Ямашкин Станислав Анатольевич
кандидат технических наук, доцент
Мордовский государственный университет имени Н.П. Огарёва
Мишкин Егор Евгеньевич
магистр
Мордовский государственный университет имени Н.П. Огарёва

ПРИМЕНЕНИЕ КОНЕЧНОГО АВТОМАТА И ПАТТЕРНА SAGA ДЛЯ ОРКЕСТРАЦИИ LLM-АГЕНТОВ

Аннотация: В статье рассматривается применение конечного автомата в сочетании с паттерном SAGA для оркестрации долгоживущих процессов LLM-агентов. Предлагается подход на основе конечного автомата, реализованного с использованием библиотеки MassTransit на платформе .NET, который обеспечивает формальное описание логики переходов, надёжное хранение состояния и встроенный механизм компенсирующих действий. Приводится диаграмма состояний оркестратора и листинги программной реализации, демонстрирующие управление итеративным процессом рассуждений и действий LLM-агента. Особое внимание уделяется переходу из состояния компенсации обратно в состояние рассуждения, что гарантирует устойчивость системы и непрерывность выполнения задач при возникновении ошибок.

Ключевые слова: LLM-агенты, конечный автомат, паттерн SAGA, оркестрация, распределённые системы, управление состоянием, компенсирующие операции.

Keywords: LLM agents, finite state machine, SAGA pattern, orchestration, distributed systems, state management, compensating operations.

Актуальность исследования.

Современные системы на основе больших языковых моделей всё чаще строятся как распределённые приложения, объединяющие множество взаимодействующих компонентов: сервисы оркестрации, исполнители LLM-рассуждений, специализированные воркеры для различных парадигм агентов, а также внешние инструменты и API. В таких системах основным способом взаимодействия между компонентами выступает асинхронный обмен сообщениями через брокеры сообщений [1]. При реализации многошаговых агентных процессов, затрагивающих как изменение состояния задачи, так и публикацию событий о переходе между этапами, возникает вопрос атомарности и

надёжности этих операций. Отсутствие гарантий сохранения состояния и согласованности действий в долгоживущих процессах может приводить к потере контекста выполнения, невозможности восстановления после сбоев, нарушению целостности данных и значительным сложностям в отладке распределённых агентных приложений.

В связи с этим представляется актуальным исследование и описание применения конечных автоматов и паттерна SAGA для оркестрации LLM-агентов, обеспечивающих надёжное хранение состояния долгоживущих процессов и выполнение компенсирующих действий при возникновении ошибок.

Постановка проблемы исследования.

Одной из ключевых проблем в распределённых микросервисных системах является обеспечение согласованности между изменением состояния данных и публикацией событий в брокер сообщений. В контексте LLM-агентных систем эта проблема проявляется особенно остро, поскольку каждый шаг долгоживущего процесса включает не только обновление контекста выполнения, но и инициирование асинхронных событий, направляемых в различные компоненты системы [3]. При сбое на этапе публикации события после успешного сохранения состояния система теряет возможность продолжить выполнение задачи, а при обратном сценарии возникает риск дублирования операций или несогласованности между компонентами. В долгоживущих агентных процессах, где количество итераций может достигать десятков, а каждая операция связана с обращением к внешним API и изменением данных, подобные нарушения согласованности приводят к критическим последствиям: потерянному контексту, невозможности восстановления после сбоев и необходимости ручного вмешательства [4].

Таким образом, возникает необходимость в разработке подхода, который обеспечивал бы атомарное обновление состояния и публикацию событий в долгоживущих агентных процессах, а также предоставлял встроенные механизмы для выполнения компенсирующих действий при возникновении ошибок. Целью настоящего исследования является анализ существующих методов решения данной проблемы и описание применения конечных автоматов в сочетании с паттерном SAGA как эффективного способа оркестрации LLM-агентов.

Обзор научной литературы.

Вопросам обеспечения согласованности и атомарности событий в распределённых системах посвящено значительное количество научных работ. Многие авторы подчёркивают проблему обеспечения атомарности между изменением состояния и публикацией событий как одну из наиболее острых при построении событийно-ориентированных архитектур, особенно в контексте долгоживущих бизнес-процессов [5].

В работах, посвящённых распределённым транзакциям, рассматриваются различные подходы к решению данной проблемы. Среди основных способов выделяются: прямая публикация события после коммита транзакции, использование двухфазного коммита, а также паттерн Saga. Распределённые транзакции и двухфазный коммит обеспечивают строгую согласованность, однако они значительно усложняют архитектуру системы, снижают производительность и доступность, особенно в условиях горизонтального масштабирования, что делает их неприменимыми в высоконагруженных распределённых системах.

Отдельное место среди решений занимает паттерн Saga. Данный паттерн предлагает подход к управлению долгоживущими транзакциями путём разбиения их на последовательность локальных транзакций, каждая из которых сопровождается компенсирующей операцией для отката в случае ошибки [6; 7]. В литературе выделяются две основные реализации Saga: хореография, где каждый сервис самостоятельно публикует события и реагирует на них, и оркестрация, где центральный координатор управляет последовательностью выполнения шагов и компенсаций. Исследования показывают, что оркестрируемый подход обеспечивает лучшую управляемость и наблюдаемость процесса, что особенно важно для сложных сценариев с множеством участников.

В контексте управления состоянием долгоживущих процессов значительный интерес представляют работы, посвящённые применению конечных автоматов [8]. Конечные автоматы позволяют формально описать все возможные состояния процесса, допустимые переходы между ними и события, инициирующие эти переходы. Использование конечных автоматов обеспечивает предсказуемость поведения, упрощает отладку и тестирование, а также позволяет выявлять недопустимые состояния на этапе проектирования.

Большинство фреймворков для работы с многоагентными системами предлагает собственные механизмы управления состоянием, но они не всегда обеспечивают строгих гарантий атомарности и не предоставляют встроенных средств для реализации компенсирующих операций. Таким образом, если в системе требуются дополнительные процессы валидации и контроля надёжности агентов, а также необходимость отката при обнаружении ошибок, идеальным решением становится использование конечного автомата в сочетании с паттерном Saga, что позволяет формализовать логику переходов между этапами выполнения и обеспечить согласованность данных через компенсирующие операции.

Материалы и методы.

Для решения задачи предлагается построить прототип оркестратора LLM-агента на основе конечного автомата с использованием паттерна SAGA. В качестве платформы

реализации выбрана экосистема .NET с библиотекой MassTransit. Переходы между состояниями инициируются сообщениями, поступающими через брокер Kafka от компонентов системы: сервиса LLM-рассуждений, воркеров выполнения инструментов и сервиса валидации.

Необходимо определить все возможные состояния процесса выполнения задачи LLM-агентом, а также события, инициирующие переходы между ними. На рисунке 1 представлена диаграмма состояний оркестратора LLM-агента.

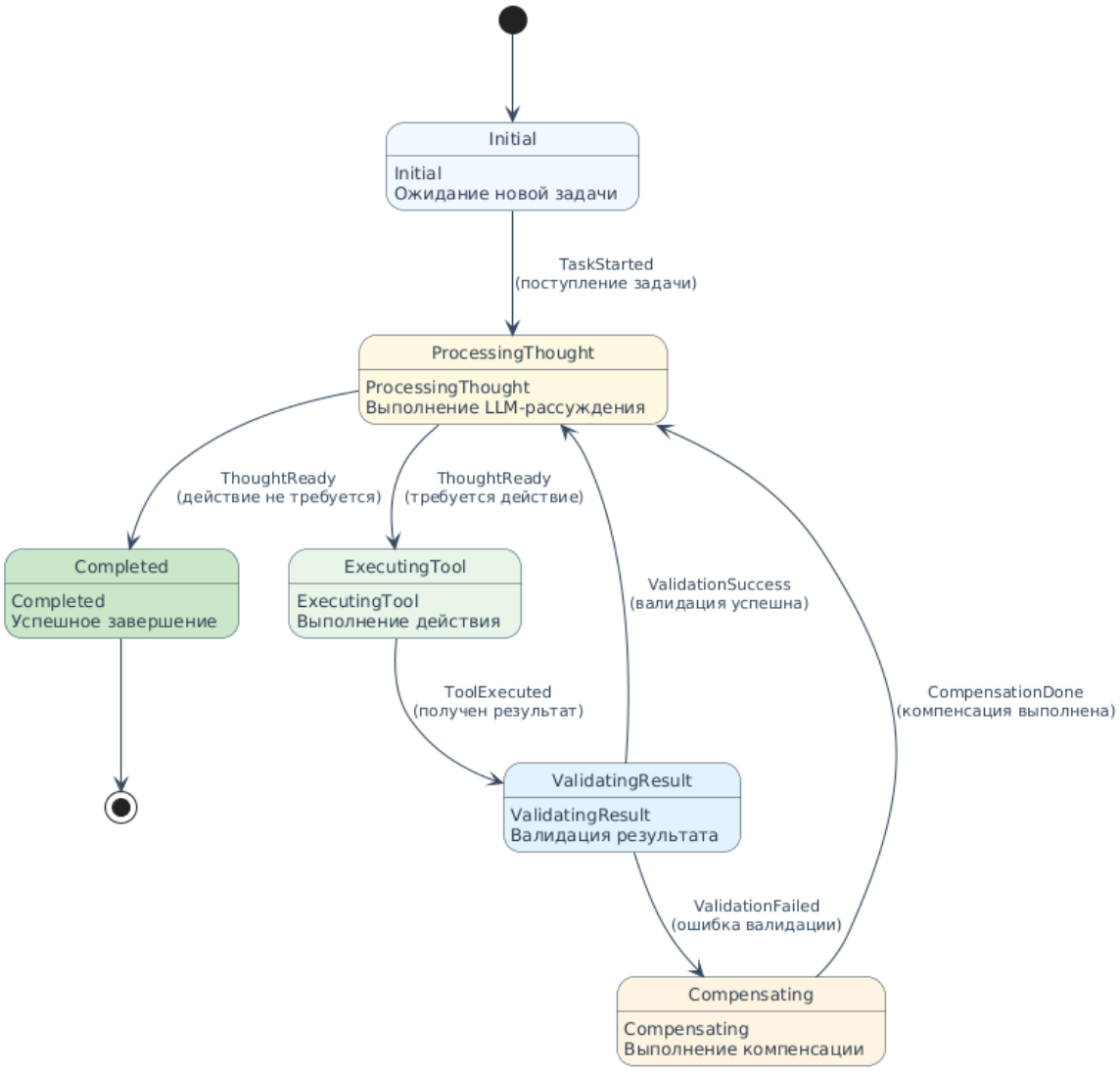

Рисунок. 1. Диаграмма состояний оркестратора LLM-агента

Диаграмма отражает итеративный характер работы LLM-агента. После успешной валидации одного шага агент переходит к генерации следующей мысли, и процесс повторяется до тех пор, пока не будет достигнуто состояние Completed. Такой же подход используется после выполнения компенсирующих операций. Это решение обусловлено

обеспечением устойчивости системы, где ошибка валидации не является фатальной, а служит сигналом для корректировки поведения агента, позволяя ему сгенерировать новое решение с учётом полученной информации.

Для реализации описанной модели разработан класс конечного автомата. В листинге 1 представлено определение состояния саги, содержащее данные для восстановления контекста выполнения агента.

Листинг 1. Состояния саги.

```
public class AgentSagaState : SagaStateMachineInstance
{
    public Guid CorrelationId { get; set; }
    public string CurrentState { get; set; }
    public string UserPrompt { get; set; }
    public string LastThought { get; set; }
    public string LastAction { get; set; }
    public string PreviousActionId { get; set; }
    public bool IsTaskCompleted { get; set; }
}
```

Данные поля автоматически сохраняются в персистентном хранилище при каждом изменении состояния, обеспечивая надёжное хранение долгоживущего процесса и восстановление после сбоев.

В листинге 2 представлена основная реализация конечного автомата.

Листинг 2. Реализация конечного автомата.

```
public class AgentOrchestrationStateMachine :
MassTransitStateMachine<AgentSagaState>
{
    public State ProcessingThought { get; private set; }
    public State ExecutingTool { get; private set; }
    public State ValidatingResult { get; private set; }
    public State Compensating { get; private set; }
    public State Completed { get; private set; }

    public Event<IAgentTaskStart> TaskStarted { get; private set; }
    public Event<IThoughtGenerated> ThoughtReady { get; private set; }
    public Event<IToolExecuted> ToolExecuted { get; private set; }
    public Event<IValidationFailed> ValidationFailed { get; private
set; }
    public Event<IValidationSuccess> ValidationSuccess { get; private
set; }
    public Event<ICompensationDone> CompensationDone { get; private
set; }

    public AgentOrchestrationStateMachine()
    {
        Event(() => TaskStarted, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));
```

```
        Event(() => ThoughtReady, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));
        Event(() => ToolExecuted, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));
        Event(() => ValidationFailed, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));
        Event(() => ValidationSuccess, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));
        Event(() => CompensationDone, x => x.CorrelateById(ctx =>
ctx.Message.TaskId));

        Initially(
            When(TaskStarted)
                .Then(ctx => ctx.Saga.UserPrompt = ctx.Message.Prompt)
                .Activity(x => x.OfType<RequestLlmThoughtActivity>())
                .TransitionTo(ProcessingThought)
        );

        During(ProcessingThought,
            When(ThoughtReady)
                .Then(ctx => ctx.Saga.LastThought =
ctx.Message.Thought)
                .If(ctx => ctx.Message.NeedsToolUse,
                    b => b.Activity(x =>
x.OfType<ExecuteToolActivity>()).TransitionTo(ExecutingTool))
                .If(ctx => !ctx.Message.NeedsToolUse,
                    b => b.TransitionTo(Completed))
        );

        During(ExecutingTool,
            When(ToolExecuted)
                .Then(ctx =>
                {
                    ctx.Saga.LastAction = ctx.Message.ActionName;
                    ctx.Saga.PreviousActionId = ctx.Message.ActionId;
                })
                .Activity(x => x.OfType<ValidateResultActivity>())
                .TransitionTo(ValidatingResult)
        );

        During(ValidatingResult,
            When(ValidationFailed)
                .Activity(x =>
x.OfType<ExecuteCompensationActivity>())
                .TransitionTo(Compensating),
            When(ValidationSuccess)
                .TransitionTo(ProcessingThought)
        );

        During(Compensating,
            When(CompensationDone)
```

```
                .TransitionTo(ProcessingThought)
        );

        SetCompletedWhenFinalized();
    }
}
```

Представленная реализация демонстрирует ключевые аспекты предлагаемого подхода. Механизм корреляции событий MassTransit через CorrelateById автоматически связывает входящие сообщения из Kafka с соответствующим экземпляром саги, что исключает необходимость ручной реализации маршрутизации. Activity инкапсулируют бизнес-логику вызова LLM, выполнения инструментов, валидации и компенсации в отдельные классы, обеспечивая разделение ответственности и упрощая тестирование каждого этапа процесса.

Важной особенностью реализации является переход после выполнения компенсации обратно в состояние ProcessingThought. Ошибка валидации не является фатальной, а служит сигналом для корректировки поведения агента. После выполнения компенсирующих операций система возвращается к этапу рассуждения, позволяя агенту сгенерировать новое решение с учётом полученной информации. Такой подход обеспечивает непрерывность выполнения задачи и повышает общую надёжность системы, поскольку единичные ошибки не приводят к остановке всего процесса.

Выводы.

В результате проведённого исследования разработан прототип оркестратора LLM-агента на основе конечного автомата и паттерна SAGA, реализованный на платформе .NET с использованием MassTransit. Предложенный подход обеспечивает формальное описание логики переходов, надёжное хранение состояния и встроенный механизм компенсирующих действий. Переход из состояния компенсации обратно в состояние рассуждения гарантирует устойчивость системы и непрерывность выполнения задач при возникновении ошибок.

Проведённый анализ показал, что существующие фреймворки не предоставляют встроенных механизмов атомарного обновления состояния и компенсирующих операций. Предложенный подход решает указанные проблемы, позволяя строить надёжные LLM-агентные системы для промышленных сценариев. Дальнейшие исследования могут быть направлены на масштабирование для мульти-агентных сценариев.

Литература

1. Баженов А. Э. Оптимизация взаимодействия микросервисов с использованием асинхронных вызовов и брокеров сообщений (Kafka, RabbitMQ) / А. Э.

Баженов, А. В. Райков, М. В. Нехаев, Н. В. Ореховский // Программные системы и вычислительные методы. – 2025. – № 4. – С. 77-93. URL: https://elibrary.ru/item.asp?id=88757183

2. Булискерия, Б. Б. Архитектурные паттерны отказоустойчивости в микросервисных системах: системный анализ и синтез / Б. Б. Булискерия, Д. М. Пряшников // Оптические технологии, материалы и системы (Оптотех - 2025): Международная научно-техническая конференция, Москва, 08–12 декабря 2025 года. – Москва: МИРЭА - Российский технологический университет, 2026. – С. 1309-1318. URL: https://elibrary.ru/item.asp?id=89137901

3. Ковалев, А. К. Применение предобученных больших языковых моделей в задачах воплощенного искусственного интеллекта / А. К. Ковалев, А. И. Панов // Доклады Российской академии наук. Математика, информатика, процессы управления. – 2022. – Т. 508, № 1. – С. 94-99. URL: https://elibrary.ru/item.asp?id=49991315

4. Пикина, Н. Е. Проблемы создания LLM агентов и способы их преодоления / Н. Е. Пикина, Е. А. Воробьев // Инновации в образовательном процессе : Сборник трудов XXIII Международной научно-практической конференции, посвященной 70-летию основания института, Году защитника Отечества, 80-летию Победы в Великой Отечественной войне, в рамках Десятилетия науки и технологий, Чебоксары, 24 апреля 2025 года. – Чебоксары: Политех, 2025. – С. 76-79. URL: https://elibrary.ru/item.asp?id=82969301

5. Канаева, И. А. Современные архитектуры разработки приложений / И. А. Канаева, М. А. Магомедов // Информационные технологии в экономике и управлении: сборник материалов V всероссийской научно-практической конференции, Махачкала, 23–24 ноября 2022 года. – Махачкала: Дагестанский государственный технический университет, 2022. – С. 120-123. URL: https://elibrary.ru/item.asp?id=50345626

6. Рубцов, Д. В. Использования шаблона "САГА" для поддержания согласованности данных в микросервисной архитектуре / Д. В. Рубцов // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. – 2020. – № 5. – С. 106-111. URL: https://elibrary.ru/item.asp?id=43784119

7. Кучеренко, Н. Ю. Проблема целостности данных в микросервисной архитектуре / Н. Ю. Кучеренко // Столыпинский вестник. – 2022. URL: https://elibrary.ru/item.asp?id=49403104

8. Сборник научных трудов кафедры автоматики и промышленной электроники. – Москва: Федеральное государственное бюджетное образовательное учреждение высшего образования "Российский государственный университет имени А.Н.

Косыгина (Технологии. Дизайн. Искусство)", 2021. – 153 с. URL: https://elibrary.ru/item.asp?id=45772617