поток обработки событий.
Неожидаемое	Событие, которое не ожидает не один из потоков обработки событий.
Запланированное	Событие, наступление которого запланировано в данный момент времени.
Незапланированное	Событие, наступление которого незапланировано в данный момент времени.
Зарегистрированное	Событие, которое зарегистрировано в системе.
Незарегистрированное	Событие, которое не зарегистрировано в системе.

Системы обработки событий получают и обрабатывают большое количество событий, происходящих как вне, так и внутри системы в результате выполнения различных действий. События извлекаются из систем мониторинга, агрегируются и преобразуются. События подразделяются на элементарные (простые) и сложные.

Элементарное событие связано с получением данных от внешних систем мониторинга в определенный момент времени и обусловлено некоторой активностью пользователей, приложений или изменением данных.

Сложные события являются вычисляемыми событиями, которые обнаруживаются системой на основе различных операций над элементарными и/или другими сложными событиями. Сложные события подразделяются на агрегированные и составные события.

Агрегированное событие – это сложное событие, построенное на основе запроса к входному потоку элементарных событий.

Составное событие – это комбинация элементарных и/или сложных событий, с применением ограничений, логических отношений и комбинаций временной последовательности.

1.2.2 Общий сценарий и этапы обнаружения событий в системах потоковой обработки данных

В литературе существует обобщенный сценарий обнаружения событий в системах потоковой обработки данных. Можно выделить 4 этапа обнаружения событий (рисунок 1).

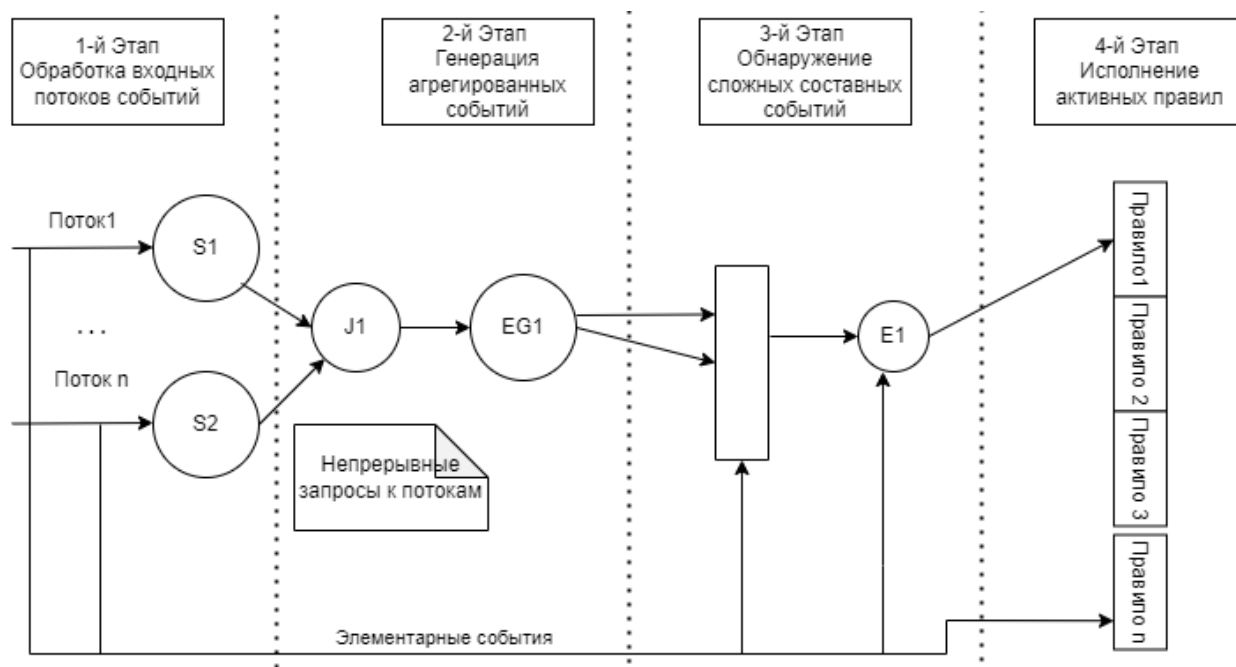


Рисунок 1 – Этапы обработки событий

На первом этапе осуществляется обработка входных потоков событий. Потоки событий фильтруются и подготавливаются для дальнейшей обработки.

На втором этапе происходит генерация агрегированных событий. Агрегированные события возникают путем непрерывных запросов к потокам событий. После агрегации события поступают в очередь событий.

Третий этап предназначен для обнаружения сложных составных событий. Этот этап представляет собой компонент обработки сложных событий, где сложные события обнаруживаются на основе определения выражений событий.

Четвертый этап – этап исполнения активных правил. Исполнение активных правил – это компонент, который обрабатывает правила, связанные с событиями. При обнаружении событий оцениваются условия, и если они оцениваются как истинные, выполняются соответствующие действия. Правила могут вызывать события, приводящие к выполнению обнаружения событий и выполнения правил.

1.2.3 Особенности обнаружения сложных составных событий

Простые (или примитивные) события основаны на предметной области. Сложные составные события создаются из одного или нескольких простых или сложных событий с использованием операторов событий. Существующие языки спецификации событий и механизмы обнаружения основаны на четко определенной временной семантике на основе точек и интервалов. Хотя обе семантики событий широко используются в обработке событий, было показано, что они неадекватны для поддержки многих областей приложений [10].

Возникновение сложного события основано на событиях инициатора, детектора и терминатора. Инициатор – это составляющее событие, появление которого запускает сложное событие. Детектор – это составляющее событие, возникновение которого обнаруживает и инициирует сложное событие. Терминатор является составляющим событием, которое отвечает за завершение сложного события, т. е. невозможно более возникновение сложного события с тем же событием инициатором. Все операторы имеют инициатор и детектор, но не терминатор. Кроме того, детектор и терминатор могут быть одним и тем же состав.

Основной недостаток спецификации и обнаружения сложных событий с использованием точечной, интервальной или обобщенной семантики заключается в том, что событие обнаруживается только тогда, когда происходят все необходимые составляющие его события. Во всех остальных случаях вхождения составных событий игнорируются (т. е. удаляются). Это критическая проблема, поскольку активные правила, связанные с событием, срабатывают при

обнаружении этого события. Другими словами, событие решает, когда активировать связанные с ним активные правила. Из-за критической роли, которую играют события в активных правилах, этот метод обнаружения событий не позволяет активным правилам моделировать политики, требуемые приложениями.

Давайте возьмем политику и смоделируем ее, используя существующую спецификацию обнаружения сложных событий. «Предпримите действия, когда первое возникновение интереса предшествует второму возникновению интереса». Например, в университетском городке между 00:00 и 06:00 дверь здания должна быть открыта до того, как будет открыта дверь любого офисного помещения. Эта политика может быть смоделирована с использованием активных правил и оператора сложного события SEQUENCE. Оператор SEQUENCE обнаруживается всякий раз, когда первое событие предшествует второму событию. После обнаружения события срабатывают правила, связанные с событием, проверяются условия и предпринимаются действия. С другой стороны, когда второе событие происходит без появления первого события, это событие игнорируется. В реальных жизненных ситуациях это нельзя игнорировать, поскольку это может указывать на взлом или нарушение безопасности, и необходимо предпринять ряд действий [11].

Использование обработки событий в различных предметных областях требует дополнительных возможностей семантики обнаружения текущих событий, чтобы сделать вывод, что было обнаружено событие, составляющее сложное событие, но не другие события, чтобы завершить обнаружение этого сложного события. Для выявления таких случаев мы предлагаем режимы обнаружения сложных событий. Результат простого обнаружения событий одинаков во всех режимах. Эти режимы позволяют предпринимать дополнительные действия и альтернативные действия, когда событие не завершено из-за возникновения/ненаступления составляющих событий.

При оставлении событий может быть несколько вхождений событий, которые могут удовлетворять составному событию, например, рассмотрим составное событие ($E_1 ; E_2$) и три вхождения события для E_1 : e_{11} , e_{12} и e_{13} .

При наступлении e_{21} потребление события должно быть четко определено, а именно, какие события E_1 объединять с e_{21} .

Потребление событий используется для принятия решения о том, какие события-компоненты из истории событий считаются составным событием и как параметры составного события вычисляются из его компонентов. Режимы потребления событий классифицируются в следующих контекстах событий:

В режиме потребления «Недавние» в составном событии используется только самый новый экземпляр события-инициатора E . В приведенном выше примере экземпляр e_{11} события E_1 является инициатором составного события ($E_1 ; E_2$). Если обнаруживается новый экземпляр события E_1 (например, e_{12}), более старый экземпляр удаляется. перезаписывается новым экземпляром. Затем экземпляр e_{21} события E_2 объединяется с самым новым доступным событием: (e_{13} , e_{21}). Когда составное событие обнаружено, все компоненты этого события, которые не могут быть будущими инициаторами, удаляются из истории событий. Этот режим потребления удобен, например, в системах, где есть высокая скорость показаний датчиков, и не имеет значения, если некоторые показания будут потеряны.

В режиме потребления «Хроника» для события пара инициатор-терминатор уникальна. Самый старый инициатор и самый старый терминатор для каждого события объединяются для формирования составного события. В этом примере экземпляр e_{21} объединяется с самым старым доступным событием E_1 : (e_{11} , e_{21}). Когда составное событие обнаружено, все составляющие составного события удаляются из истории событий. Этот режим потребления используется, когда необходимо поддерживать связь между событиями.

В режиме потребления «Непрерывный» каждый инициатор составного события начинает обнаружение этого составного события. Возникновение события-терминатора может затем обнаруживать одно или несколько вхождений

одного и того же составного события, т. е. терминатор завершает те составные события, в которых были обнаружены все компоненты (за исключением терминатора). В примере e21 объединяется со всеми инициаторами события E: ((e11, e21) (e12, e21) (e13, e21)) и не удаляет использованные события. Режим непрерывного потребления отличается от режимов недавнего и хронического потребления тем, что в недавнем и хроническом режимах один инициатор связан с одним терминатором, а в режиме непрерывного потребления один терминатор связан с одним или несколькими инициаторами. Основная проблема с этим режимом потребления заключается в том, что он создает комбинации событий, в которых некоторые или все события представляют интерес. Это увеличивает нагрузку на систему и требует большей емкости хранилища.

В режиме потребления «Кумулятивный» все вхождения типа события накапливаются как экземпляры этого события до тех пор, пока не будет обнаружено составное событие. В примере e21 объединяется со всеми доступными вхождениями события E1(e11, e12, e13, e21) и удаляет использованные события. Когда обнаружен терминатор, т. е. обнаружено составное событие, все экземпляры событий, составляющие составное событие, удаляются из истории событий.

1.3 Обзор подходов к моделированию сложных составных событий

1.3.1 Алгебраическая временная модель сложных составных событий

Составные события могут быть составлены на основе определенной алгебры композиции событий. Следовательно, составное выражение события определяется рекурсивно, как выражение события, сформированное с использованием набора примитивных выражений события, операторов событий и выражений составного события [12].

Состав составного события зависит от обнаружения составляющих его событий. Процесс композиции поддерживается композиционным механизмом. Это поддерживает структуру составного события и желаемый порядок

вхождения событий-компонентов в нем. Каждый раз, когда происходит событие, оно вставляется в механизм. Если событие вносит вклад в композицию какого-либо составного события, выводится новое состояние композиции, а старое удаляется. Некоторое конечное состояние указывает на законченную композицию составного события. Если событие входит в композицию нескольких составных событий, приоритеты правил определяют порядок их оценки. Важно, чтобы все события, вносящие вклад в составное событие, происходили в правильном порядке. Этот порядок определяется составным выражением события и, в частности, операторами события.

Алгебраические операторы позволяют выразить состав событий для обозначения новых типов событий.

Диапазон операторов событий варьируется от системы к системе. Наиболее распространенными можно выделить 8.

Список операторов алгебры событий представлен в таблице 2.

Таблица 2 – Операторы алгебры событий

№	Название	Обозначение	Пример использования
1	Дизъюнкция	OR, $\vee$	$E1 \vee E2$
2	Конъюнкция	AND, Δ	$E1 \Delta E2$
3	Последовательность	SEQ	$SEQ(E1; E2)$
4	Одновременный	SIM	$SIM(E1; E2)$
5	История	TIMES	$TIMES(n, E1)$
6	Отрицание	NOT, $\neg$	$(E2)[E1, E3]$
7	Закрытие	CLOSURE	CLOSURE
8	Любой	ANY	$ANY(m, E1, E2, \dots, En)$

Оператор «Дизъюнкция» обозначает, что составное событие происходит, когда произошло либо $e1$, либо $e2$.

$$(E1 \nabla E2)(t) = E1(t) \vee E2(t).$$

Оператор «Конъюнкция» обозначает, что составное событие происходит, когда и $e1$, и $e2$ произошли в любом порядке.

$$(E1 \Delta E2)(t) = (\exists t1)((E1(t1) \wedge E2(t)) \vee (E2(t1) \wedge E1(t))) \wedge t1 \leq t).$$

Оператор «Последовательность» обозначает, что составное событие происходит, когда событие $e1$ происходит до $e2$.

$$SEQ(E1; E2)(t) = (\exists t1)(E1(t1) \wedge E2(t) \wedge (t1 < t)).$$

Оператор «Одновременный» обозначает, что составное событие происходит, когда события $e1$ и $e2$ происходят одновременно.

$$SIM(E1; E2)(t) = (E1(t) \vee E2(t) \vee (t1 = t2)).$$

Оператор «Отрицание» обозначает, что составное событие обнаруживает не наступление события $e1$ во временном интервале.

$$(E2)[E1, E3](t) = (\exists t1)(\forall t2)(E1(t1) \wedge \sim E2(t) \wedge E3(t) \wedge \\ \wedge ((t1 \leq t2 < t) \rightarrow \sim (E2(t2) \vee E3(t2)))).$$

Оператор «Закрытие» обозначает, что составное событие возникает только после того, как сигнализируется первое событие времени $e1$, независимо от более поздних вхождений e во временном интервале.

Оператор «Любой» это составное событие возникает, когда произошло m из n различных событий.

$$ANY(m, E1, E2, \dots, En)(t) = (\exists t1)(\exists t2) \dots (\exists tm - 1)(Ei(t1) \wedge \\ \wedge Ej(t2) \wedge \dots \wedge Ek(tm-1) \wedge E1(t) \wedge (t1 \leq t2 \leq \dots \leq tm-1 \leq t) \wedge \\ \wedge (1 \leq i, j, \dots, k, l \leq n) \wedge \\ \wedge (i \neq j \neq \dots \neq k \neq l)).$$

Оператор «История» обозначает, что составное событие сигнализируется, когда событие $e1$ происходит n раз в течение интервала времени.

1.3.2 Моделирование сложных составных событий на основе графов

Граф событий основан на двойственном соотношении состояния и времени системных событий. События изменяют значения переменных, описывающих

состояние системы. События также влияют на возникновение будущих событий. Влияние событий на переменные состояний системы представлено вершинами (узлами). Отношения между событиями представляются в виде направленных ребер (дуг) между вершинами событий [13].

Графы событий не являются блок-схемами программы, а представляют собой представление структуры системы, которое можно использовать в качестве предварительного шага в разработке имитационной модели. То, что мы будем называть графом событий, строится следующим образом. Сначала определяются и нумеруются системные события. Событие определяется преобразованием переменных состояния системы, которое происходит при возникновении события. Вершины событий на ориентированном графе представляют собой изменения состояния системы. Набор переменных состояния, которые могут быть изменены событием i , будет обозначаться как S_i . Переменные состояния могут быть либо стохастическими, либо детерминированными. Схожие вершины событий различаются атрибутами.

События, отличающиеся только значениями своих атрибутов, представлены на графе событий одной вершиной. Соединяющие вершины событий представляют собой направленные ребра, указывающие, как события влияют на возникновение другого события. Различают два класса ребер: ребро (сплошное) для планирования событий и ребро (штриховое) для отмены событий. Ребра имеют атрибуты времени задержки, вершины события возникновения (голова ребра), вершины события завершения (хвост ребра) и условий, связанных с инициированием события.

Преимущество графов событий заключается в возможности качественного и количественного анализа, но решение на основе графов событий является устаревшим и не универсальным.

1.3.3 Моделирование сложных составных событий на основе автоматов

Конечный автомат – это абстрактная модель вычислений, которая используется для моделирования логики. В общем виде конечный автомат состоит из:

- набор потенциальных входных событий;
- набор вероятных выходных событий, соответствующих потенциальным входным событиям;
- набор ожидаемых состояний, которые может демонстрировать система.

В работе представлено моделирование событий на основе расширенного конечного автомата [14].

Расширенный конечный автомат описывается пятью кортежами.

$$\text{EFSM} = (Y, I, O, V, T),$$

где Y – набор состояний;

I – набор входных событий;

O – набор выходных событий;

V – множество предикатов и функций обновления переменной;

T – набор переходов.

Пример расширенного конечного автомата представлен на рисунке 2.

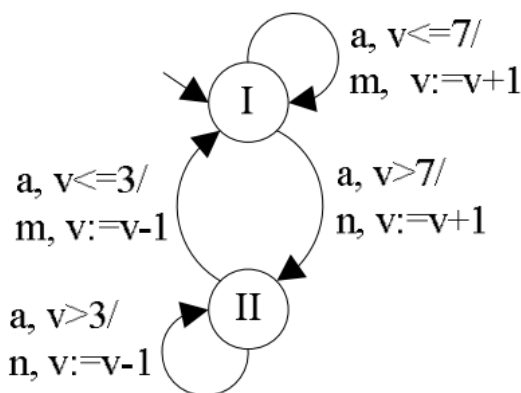


Рисунок 2 – Пример представления расширенного конечного автомата

В данном примере автомат содержит набор состояний $Y = \{I, II\}$, входное событие $I = \{a\}$, выходные события $O = \{m, n\}$. Предикаты $v > 3$, $v \leq 3$, $v > 7$, $v \leq 7$, а также функции обновления переменной $v: = v-1$, $v: = v+1$.

Достижение автоматом заключительного состояния рассматривается как факт обнаружения сложного составного события.

Проблемой моделирования сложных составных событий на основе автоматов заключается в ограниченности использования. Соответственно с помощью конечных автоматов невозможно реализовать полный функционал для обработки и обнаружения сложных составных событий.

1.3.4 Моделирование сложных составных событий на основе сетей Петри

Сети Петри является мощным формализмом для моделирования в информатике, системной инженерии и многих других дисциплинах. Сети Петри сочетают в себе четко определенную математическую теорию с графическим представлением динамического поведения систем. Теоретический аспект сетей Петри позволяет точно моделировать и анализировать поведение системы, а графическое представление сетей Петри позволяет визуализировать изменения состояния моделируемой системы. Эта комбинация является основной причиной большого успеха сетей Петри. Следовательно, сети Петри используются для моделирования различных видов динамических систем, управляемых событиями.

Сеть Петри – это особый вид двудольных ориентированных графов, содержащих три типа объектов. Этими объектами являются места, переходы и направленные дуги. Направленные дуги соединяют места с переходами или переходы с местами [15].

Сеть Петри представляет из себя кортеж вида

$$N = \{P, T, F\},$$

где P – множество мест;

T – множество переходов;

F – множество дуг.

Раскрашенные сети Петри являются дополнением к существующим языкам и используются для моделирования и описания поведения параллельных и распределенных систем.

Расширив раскрашенные сети Петри до условных раскрашенных сетей Петри, появляется возможность моделирования сложных составных событий.

Обнаружение события представляет собой процесс получения результата свершения (обнаружено событие или нет). Поэтому его удобнее моделировать как место, а событие можно моделировать как «цвет». Если событие обнаружено, то на это место кладется цветной токен. Более того, действие правила может быть событием другого правила, моделирующие действия как места. Условная часть правила будет оцениваться после обнаружения события. Этот переход называют условным переходом, чтобы подчеркнуть важность условий. Если в каждой входной позиции есть маркер, то переход разрешен, а если его условие оценено как истина, то переход возможен. Вышеупомянутая идея может быть объяснена интуитивно, правило отображается в переход, событие и действие отображаются в места ввода и вывода перехода.

1.4 Обзор реализаций систем обнаружения сложных составных событий

Рассмотрим несколько примеров реализаций систем потоковой обработки событий.

Одной из реализаций система обнаружения сложных составных событий представлена компанией «Huawei Technologies Co. Ltd.».

Поток событий путем запросов делится на несколько очередей агрегированных событий. Каждой очереди агрегированных событий соответствует свой узел обработки событий. В данных узлах путем вычислений, осуществляется обработка событий в сложное составное событие.

На рисунке 3 представлена архитектура обработки сложных событий от компании «Huawei Technologies Co. Ltd.» .

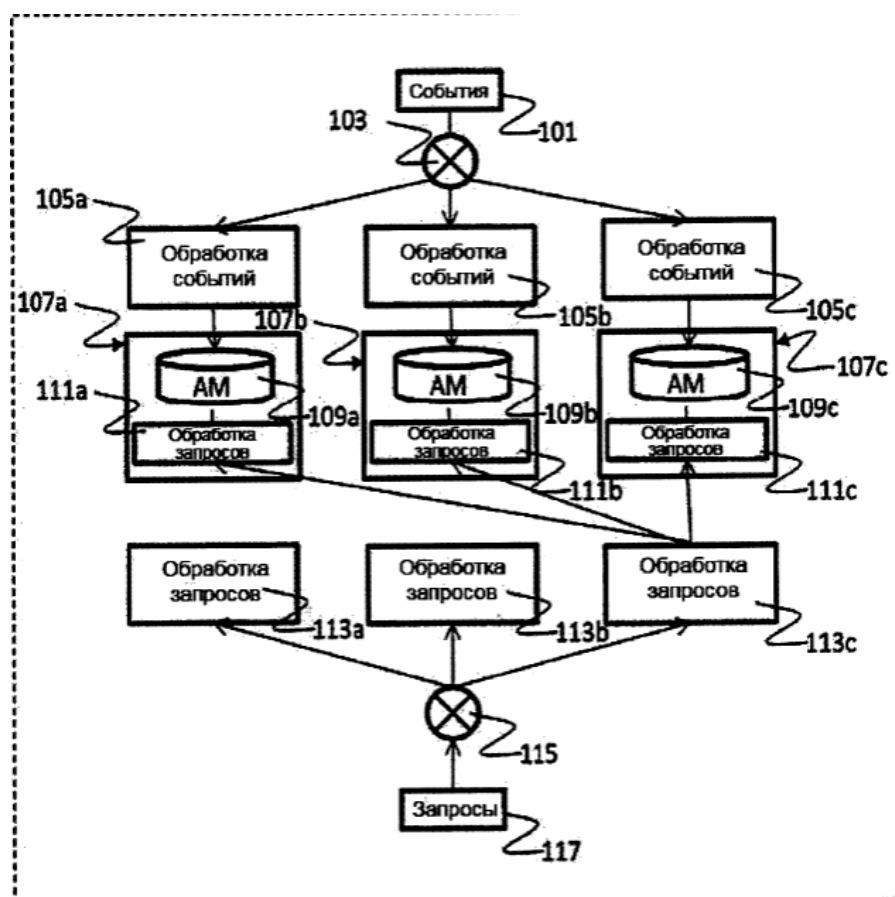


Рисунок 3 – Архитектура системы обработки сложных событий «Huawei Technologies Co. Ltd.»

Следующая система состоит из следующих компонентов: диспетчера источников данных, механизма непрерывной обработки запросов, диспетчера каталогов, планировщика, монитора и оптимизатора времени выполнения. Диспетчер источников данных принимает непрерывные потоки данных и вставляет входные кортежи в соответствующие входные очереди планов запросов. Монитор отслеживает различные характеристики потока входных данных, которые предоставляют полезную информацию для оптимизации запросов, планирования и управления. Он также контролирует выходные значения [16].

Механизм непрерывной обработки запросов отвечает за создание планов запросов и их динамическую оптимизацию. Менеджер каталога хранит и управляет всеми метаданными в системе, включая метаданные потока, подробные планы запросов, статистику и информацию о ресурсах. Планировщик

определяет, какой запрос, оператор или путь оператора выполняется в любой временной интервал на основе требований запроса и измеренных выходных значений. Оптимизатор времени выполнения получает измеренные значения выходных данных и корректирует планирование и сброс нагрузки в соответствии с требованиями.

Оптимизатор времени выполнения использует ряд механизмов доставки, таких как планирование, сброс нагрузки и управление допуском, для максимального удовлетворения запросов.

Архитектура системы обработки сложных составных событий представлена на рисунке 4.

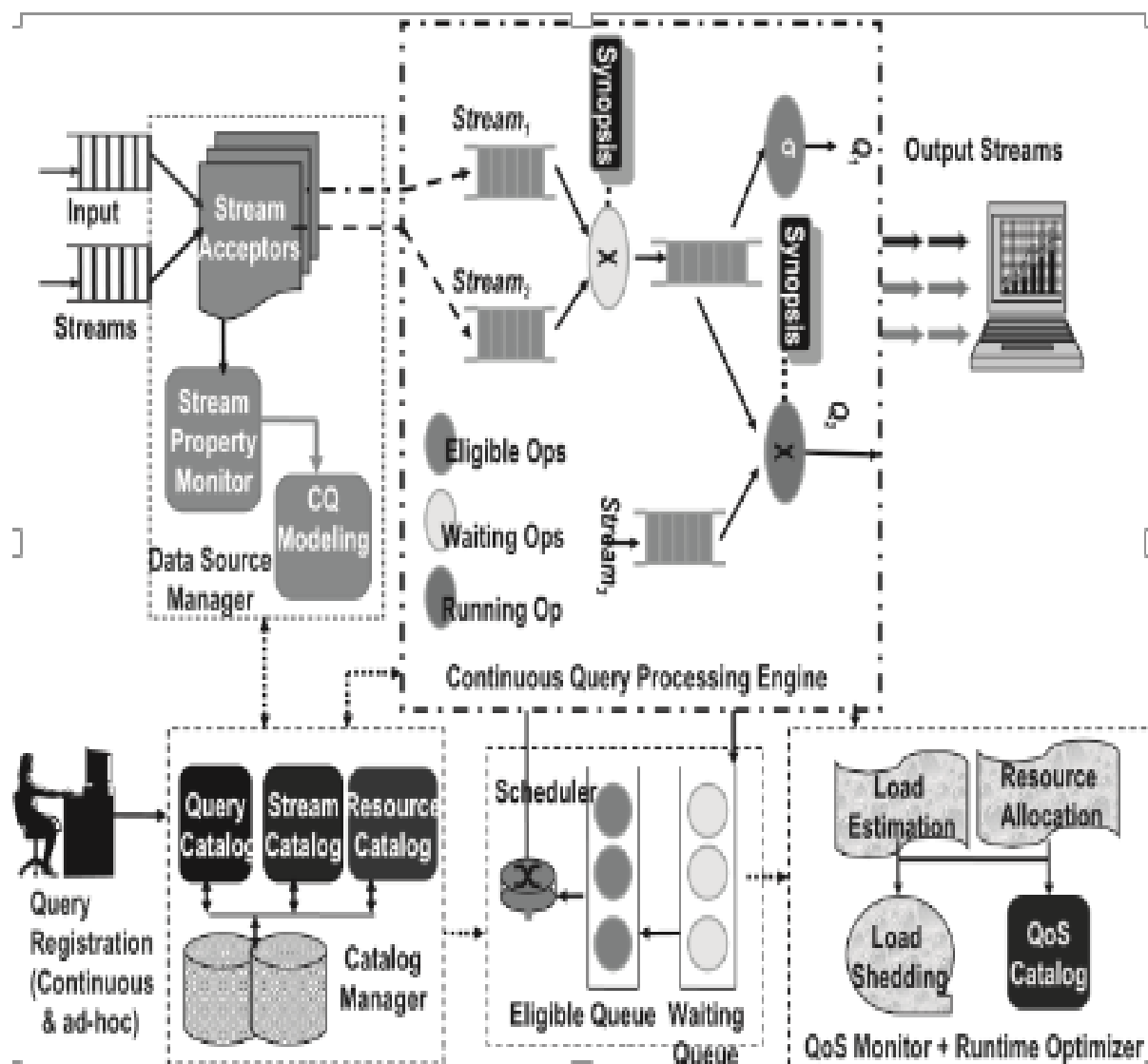


Рисунок 4 – Архитектура системы обработки сложных составных событий

Другим примером является «Система управления чрезвычайными ситуациями».

Архитектура данной система представлена на рисунке 5.

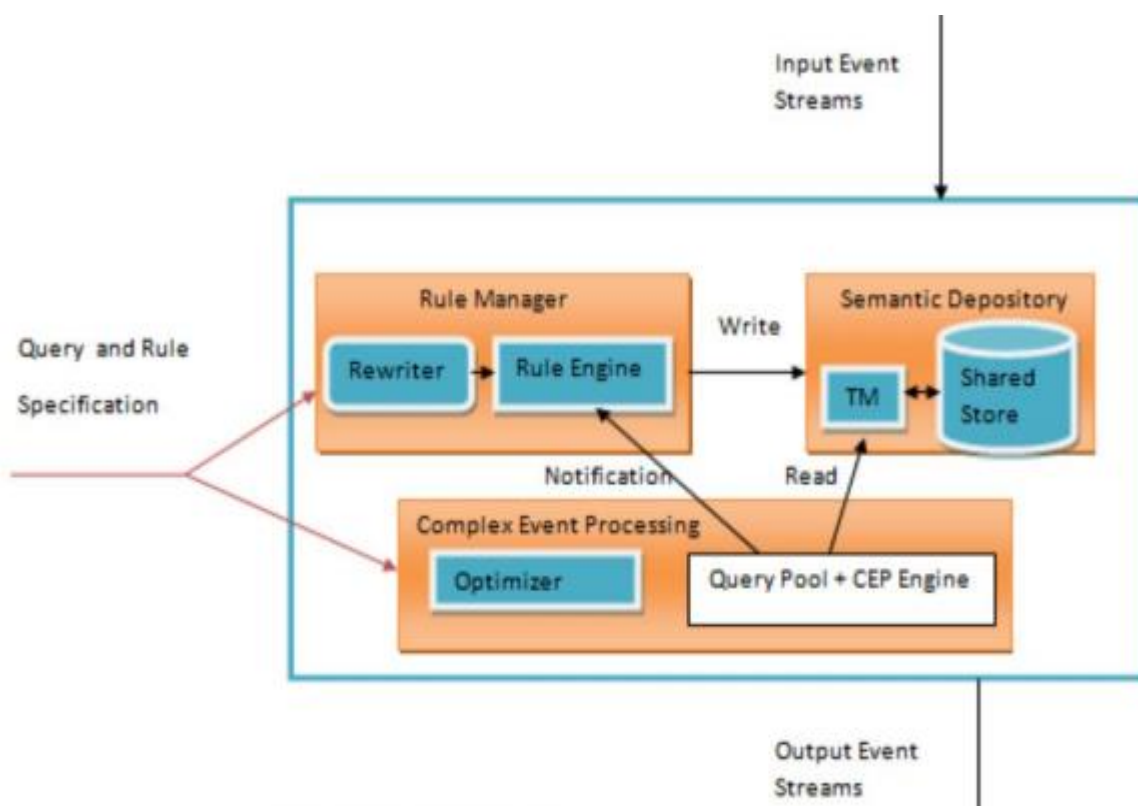


Рисунок 5 – Архитектура системы обработки сложных событий

Жизненные показатели пациента, обнаруженные датчиком, отправляется в систему обработки событий на основе активных правил. Анализатор временного контроля доступа, как только обнаруживается чрезвычайная ситуация, запускает управление чрезвычайными ситуациями [17].

Пациент и врач должны быть зарегистрированы в системе управления чрезвычайными ситуациями. Сервер обработки событий используется для наблюдения за пациентом. После того, как пациент зарегистрировался, идентификатор пациента будет автоматически обновляться на сервере, и все показатели здоровья пациента постоянно отслеживается сервером обработки сложных событий. Сервер используется для обнаружения ненормального состояния пациента.

Всякий раз, когда обнаруживается экстренная ситуация, сведения о экстренном пациенте хранятся отдельно на сервере. У каждого пациента есть

уникальный логин, и как только они войдут в наше приложение, они увидят значения непрерывного мониторинга этого пациента. После обнаружения чрезвычайной ситуации они смогут увидеть тип чрезвычайной ситуации, а также тип политики. Политика экстренной помощи также динамически обновляется администратором больницы. После обнаружения чрезвычайной ситуации аномальные значения пациента отправляются в компонент временной политики управления доступом. Переданные значения будут проверены и перенаправлены врачу [18].

Обработка событий на основе активных правил обеспечивает настройку мониторинга в режиме реального времени для обнаружения аномальных событий в основных показателях жизнедеятельности пациента. Активное правило обработки сложных событий обеспечивает эффективную обработку и обнаружение событий в режиме реального времени.

Выводы

Процесс обработки событий продолжается более шестидесяти лет и продолжает быть актуальным и в сегодняшнее время. В связи с усложнением модели событий, усложнились методы и принципы их обработки.

В данной главе было выделено понятие события и рассмотрены основные этапы развития обработки событий. Классификация видов событий позволило выделить элементарные и сложные события. А сложные в свою очередь разделили на агрегированные и составные. Сложные составные события строятся на основе операторов алгебры событий и такие события несут в себе особый интерес, так как сочетают в себе и элементарные, и агрегированные и составные события.

События, как и данные поступают в системы в виде потока. Соответственно, для определения сложных составных событий были рассмотрены этапы обработки событий.

Совместив потоковую обработку сложных составных событий и механизм активных правил, можно получить управляющую систему.

Для моделирования сложного составного события были рассмотрены такие модельные представления как: графы событий, конечные автоматы и раскрашенные условные сети Петри.

Приняв во внимания недостатки конечных автоматов и графов событий и конечных автоматов, было выбрано модельное представление в виде раскрашенных условных сетей Петри.

2 Модельные представления и алгоритмы обнаружения сложных составных событий

2.1 Модельное представление сложных составных событий в виде асинхронного процесса

Асинхронный процесс описывает отношения между двумя или более объектами, или событиями, которые взаимодействуют в одной системе, но не происходят через фиксированные промежутки времени. Из-за этого эти объекты или события не скоординированы друг с другом, и они не полагаются на существование друг друга для своего функционирования.

Асинхронным процессом можно назвать четверку типа:

$$P = \langle S, F, I, R \rangle,$$

где S – непустое множество событий;

F – отношение непосредственного следования событий, определяемое на множестве $S \times S$ ($F \subset S \times S$);

I – множество инициаторов ($I \subset S$);

R – множество результатов ($R \subset S$).

Инициаторы – подмножество событий, активирующих процесс. Назначение инициаторов осуществляется на основе смыслового содержания (семантики) процесса.

Результаты – подмножество, состоящее из финальных событий. Их выбор также опирается на семантику процесса.

В современном мире вычислительные машины и дискретные системы строятся на основе модульного принципа. Аппаратные средства и программное обеспечение, выполняются в виде отдельных функциональных структурных единиц. Последние объединяются композируются, комплексируются, агрегатируются, образуя сложные системы. Соответственно необходим аппарат, позволяющий строить описания функционирования системы по описаниям ее фрагментов – аппарат композиции. Данный аппарат основан на введении

определенных операций над компонуемыми асинхронными процессами и отличается от обычно используемых в автоматной проблематике методов композиции учетом динамических особенностей процессов.

Выделяют 3 вида композиции процессов: параллельная композиция, последовательная композиция, замыкание [19].

Параллельной композицией двух асинхронных процессов $P1$ и $P2$ называют асинхронный процесс $P3$, который моделирует совместное функционирование $P1$ и $P2$. При независимом функционировании $P1$ и $P2$ результат композиции $P3$ должен быть задан таким образом, что:

- события результата композиции представляются декартовым произведением событий составляющих композиции;
- если число ситуаций в $P1$ и $P2$ соответственно равно $|S1|$ и $|S2|$, то результат композиции $P3$ должен иметь число ситуаций $|S3| = |S1| \times |S2|$;
- инициаторами в $S3 \subseteq S1 \times S2$ должны быть объявлены ситуации s_{k3} , в которых обе подситуации s_{i1} и s_{j2} принадлежат инициаторам в $P1$ и $P2$;
- результатами в $S3$ должны быть объявлены ситуации, в которых обе подситуации s_{i1} и s_{j2} принадлежат результатам в $P1$ и $P2$;
- пара ситуаций s_{i3} и s_{j3} процесса $P3$ должна быть связана отношением $s_{i3} F s_{j3}$, если для одной из подситуаций в составе асинхронных процессов $P1$ или $P2$ выполняется $s_{i1} F s_{j1}$ или $s_{i2} F s_{j2}$.

Последовательная композиция двух процессов $P1$ и $P2$ предполагает их последовательное включение, при котором $P1$ играет роль начала результата композиции $P3$, а $P2$ – роль его окончания. Процесс $P1$ в составе композиции инициирует $P2$, а завершение функционирования последнего означает остановку $P3$.

Одно из требований $P2$ должен воспринимать сообщения, выдаваемые $P1$. Для этого требуется эквивалентность некоторых значений выходной компоненты ситуаций в $P1$ соответствующим значениям входной компоненты ситуаций в $P2$ или же эквивалентность некоторых результатов процесса $P1$ соответствующим инициаторам процесса $P2$.

Однако эти требования не являются единственными. Дело в том, что для активизации P_3 обе составляющие композиции P_1 и P_2 должны быть готовы к возбуждению своих инициаторов. Процесс P_1 , отработав после инициации, окажется в некотором результате, который должен быть воспринят процессом P_2 как инициатор. В свою очередь, процесс P_2 , отработав, окажется в одном из результатов и, следовательно, оба процесса P_1 и P_2 не могут быть повторно активированы. Для того чтобы результирующий процесс P_3 был возобновляемым, его необходимо подготовить к повторной инициализации, что может быть сделано с помощью процедуры приведения процесса. Естественно, еще одно требование: при функционировании P_2 в составе композиции в нем не должно возникать новых траекторий.

Операция замыкания является одноместной и выполняется над приведенным процессом, у которого по семантическим соображениям выделяются пары эквивалентных компонент ситуаций – входной и выходной – или пары эквивалентных инициаторов и результатов. Сначала рассмотрим замыкание по эквивалентным парам «результат–инициатор», принадлежащим соответственно множествам R^{**} и I^{**} . Тогда множества $I^* = I \setminus I^{**}$ и $R^* = R \setminus R^{**}$ являются несущественными инициаторами и результатами. Замыкание производится следующим образом:

- осуществляется редукция P по несущественным инициаторам I^* ;
- проводится редукция P по несущественным результатам R^* ;
- склеиваются попарно эквивалентные ситуации из R^{**} и I^{**} .

Таким образом, замыканием по эквивалентным парам “результат–инициатор” называется автономный асинхронный процесс $P_3(I^{**}, R^{**}) = \langle S_3, F_3 \rangle$, удовлетворяющий ограничениям:

- $S_3 \subseteq S^n(I^{**}) \cap S^n(R^{**})$;
- $F_3 = (F(I^{**}) \cup F'(I^{**})) \cap (F(R^{**}) \cup F'(R^{**}))$, где F и F' задают

причинно-следственные связи для не приведенного процесса и его репозиции, соответствующих приведенному АПР;

- пары эквивалентных ситуаций из I^{**} и R^{**} склеены.

Асинхронный процесс по существу является общей моделью описания динамики поведения параллельно функционирующих асинхронных систем. Эта модель задает допустимые последовательности действий над некоторыми объектами системы, каждой из которых соответствует некоторая траектория асинхронного процесса. В этом смысле можно говорить о том, что асинхронный процесс есть модель управляющей структуры системы. Асинхронный процесс может служить инструментом исследования весьма общих свойств и закономерностей параллельных систем, независимо от семантики их работы. Поэтому асинхронный процесс можно считать не интерпретированной моделью. С другой стороны, асинхронный процесс можно понимать, как метамодель, порождающую различные конкретные динамические модели (в том числе рассматриваемые далее).

2.2 Модельное представления сложных составных событий в виде раскрашенных условных сетей Петри

Раскрашенная условная сеть Петри представляет из себя кортеж из 10 элементов

$$CCPN = \{\Sigma, P, T, A, N, C, Con, D, \tau, I\},$$

где Σ – конечное множество непустых типов, называемых наборами цветов;

P – конечное множество мест;

$$P = P_{pri} \cup P_{com} \cup P_{vir} \cup P_{cor},$$

где P_{pri} – примитивное место;

P_{com} – место копирования;

P_{vir} – виртуальное место;

P_{cor} – композитное место.

T – конечное множество переходов;

$$T = T_{coru} \cup T_{comp} \cup T_{time},$$

где T_{coru} – переход копирования;

T_{comp} – композитный переход;

T_{time} – временной переход.

A – конечное множество дуг, в котором

$$P \cap T = P \cap A = T \cap A.$$

$$A = A_{inh} \cup A_{nor},$$

где A_{inh} – ингибиторная дуга;

A_{nor} – нормальная дуга.

N – узловая функция, определяется для $A \in P \times T \cup T \times P$;

C – цветовая функция, определяется для $P \in \Sigma$;

Con – функция оценки состояния. Определяется для T_{rule} или T_{comp} , как

$$\forall t \in T: [Type(Con(t)) = \beta].$$

D – интервальная функция времени, определяется для T_{comp} , как интервал $[d1, d2]$, соответственно начало и конец интервала, где $t \in T_{comp}$;

τ – функция метки времени. Определяется для $M(p) \in \{0\} \cup \mathbb{R}^+$, которые присваивают каждому месту отметку времени;

I – функция инициализации Он определяется для P в выражения, такие что

$$\forall p \in P: [Type(I(p)) = (C(p)ms, \tau(C(p)ms)];$$

В CCPN маркер представляет собой кортеж из четырех элементов:

$$\{p, c, data, timestamp\},$$

где $p \in P$;

$c \in C(p)$;

$data$ – информация о цвете, соответствующая цветовой структуре c ;

$timestamp$ – временная метка, указывающая время когда токен помещается на место $p(1;3)$.

В раскрашенных условных сетях Петри места подразделяются на примитивные, виртуальные и композитные. Примитивные места отображают инициализирующие события. Виртуальное место используется для хранения составных событий. Композитные места – отображают события детекторы [20].

Визуализация мест приведена на рисунке 6.

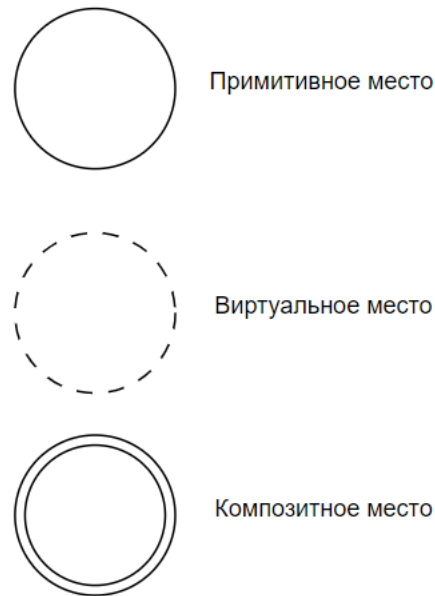


Рисунок 6 – Визуализация мест в раскрашенных условных сетях Петри

Переходы подразделяются на переходы копирования и композитные переходы. Композитные переходы используются для создания составных событий, а копирующие переходы используются для создания копий событий.

Визуализация переходов приведена на рисунке 7.

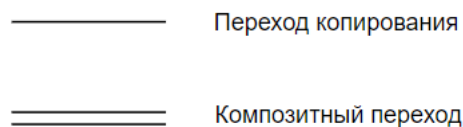


Рисунок 7 – Визуализация переходов в раскрашенных условных сетях Петри

Дуги подразделяются на нормальные дуги и дуги ингибитора. Дуги-ингибиторы используются для отрицания составных событий.

Визуализация дуг приведена на рисунке 8.

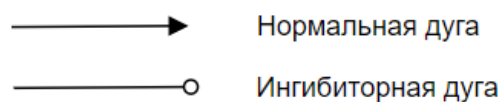


Рисунок 8 – Визуализация дуг в раскрашенных условных сетях Петри

2.3 Алгоритм развертывания раскрашенной условной сети Петри для моделирования сложного составного события

Изначально, сложное составное событие представляется в виде логического выражения. Для развертывания выражения в условную раскрашенную сеть Петри необходим алгоритм.

В рамках работы предлагается следующий алгоритм для развертывания условной раскрашенной сети Петри рисунок 9.

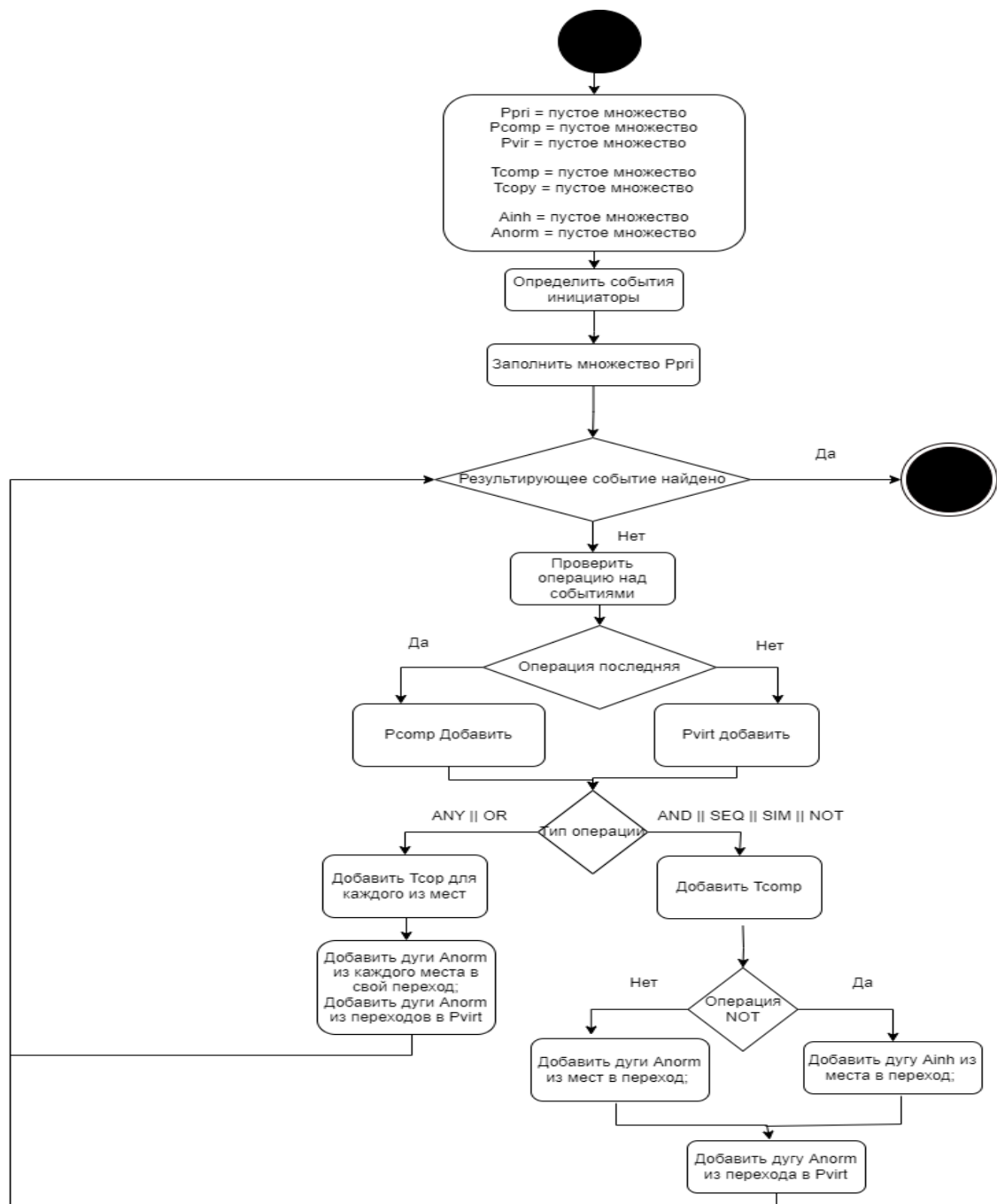


Рисунок 9 – Алгоритм развертывания условной раскрашенной сети Петри

На вход алгоритма подается логическое выражение. Множества мест, переходов и дуг инициализируются и становятся равными пустым множествам. Затем определяется множество примитивных инициализирующих мест P_{pri} . В цикле происходит определение операций между событиями. В зависимости от типа операции, происходит заполнение множеств переходов и дуг. Когда найдена последняя операция над событиями, в множество композитных мест добавляется результирующее место и алгоритм завершается.

2.4 Алгоритмы обнаружения сложных составных событий на раскрашенной условной сети Петри для различных режимов потребления

Перед описанием алгоритмов обнаружения сложных составных событий, необходимо ввести сущность «Лист ожидания», с его помощью определяется список ожидаемых событий из очереди, в зависимости от задаваемых параметров, будет определено поведения событий.

«Лист ожидания» представляет из себя список с таким наборов атрибутов как:

- наименование сети Петри;
- место;
- режим потребления.

Строки попадают в «Лист ожидание» при инициализации сети Петри и во время её работы.

Строки удаляются после того, как событие в списке было найдено.

Процесс обнаружение сложных составных событий можно разделить на 3 этапа:

- 1) маршрутизация входного события;
- 2) обработка события на раскрашенной условной сети Петри;
- 3) обработка событий в условиях различных режимов потребления.

Алгоритм маршрутизации входного события представлен на рисунке 10.

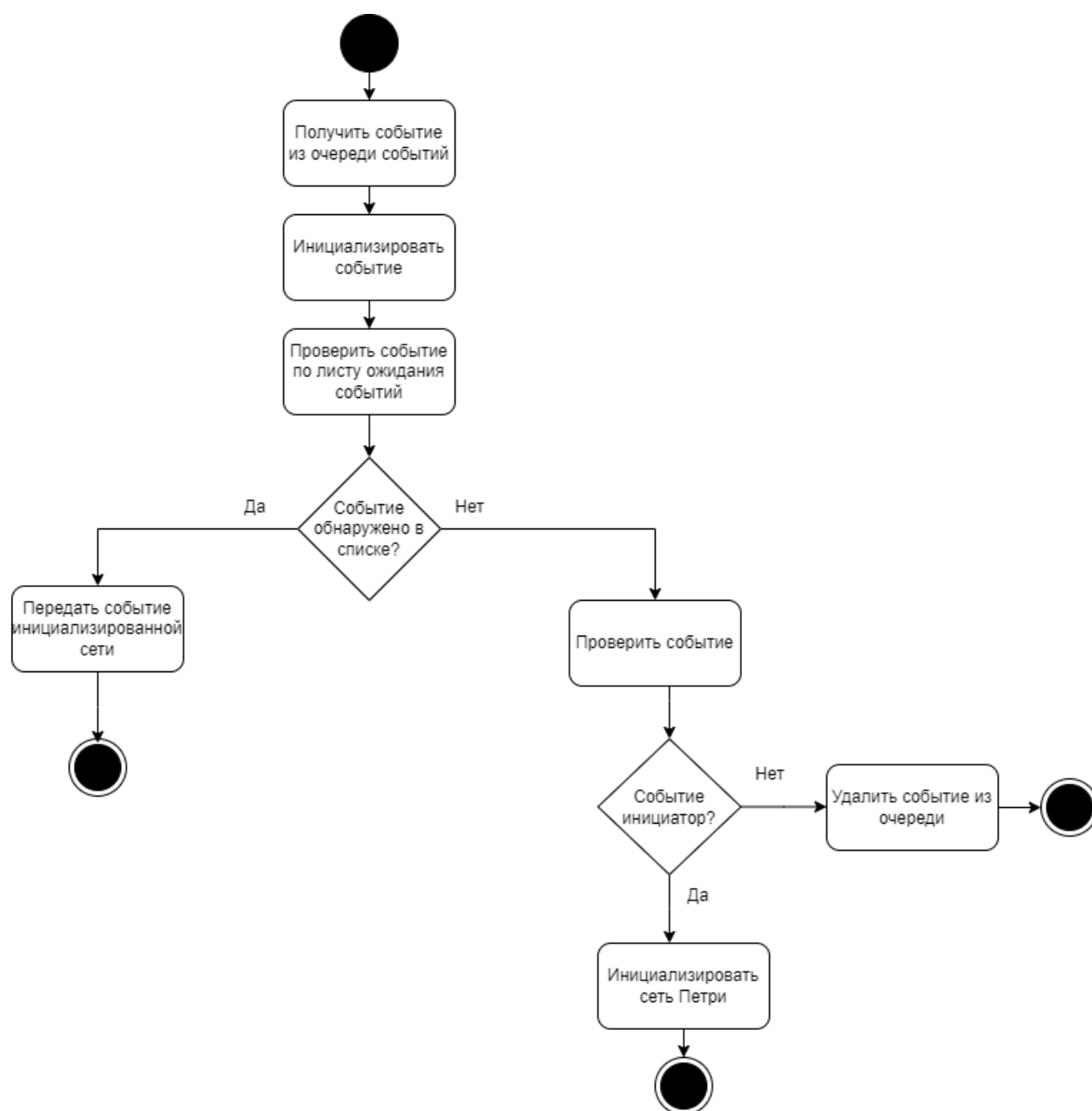


Рисунок 10 – Алгоритм маршрутизации событий

После того, как событие пришло из очереди событий, оно проверяется по листу ожидания событий. Если событие соответствует событию из листа ожиданий, оно передается в сеть Петри. Иначе, событие проверяется по репозиторию метаданных и если событие инициализирует сложное составное событие, то соответственно, инициализируется сеть Петри. Если событие не найдено в листе ожидания и событие не инициализирует никаких сложных составных событий, то событие удаляется из очереди.

Алгоритм обнаружения сложных составных событий по условной раскрашенной сети Петри представлен на рисунке 11.

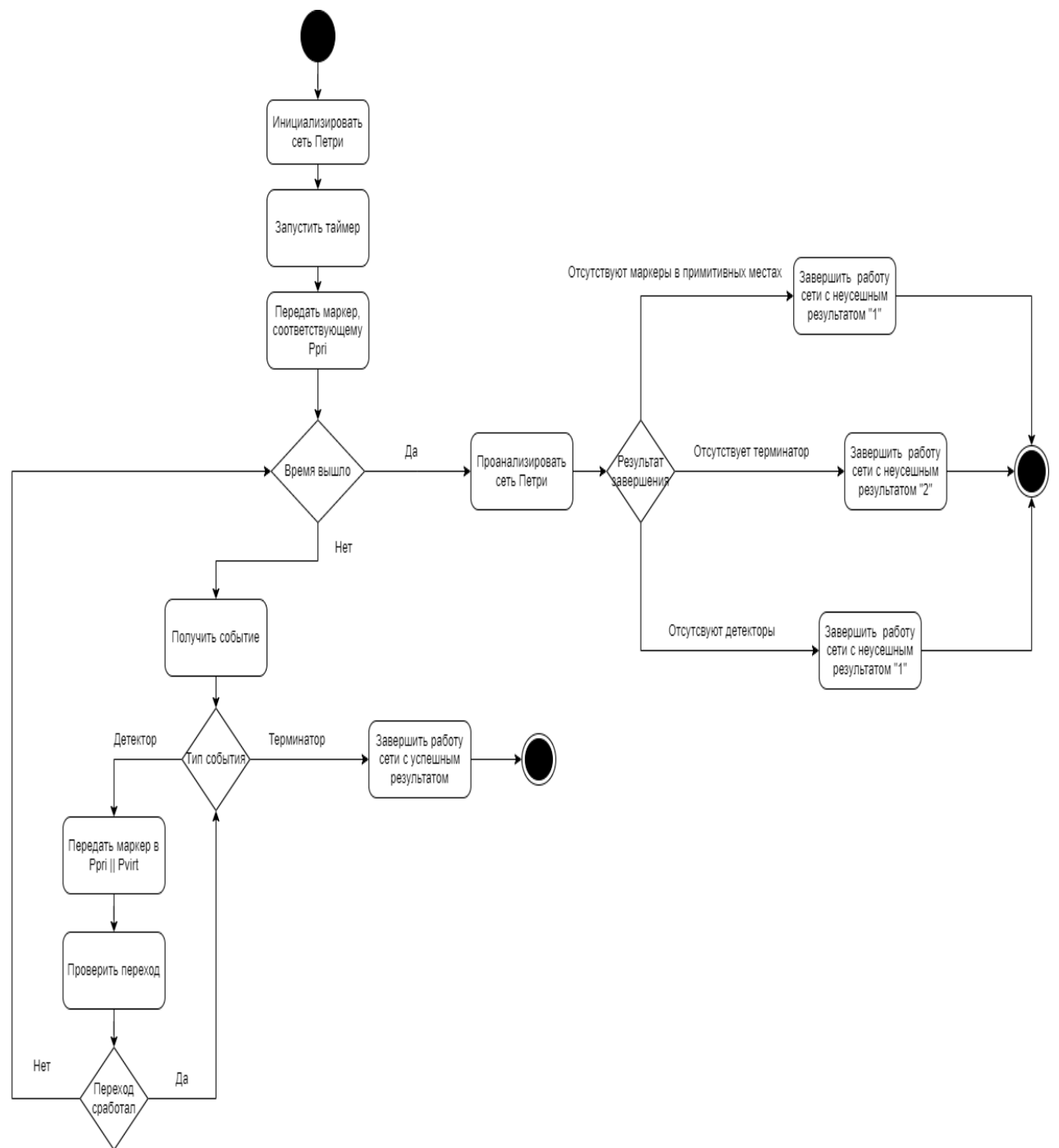


Рисунок 11 – Алгоритм обнаружения сложных составных событий по условной раскрашенной сети Петри

После того, как событие пришло из очереди событий, оно проверяется по листу ожидания событий. Если событие соответствует событию из листа

ожиданий, оно передается в сеть Петри. Иначе, событие проверяется по репозиторию метаданных и если событие инициализирует сложное составное событие, то соответственно, инициализируется сеть Петри. Если событие не найдено в листе ожидания и событие не инициализирует никаких сложных составных событий, то событие удаляется из очереди.

Обработка событий в условиях различных режимов потребления определяется для каждой операции над событиями.

Например, алгоритм для операции «AND» с условием режима потребления «Недавние» представлен на рисунке 12.

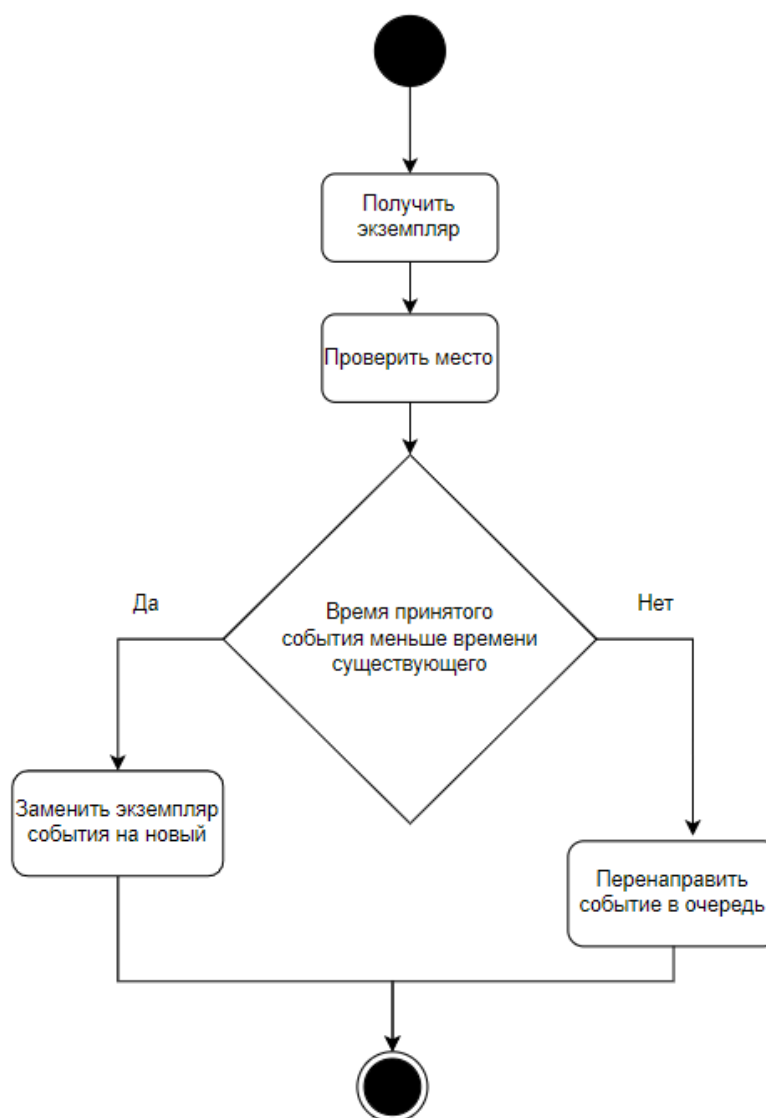


Рисунок 12 – Режим потребления «Недавние» для операции «AND»

В данном случае время экземпляра события сравнивается с временем пришедшего экземпляра, и в случае, если время нового экземпляра меньше, то происходит замена старого экземпляра.

Алгоритм для операции «SIM» с условием режима потребления «Недавние» представлен на рисунке 13.

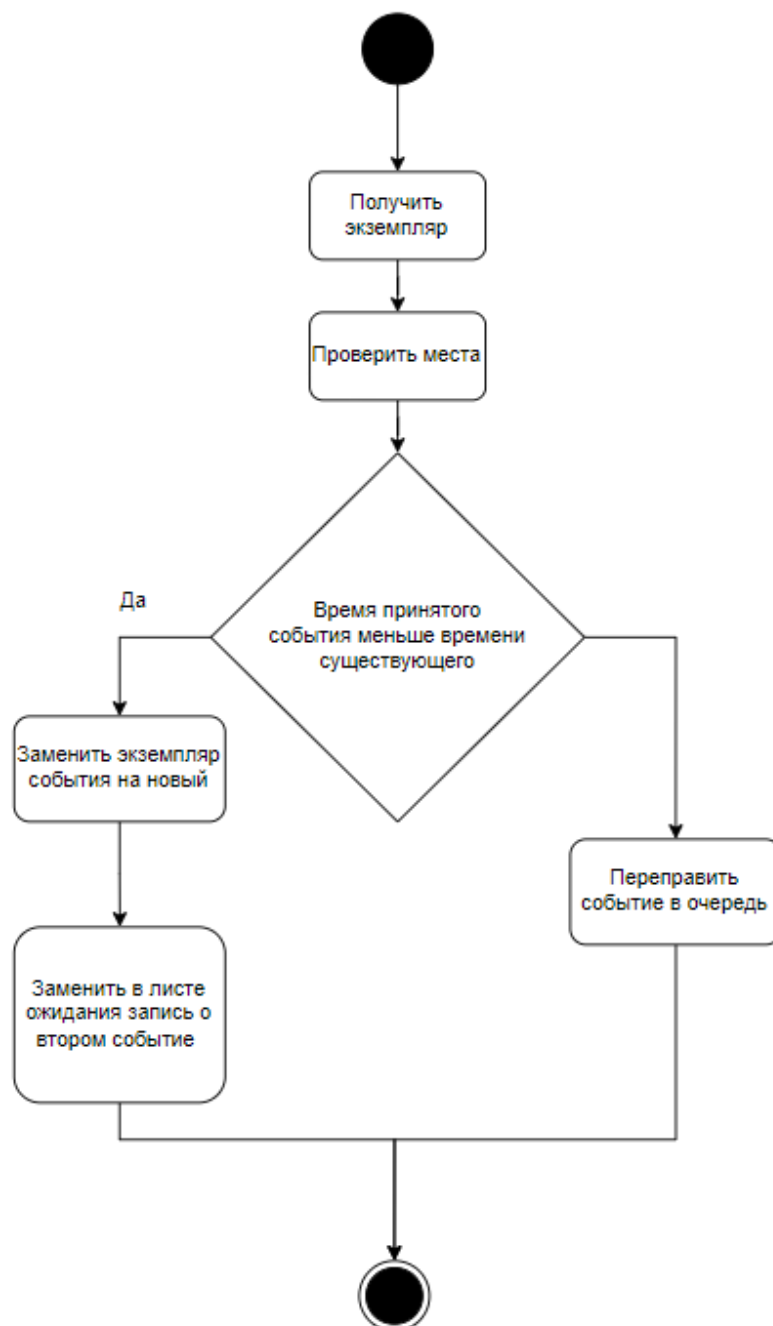


Рисунок 13 – Режим потребления «Недавние» для операции «SIM»

В отличие от операции «AND» при замене экземпляра на новый экземпляр, необходимо также изменить запись в листе ожидания второго события, таким образом, что время нового экземпляра было равным времени второго.

Алгоритм для операции «SEQ» с условием режима потребления «Недавние» представлен на рисунке 14.

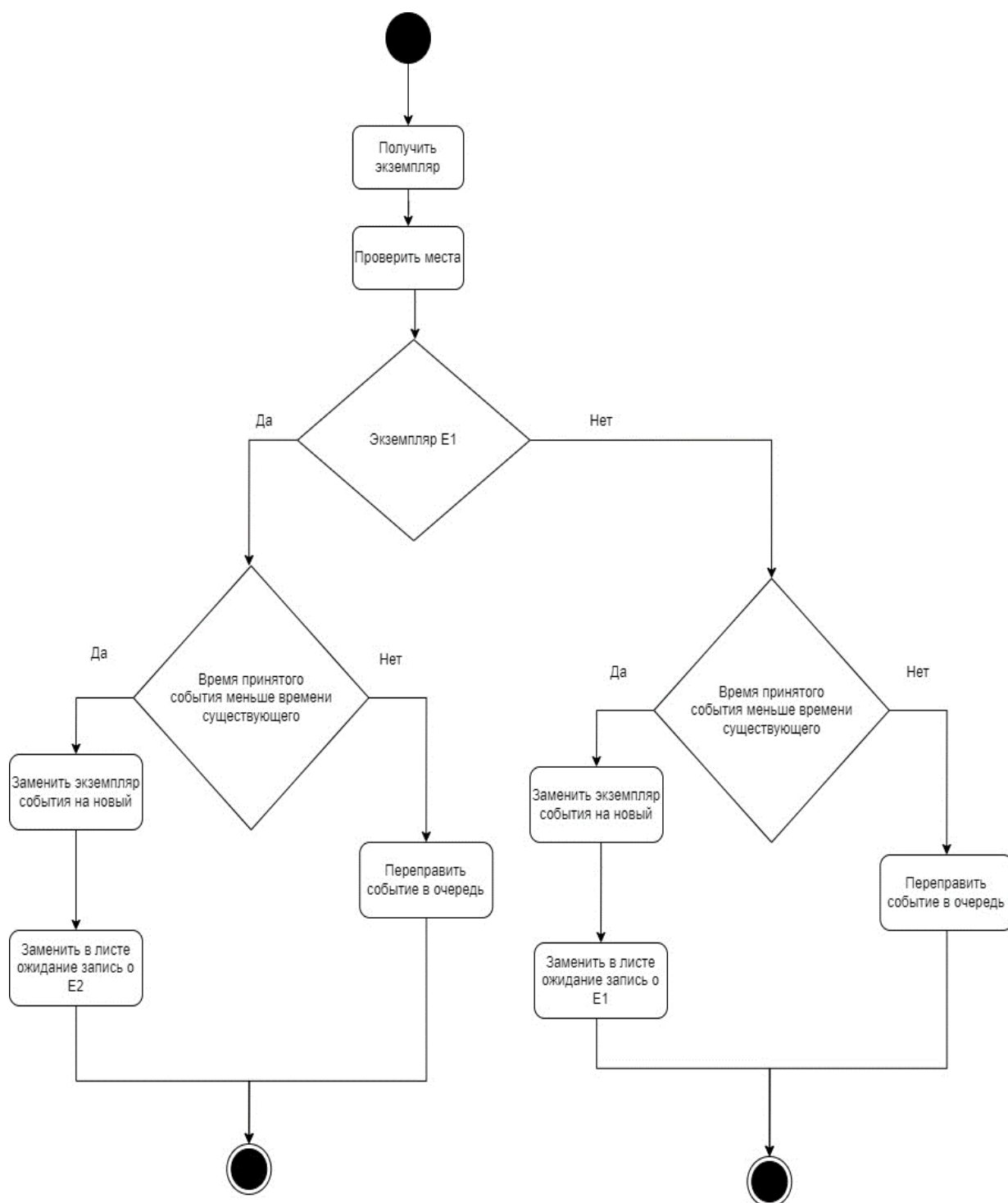


Рисунок 14 – Режим потребления «Недавние» для операции «SEQ»

Если на вход пришел экземпляр события E_1 , то при перезаписи новым экземпляром необходимо заменить в листе ожидания запись о E_2 с условием, что время $E_2 <$ времени E_1 . Если же на вход пришел экземпляр события E_2 , то необходимо в листе ожидания перезаписать E_1 с условием, что время $E_1 >$ времени E_2 .

Реализация режима потребления «Хроника» отличается от режима потребления «Недавние», условием проверки времени приходящего экземпляра. Если в режиме потребления «Недавние» осуществлялась замена экземпляра новым, то в режиме потребления «Хроника», экземпляр заменялся на более старым.

Обработка событий такими операциями как «OR» или «ANY» не изменяются с учетом режима потребления.

Выводы

В данной главе сложное составное событие было рассмотрено как асинхронный процесс. С помощью данной модели можно описать динамику поведения асинхронных систем, которые работают параллельно.

Для реализации алгоритмов обработки сложных составных событий, сложное составное событие было описано в рамках раскрашенной условной сети Петри.

На основании описания раскрашенной условной сети Петри был разработан алгоритм развертывания логического выражение в раскрашенную сеть Петри, а также группа алгоритмов для обнаружения сложного составного события.

Данная группа включает в себя алгоритм маршрутизации входного события, алгоритм обработки события на раскрашенной условной сети Петри, а также алгоритмы обработки событий в условиях различных режимов потребления;

Совокупность разработанных алгоритм, дает возможность для реализации микросервиса обнаружения сложных составных событий.

3 Программные средства обнаружения сложных составных событий в составе сервиса потоковой обработки событий и исполнения активных правил

3.1 Сервис потоковой обработки событий и исполнения активных правил

Сервис потоковой обработки событий и исполнения активных правил предназначен для реализации событийно-ориентированной обработки данных и событий, поступающих от наблюдаемых систем (баз данных, технических объектов), а также выявляемых в соответствии с заданными шаблонами, и реагирования на события в виде исполнения активных правил, воздействующих на управляемые системы (базы данных, технические объекты) [21].

Применительно к базам данных механизмы исполнения активных правил могут быть использованы для поддержания ограничений целостности, материализованных представлений, производных данных, координировании распределённых вычислений, поддержки транзакций, принятия решений по результатам изменений данных. Применительно к техническим объектам механизм исполнения активных правил могут быть использованы в приложениях управления и контроля техническими объектами, в медицинских приложениях для предупреждения врачей о состоянии пациента, в транспортных приложениях для прогноза пробок и в контроле воздушных сообщений для обнаружения потенциально опасных маршрутов и предупреждении диспетчера о сближении самолетов.

Основными функциями сервиса являются:

- конструирование и управление активными правилами – создание и редактирование правил и их компонент, проверка корректности правила, подключение и отключение правила;

- анализ активных правил – выявление и устранение коллизий исполнения активных правил (статический анализ на этапе конструирования и динамический анализ на этапе исполнения);
- обработка событий – обработка потока событий от наблюдаемых систем, первичная обработка входных событий и формирование событий на основе агрегации входных данных, выявление сложных событий на основе заданных шаблонов, доставка событий к соответствующим компонентам сервиса для их обработки;
- исполнение активных правил – выбор экземпляра активного правила (группы правил) для исполнения, вычисление условия экземпляра активного правила, принятие решение о выполнении действия экземпляра активного правила, применение действия к исполняемому объекту;
- журнализация и анализ данных журналов событий – журнализация обрабатываемых событий, исполнения активных правил, обработки компонент активных правил, выявление закономерностей в журналах событий, моделирование новых событий и активных правил на основе выявленных закономерностей;
- аудит обрабатываемых событий и активных правил в режимах онлайн (на основе актуальной информации) и офлайн (на основе данных журналов), выявление «проблемных» событий;
- настройка и масштабирования сервиса – настройка параметров функционирования отдельных компонент с учетом результатов аудита, масштабирование отдельных компонент (изменение параметров и распределение очередей и потоков обработки) и сервиса в целом (подключение/отключение компонентов), синхронизация глобальных и локальных метаданных, описывающих конфигурацию системы и отдельных компонент.

Архитектура сервиса потоковой обработки событий и исполнения активных правил представлена на рисунке 15.

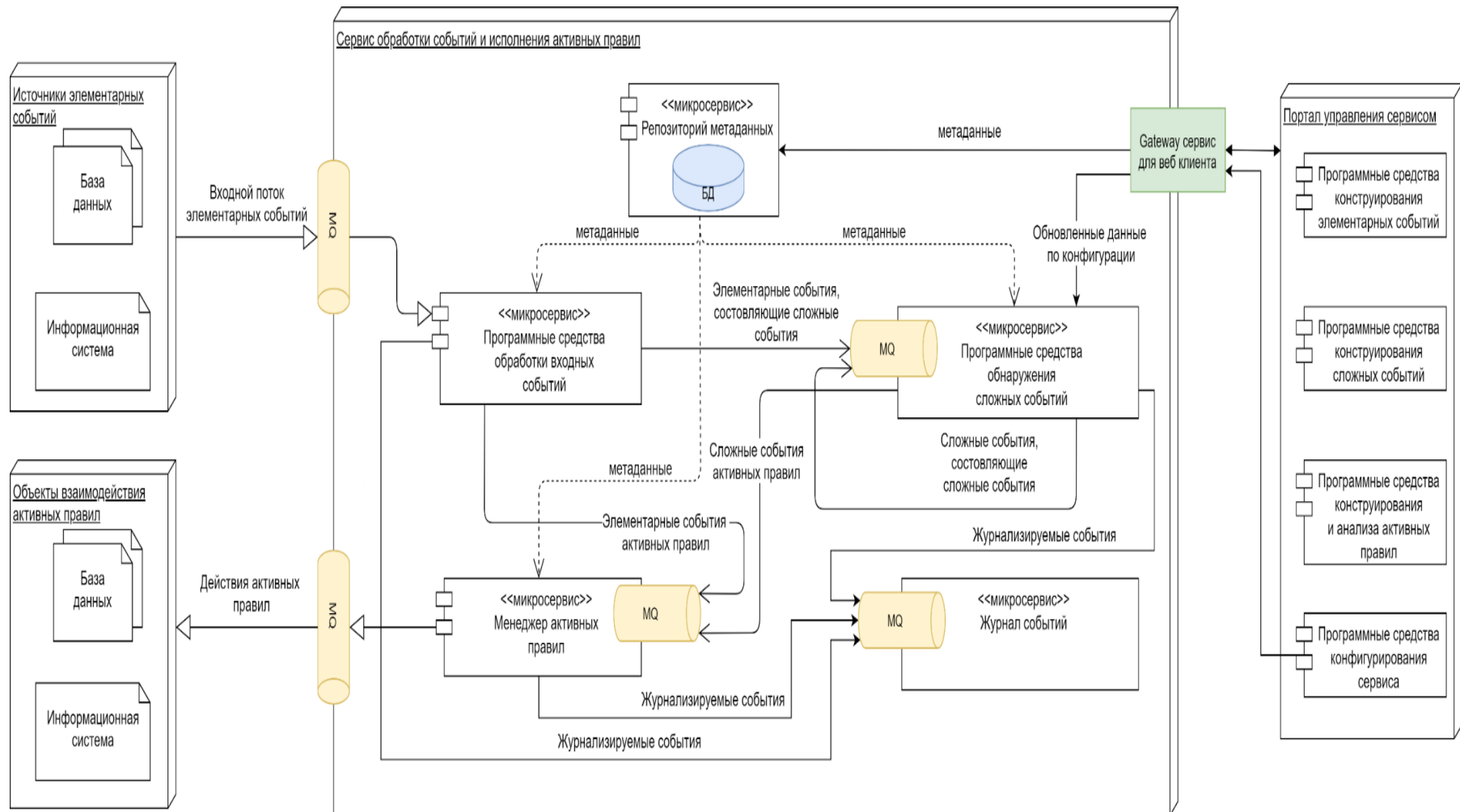


Рисунок 15 – Архитектура сервиса потоковой обработки событий и исполнения активных правил

Архитектура сервиса потоковой обработки событий и исполнения активных правил построена на принципах интеграции микросервисного и событийно-ориентированных подходов. Для доставки событий, поступающих от наблюдаемых систем, используется событийно-ориентированный подход на основе механизма очередей и платформы потоковой обработки данных Kafka с применением фреймворка Kafka Stream [22].

3.2 Микросервис обнаружения сложных составных событий в составе сервиса

3.2.1 Архитектура микросервиса обнаружения сложных составных событий

Микросервисная архитектура представляет собой архитектурный подход, который включает разделение больших приложений на более мелкие функциональные блоки, способные функционировать и обмениваться данными независимо друг от друга [23].

В рамках Сервиса потоковой обработки событий и исполнения активных правил, необходимо разработать микросервис отвечающий за обнаружение сложных составных событий.

Микросервис обнаружения сложных составных событий должен реализовывать следующие функции:

- принимать на вход события из очереди из очереди;
- реализовывать процесс работы условной раскрашенной сети Петри;
- определять соответствие входящего события сетям Петри;
- определять свершение сложного составного события с различными вариантами завершения;
- реализовывать работу сети Петри в условиях различных режимов потребления.

На основании данных требования была разработана следующая архитектура микросервиса (рисунок 16).

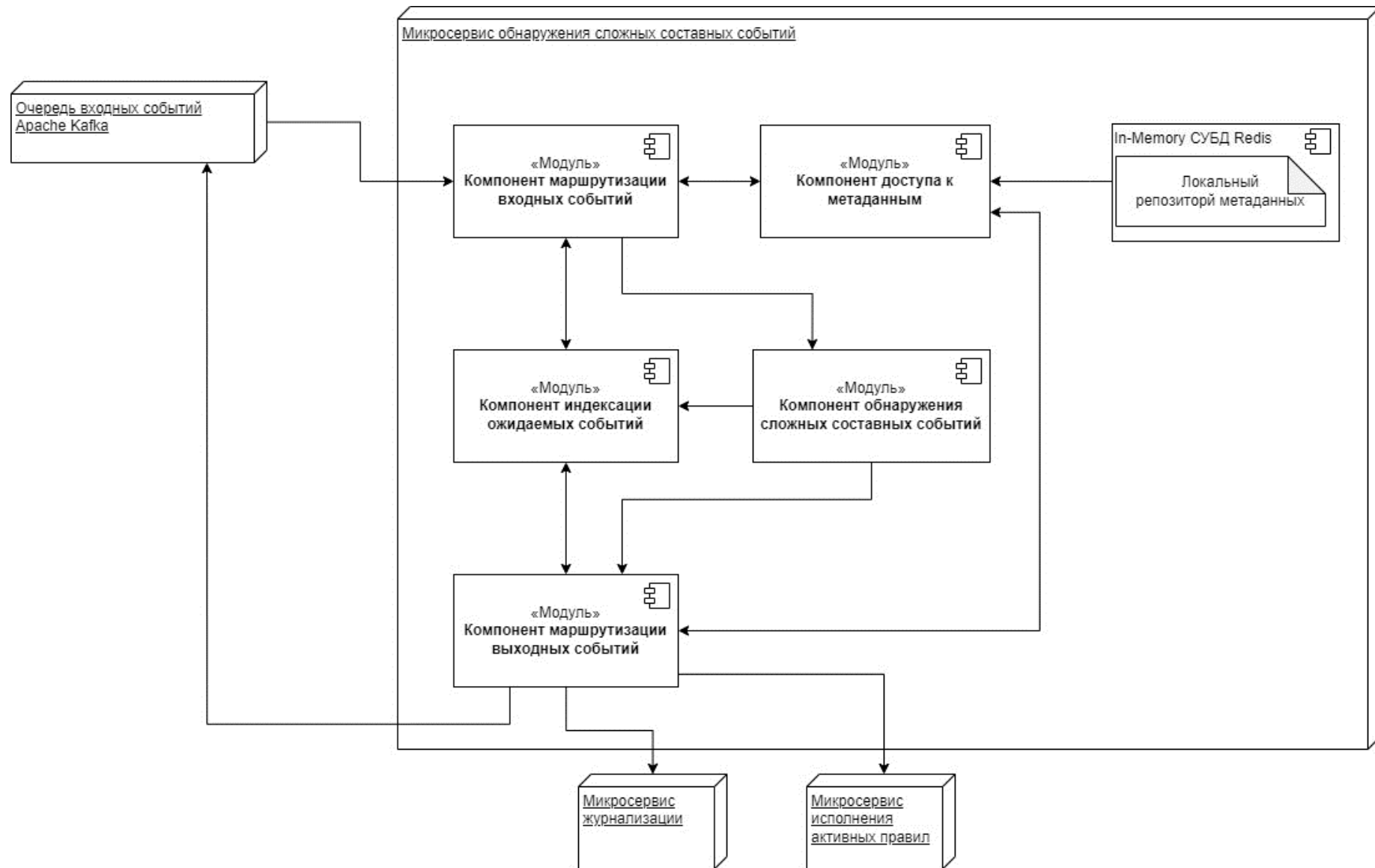


Рисунок 16 – Архитектура микросервиса обработки сложных составных событий

В локальном репозитории метаданных хранится информация о существующих видах сложных составных событиях, и событиях их инициализирующих.

Доступ к локальному репозиторию метаданных обеспечивает компонент доступа к метаданным.

Компонент маршрутизации входных событий предназначен для определения маршрута для входных событий. Маршрут для входных событий определяется исходя из данных полученных из локального репозитория метаданных и компонента индексации ожидаемых событий.

Компонент индексации ожидаемых событий хранит в себе лист ожидания и механизмы, осуществляющие управление данным листом.

Компонент обнаружения сложных составных событий представляет из себя процесс, реализующий работу раскрашенной условной сети Петри.

Компонент маршрутизации выходных событий опираясь на локальный репозиторий метаданных определяет маршрут для выходного события.

3.2.2 Принцип обработки событий в микросервисе обнаружения сложных составных событий

Адаптировав алгоритмы обработки событий под компоненты системы, можно получить возможность отобразить принципы работы микросервиса. Для отображения принципов работы используется диаграмма последовательности.

Диаграмма последовательности – это тип диаграммы, который позволяет описать взаимодействие между объектами в системе в виде последовательности сообщений, действий и операций, отображая порядок выполнения действий и обмена информацией между объектами во времени [24].

В качестве объектов в диаграмме выступают компоненты микросервиса.

Диаграмма последовательности для микросервиса обнаружения сложных составных событий приведена на рисунке 17.

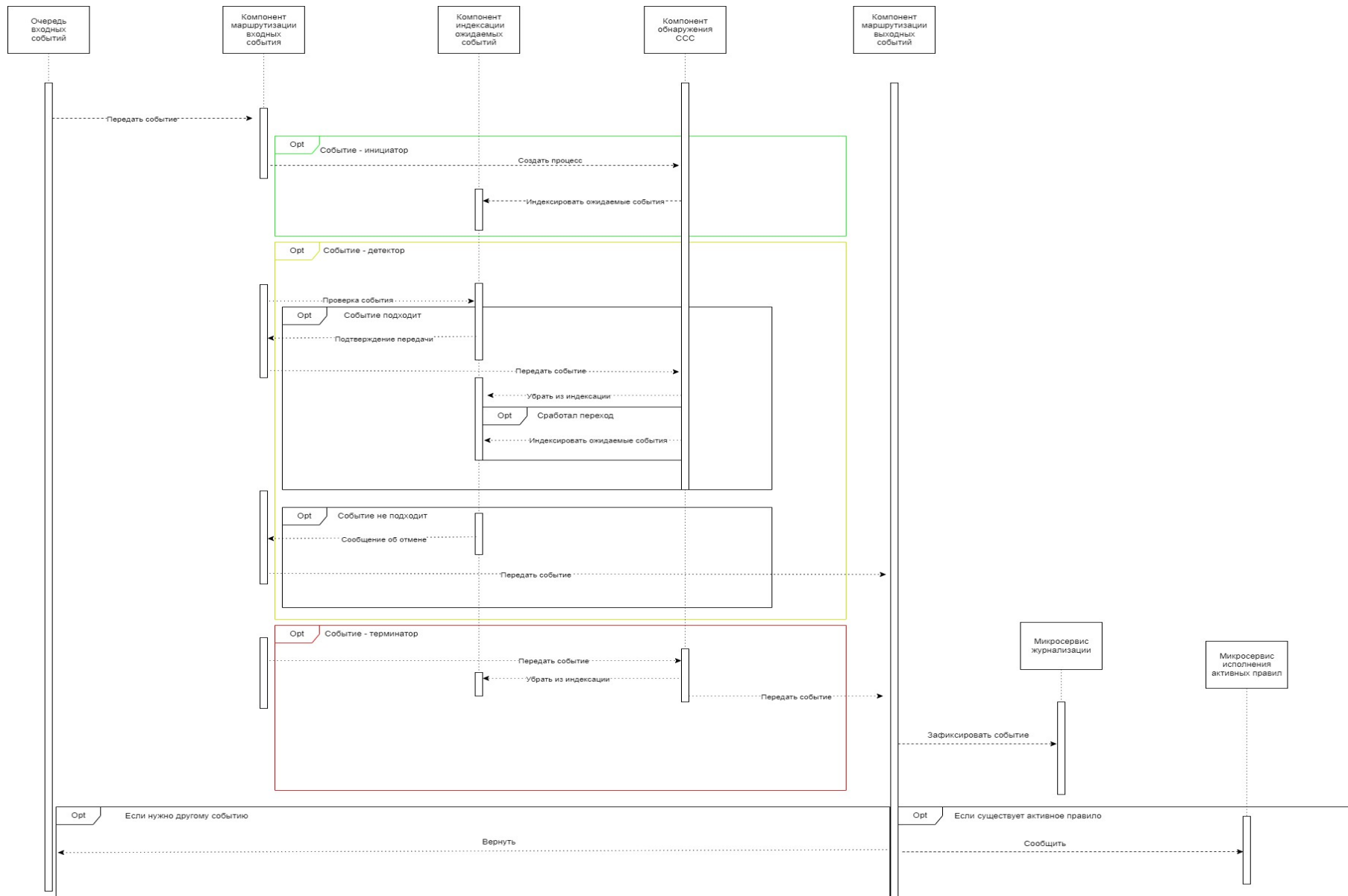


Рисунок 17 – Диаграмма последовательности для микросервиса обнаружения сложных составных событий

Очередь входных событий передает событие на вход компоненту маршрутизации входных событий. Если данное событие – инициатор, то инициализируется сеть Петри в компоненте обнаружения сложных событий. Сеть Петри в свою очередь индексирует необходимые для работы сети события в компоненте индексации ожидаемых событий.

Если событие – детектор, то событие проверяется в компоненте индексации ожидаемых событий. В случае, если событие ожидается в данном компоненте, оно передается компоненту обнаружения сложных составных событий и удаляется соответствующая запись из компонента индексации ожидаемых событий. Если при этом в сети Петри сработал переход, то записи в компоненте индексации ожидаемых событий обновляются.

Если событие детектор не ожидается списком, то, оно передается в компонент маршрутизации выходных событий.

Если событие – терминатор, то событие передается компоненту обнаружения сложных составных событий и соответствующая сеть Петри останавливает свою работу с определенным результатом. Записи связанные с данной сетью Петри удаляются из списка ожидаемых событий. Сложное составное событие считается обнаруженным.

После попадания событий в компонент маршрутизации, события журналируются, и, если существует активное правило, связанное с событием, активное правило выполняется.

3.2.3 Структура раскрашенной условной сети Петри в рамках микросервиса обнаружения сложных составных событий

Необходимо определить структуру раскрашенной условной сети Петри в рамках микросервиса обнаружения сложных составных событий.

С помощью диаграммы классов можно наиболее четко отобразить структуру раскрашенной условной сети Петри, моделируя ее классы, атрибуты, операции и отношения между объектами [25].

Диаграмма классов для структуры сети Петри представлена на рисунке 18.

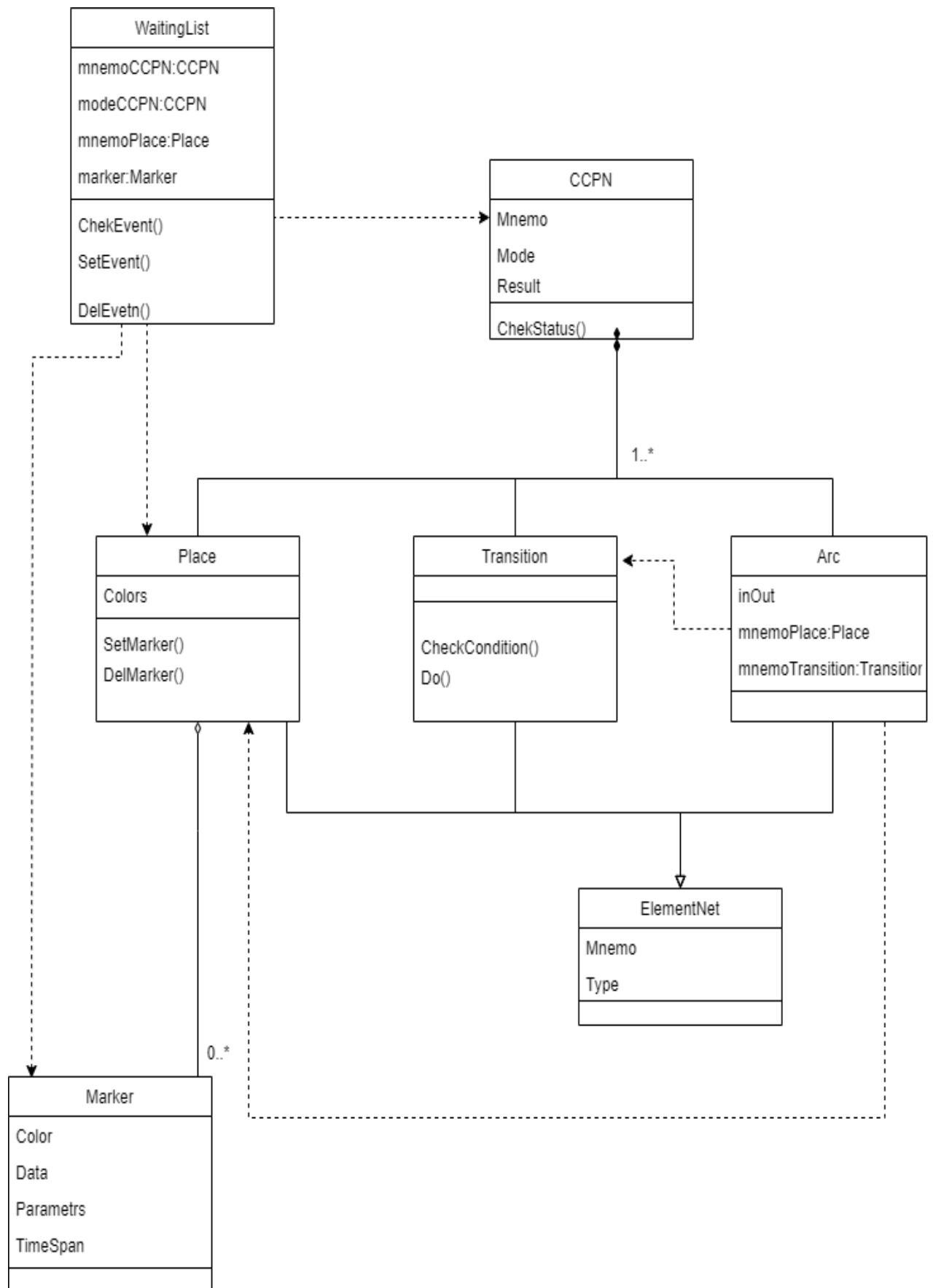


Рисунок 18 – Диаграмма классов для отображения структуры раскрашенной условной сети Петри

Класс «ElementNet» является базовым классом и служит для хранения информации об элементе сети Петри, т. е. «Мнемона» – уникальный строковый идентификатор и «Type» – тип элемента,

От класса «ElementNet» наследуются три класса: «Place», «Transition», «Arc».

Класс «Transition» содержит информацию о переходе и содержит в себе методы «Проверить условия перехода» и «Выполнить».

Класс «Arc» содержит информацию о дуге. Атрибут «inOut» определяет направление дуги. Класс использует объект «Place» и «Transition».

Класс «Place» обозначает место сети. Класс содержит такие атрибуты как: «Colors» – множество цветов, которые могут принимать маркеры. Также класс агрегирует в себе множество маркеров, содержащиеся на данном месте. Класс содержит методы добавления маркера в место и удаление маркера из места.

Класс «Marker» представляет информацию о маркере. Он содержит в себе такие атрибуты как: цвет, информацию о цвете, параметры, и время появления в месте.

Класс «CCPN» описывает раскрашенную условную сеть Петри. Данный класс связан отношением композиции с классами «Place», «Transition», «Arc». Класс содержит в себе атрибуты «Режим потребления» и «Результат», также класс имеет метод «Проверить состояние сети».

Класс «WaitingList» описывает лист ожидания для взаимодействия программы с сетью Петри. Класс содержит в себе атрибуты такие как: идентификатор сети Петри, режим потребления сети, место, где ожидается событие и ожидаемый маркер. Класс содержит методы добавления записи в список, удаление записи из списка, а также метод проверки пришедшего события.

Выводы

В данной главе были определены функции сервиса потоковой обработки событий и исполнения активных правил, а также архитектура сервиса. В рамках

данного сервиса, применяя разработанные алгоритмы обработки и обнаружения сложных составных событий, была разработана и описана архитектура микросервиса обработки сложных составных событий.

Также для данного микросервиса с помощью диаграммы последовательности были отображены принципы его работы.

Адаптировав модельное представление условных раскрашенных сетей Петри, была определена её структура в рамках микросервиса обработки сложных составных событий.

Заключение

Существующие решения проблемы обработки сложных составных событий в информационных системах имеют ряд ограничений. В данной работе предлагается использование единого модельного представления сложных составных событий на основе условных раскрашенных сетей Петри.

В ходе работы были решены следующие задачи:

- 1) исследованы и проанализированы проблемы обнаружения сложных составных событий в системах потоковой обработки событий;
- 2) исследованы структуры сложных составных событий и обоснован выбор модельного представления сложных составных событий;
- 3) разработаны алгоритмы отображения сложных составных событий в модельное представление на основе условных раскрашенных сетей Петри;
- 4) разработаны алгоритмы обнаружения сложных составных событий с учетом сценариев обработки входящих событий;
- 5) разработана архитектура программных средств обнаружения сложных составных событий в составе сервиса потоковой обработки событий и исполнение активных правил.

В ходе работы были исследованы и проанализированы особенности обработки сложных составных событий и их интеграция с потоковой обработкой и механизмом активных правил.

В ходе работы была проведена формализация сложных составных событий, основанных на алгебре событий, что позволяет описывать сложные составные события без учета предметной области. Также отмечается необходимость разработки унифицированной модели представления сложных составных событий, которая может быть использована в процессе отслеживания сложных составных событий. В качестве такой модели предлагается использовать условные раскрашенные сетки Петри.

В работе были разработаны алгоритмы развертывания сложных составных событий в раскрашенную условную сеть Петри, а также алгоритмы обнаружения сложных составных событий с учетом режимов потребления.

Также результатами работы является разработанная архитектура микросервиса и описание принципов его работы.

Результаты исследования позволяют реализовывать системы потоковой обработки сложных составных событий, обеспечивающие надежность обнаружения.

Список использованных источников

1. Petite histoire de la simulation des flux // Optiflux Conseils d'expert [Электронный ресурс]. URL: <https://www.optiflux-sas.com/petite-histoire-de-la-simulation-des-flux/> (дата обращения: 20.04.2022)
2. Сравнение ESB-решений, популярных на российском рынке // kt-team [Электронный ресурс]. URL: <https://kt-team.ru/blog/comparing-popular-esb-systems> (дата обращения: 20.04.2021)
3. What is service-oriented architecture (SOA)? // IBM [Электронный ресурс]. URL: <https://www.ibm.com/cloud/learn/soa> (дата обращения: 20.04.2021)
4. Event Driven Architecture // Pradeep Loganathan Blogs [Электронный ресурс]. URL: <https://pradeepl.com/blog/architecture/event-driven-architecture/> (дата обращения: 20.04.2021)
5. What is business process management? // IBM [Электронный ресурс]. URL: <https://www.ibm.com/cloud/learn/business-process-management> (дата обращения: 20.04.2021)
6. Olups, R. Zabbix Network Monitoring - Second Edition / R. Olups – Packt Publishing, 2016. – 754 с.
7. David C. Luckham. Event Processing for Business: Organizing the Real-Time Enterprise. — John Wiley & Sons, 2011. – 288 с.
8. Complex Event Processing: An Introduction // Confluent [Электронный ресурс]. URL: <https://www.confluent.io/learn/complex-event-processing/> (дата обращения: 20.04.2021)
9. Active Complex Event Processing: Applications in Real-Time Health Care // [Электронный ресурс]. URL: <https://www.cse.iitb.ac.in/ActiveCEP.pdf> (дата обращения: 20.04.2021)
10. Mani, K. Chandy Event-Driven Applications: Costs, Benefits and Design Approaches. / K. Mani // In Proceedings of the Second International Workshop on Rules in Database Systems. – 2006. – p. 69-85.

11. Gatziau, S. Integrating active concepts object-oriented database system / S. Gatziau, A. Geppert, K. Dittrich. – Zürich: Institut für Informatik, 1991. – 97 p.
12. Stream Data Processing: A Quality of Service Perspective. Modeling, Scheduling, Load Shedding, and Complex Event Processing// [Электронный ресурс]. URL: https://www.cse.iitb.ac.in/Stream_Data_Processing.pdf (дата обращения: 20.04.2021)
13. Chakravarthy S. Composite Events for Active Databases: Semantics, Contexts and Detection / S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim // Proceedings, International Conference on Very Large Data Bases. – San Francisco: Morgan Kaufmann Publisher, 1994. – p. 606 – 617.
14. Котов, В. Е. Сети Петри / В. Е. Котов. – М.: Наука, 1984. – 158 с.: ил.; 22 см. – Библиогр.: с. 150-152.
15. Питерсон, Дж. Теория сетей Петри и моделирование систем / Питерсон Джеймс; перевод с англ. М. В. Горбатовой и др.; под общ. ред. д. т. н. В. А. Горбатова. – М.: Мир, 1984. – 264 с.: ил.; 22 см. - Библиогр.: с. 234-261. – Предм. указ.: с. 262-263. – Перевод изд.: Petri net theory and the modeling of systems / James L. Peterson . - Prentice-Hall, 1981. – с. 262-263.
16. When to Trigger Active Rules? // [Электронный ресурс]. URL: https://www.cse.iitb.ac.in/2009/proceedings/When_to_Trigger_Active_Rules.pdf (дата обращения: 20.04.2021)
17. Simulation Modeling with Event Graphs // [Электронный ресурс]. URL: https://www.cse.iitb.ac.in/Simulation_Modeling.pdf (дата обращения: 20.04.2021)
18. Representing Extended Finite State Machines for SDL by A Novel Control Model of Discrete Event Systems // [Электронный ресурс]. URL: https://www.cse.iitb.ac.in/Peng_Wang_Kai-Yuan_Cai.pdf (дата обращения: 20.04.2021)
19. Active Rule Based Complex Event Processing For Emergency Management System // [Электронный ресурс]. URL: <https://www.cse.iitb.ac.in/IJSDR1604054.pdf> (дата обращения: 20.04.2021)

20. Composite Event Detection as a Generic Middleware Extension // [Электронный ресурс]. URL: https://www.doc.ic.ac.uk/IEEE_Network.pdf (дата обращения: 20.04.2021)
21. Buchmann Alejandro P. Building an integrated active OODBMS: Requirements, architecture, and design decisions / Alejandro P Buchmann, Jürgen Zimmermann, José A Blakeley, David L Wells // Proceedings of the Eleventh International Conference on. IEEE, 1995. – pp. 117 – 128.
22. С. В. Шибанов, Я. С. Шлепнёв. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнёв // Математическое моделирование и суперкомпьютерные технологии. 2021. – С. 403–407.
23. Xiaoou Li. Fifth Composite Event Specification in Active Database Systems: A Petri Nets Approach / Xiaoou Li, Joselito Medina Marin // Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. – Denmark: Aarhus, 2004. – pp.97 – 117.
24. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. 2-е изд. – Пер. с англ. – СПб. : Символ-Плюс, 2007. – 624 с.
25. MSDN – сеть разработчиков Microsoft [Электронный ресурс]. – URL: <https://msdn.microsoft.com/ru-ru/dn308572.aspx> (дата обращения: 28.04.2018)

Приложение А

ГЛОССАРИЙ **(рекомендуемое)**

Примитивное событие: любой факт или действие, которое происходит с поименованным объектом в определенный интервал времени (идентифицируется объектом и временем).

Сложное событие: вычисляемые события, обнаруживаемые системой в результате обработки элементарных событий или других сложных событий.

Сложное агрегированное событие: события задается путем агрегации событий, полученных путем запроса к потоку событий.

Сложное составное событие: события задаются при помощи алгебры событий и определяются через временные отношения между события.

Обработка сложных событий: это подход к анализу событий в реальном времени, с выявлением их взаимосвязей, причин и следствий.

ЕСА: нотация описания активных правил, в соответствии с которой правилом является тройка объектов – событие предметной области, условие, предназначенное для проверки параметров события, и действие, описывающее операции, совершаемые в случае истинности условия. Событие и условие в контексте активных правил иногда называют Ситуацией.

Активная база данных: база данных, в которой хранятся как сами данные, при действиях с которыми могут возникать события, так и полное описание реакции на них.

Активное правило: объект специального вида, задающий реакцию на событие.

Сеть Петри: математический объект, используемый для моделирования динамических дискретных систем.

Приложение Б
КОД ПРОГРАММЫ
(рекомендуемое)

Листинг Б.1 – Класс CCPN

```
public class CCPN
{
    public string? Name { get; set; }
    public string? Mode { get; set; }
    public List<Place>? Places { get; set; }
    public List<Transition>? Transitions { get; set; }
    public List<Arc>? Arcs { get; set; }
}
```

Листинг Б.2 – Класс Place

```
public class Place:ElementNet
{
    public string[]? Colors { get; set; }

    [JsonIgnore]
    public List<Marker>? Markers { get; set; }
}
```

Листинг Б.3 – Класс ElementNet

```
public abstract class ElementNet
{
    public string? Name { get; set; }
    public string? Type { get; set; }
}
```

Листинг Б.4 – Класс Arc

```
public class Arc:ElementNet
{
    public string? inOut { get; set; }
    public string? place { get; set; }
    public string? transition { get; set; }
}
```

Листинг Б.5 – Класс Marker

```
public class Marker
{
    public string? Color { get; set; }
    public string? Data { get; set; }

    public List<string>? Parametrs { get; set; }
    public DateTime? TimeSpan { get; set; }
}
```

Листинг Б.6 – Интерфейс ICCPNService

```
public interface ICCPNService
{
    List<CCPN> Get();
}
```

```

        CCPN Get(string name);
        CCPN CreateInDb(CCPN ccpn);

        List<CCPNAndInitializingColors> GetInitializingColors();

        Marker SetMarker(Marker marker);
    }

```

Листинг Б.7 – Класс CCPNService

```

public class CCPNService : ICCPNService
{
    List<CCPN> _ccpnList ;
    List<CCPN> _workingCCPNList;
    List<CCPNAndInitializingColors> _initializingColors;
    SynchronizedCollection<Marker> _waitingCollection;

    public CCPNService()
    {
        _initializingColors = GetInitializingColors();
        _ccpnList = new List<CCPN>();
        _waitingCollection = new
        SynchronizedCollection<Marker>();
    }

    public CCPN CreateInDb(CCPN ccpn)
    {
        _ccpnList.Add(ccpn);

        var json = JsonConvert.SerializeObject(ccpn);

        var path = Path.Combine("D:\\Desktop\\Всё по
        Диплому\\Программа\\CCPNs", ccpn.Name + ".json");
        System.IO.File.WriteAllText(path, json);
        return ccpn;
    }

    public List<CCPN> Get()
    {
        var path = Path.Combine("D:\\ CCPNs");
        string[] files = Directory.GetFiles(path);
        foreach (var f in files)
        {
            try
            {
                var ccpn =
                JsonConvert.DeserializeObject<CCPN>(File.ReadAllText(f));
                _ccpnList.Add(ccpn);
            }
            catch
            {
                return null;
            }
        }
    }
}

```

```

        }
    }
    return _ccpnList;
}

public CCPN Get(string name)
{
    var path = Path.Combine("D:\\ CCPNs", name + ".json");
    try
    {
        string buff = File.ReadAllText(path);
        var ccpn =
JsonConvert.DeserializeObject<CCPN>(buff);
        return ccpn;
    }
    catch
    {
        return null;
    }
}

public List<CCPNAndInitializingColors>
GetInitializingColors()
{
    var listCCPNAndInitializingColors = new
List<CCPNAndInitializingColors>();
    var path = Path.Combine("D:\\Desktop\\Всё по
Диплому\\Программа\\CCPNs");
    string[] files = Directory.GetFiles(path);
    foreach (var f in files)
    {
        try
        {
            CCPN ccpn =
JsonConvert.DeserializeObject<CCPN>(File.ReadAllText(f));

            foreach (var place in ccpn.Places)
            {
                if (place.Type == "pri")
                {
                    foreach( var color in place.Colors)

listCCPNAndInitializingColors.Add(new
CCPNAndInitializingColors(ccpn.Name, color));
                }
            }
        }
        catch
        {
            return null;
        }
    }
}

```

```

        return listCCPNAndInitializingColors;
    }

    public SetMarker(Marker marker)
    {
        foreach(var i in _initializingColors)
        {
            if (marker.Color == i.color)
            {
                _workingCCPNList.Add(new CCPN());
            }
        }
        return marker;
    }
}

public struct CCPNAndInitializingColors
{
    public string? ccpnName;
    public string? color;
    public string? place;

    public CCPNAndInitializingColors(string ccpnName, string
color)
    {
        this.ccpnName = ccpnName;
        this.color = color;
    }
}

```

Листинг Б.7 – Класс CCPNController

```

[ApiController]
[Route("[controller]")]
public class CCPNController : Controller
{
    private readonly CCPNService ccpnService;
    public CCPNController(CCPNService ccpnService)
    {
        this.ccpnService = ccpnService;
    }

    [HttpPost]
    public ActionResult<CCPN> Post([FromBody] CCPN ccpn)
    {
        ccpnService.CreateInDb(ccpn);

        return CreatedAtAction(nameof(Get), new { name =
ccpn.Name }, ccpn);
    }

    [HttpGet("{name}")]

```

```

public ActionResult<CCPN> Get(string name)
{
    var ccpn = ccpnService.Get(name);
    if (ccpn == null)
    {
        return NotFound($"CCPN with name = {name} not
found");
    }
    return ccpn;
}

[HttpPost]
public ActionResult<Marker> Post([FromBody] Marker marker)
{
    ccpnService.SetMarker(marker);
    return marker;
}

[HttpGet]
public ActionResult<List<CCPN>> Get()
{
    return ccpnService.Get();
}
}

```