УДК 004.89:004.7

РАСШИРЕНИЕ КОНТЕКСТНЫХ ВОЗМОЖНОСТЕЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ПОСРЕДСТВОМ ПРОТОКОЛА MODEL CONTEXT PROTOCOL (MCP)

Маслаков С. А, студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Жвакина А.В. - канд. техн. наук, доцент

Аннотация. В статье представлен подробный анализ Model Context Protocol (MCP) — открытого стандарта, разработанного для стандартизации взаимодействия больших языковых моделей (LLM) с внешними источниками данных. Рассматриваются принципы архитектурной схемы «Хост-Клиент-Сервер», механизмы передачи данных, использующие процедурный протокол JSON-RPC 2.0, а также особенности применения технологии потоковой передачи Server-Sent Events (SSE) для достижения высокой реактивности системы. Детально описывается полный итеративный цикл вызова внешних инструментов (Tool Calling) и процедура обратной инъекции полученных результатов в контекст модели для формирования окончательного ответа.

Ключевые слова. Model Context Protocol, MCP, JSON-RPC, LLM, контекстное обучение, вызов функций, архитектура ИИ-агентов.

Введение. Современный этап развития больших языковых моделей (LLM) столкнулся с фундаментальной проблемой: изоляцией ядра модели от актуальных данных и специализированных программных средств. Привычные методы интеграции требуют разработки уникальных адаптеров для каждого программного интерфейса (API), что закономерно ведет к фрагментации экосистемы и усложнению процесса разработки. Решением данной проблемы призван стать Model Context Protocol (MCP) — открытый стандарт, обеспечивающий ИИ-приложениям возможность бесшовного подключения к разнообразным внешним системам, от локальных файловых систем до распределенных облачных сервисов.

Архитектурная модель MCP. Протокол MCP использует трехуровневую архитектуру, гарантирующую необходимую гибкость и масштабируемость для построения сложных агентских систем. Структура включает следующие ключевые компоненты:

Хост (Host). Представляет собой исполняющую среду или приложение (например, интегрированную среду разработки (IDE) или специализированные ИИ-агенты), которому требуется доступ к внешним данным или функционалу. Хост выступает инициатором рабочей сессии и контролирует права доступа.

Клиент (Client). Является промежуточным слоем, встроенным в хост, и устанавливает прямое соединение формата «один-к-одному» с MCP-сервером. Основная задача Клиента — преобразование высокоуровневых запросов от LLM в стандартизованные сообщения протокола.

Сервер (Server). Это автономный программный модуль, который предоставляет специфические функциональные возможности. В зависимости от реализации, Сервер может обеспечивать доступ к локальным файлам, выполнять поисковые запросы или взаимодействовать с API внешних сервисов, таких как GitHub. Такая модульная архитектура (Рисунок 1) позволяет строго разграничить логику принятия решений (LLM) и логику получения и обработки данных, что значительно упрощает обслуживание и замену отдельных элементов системы.

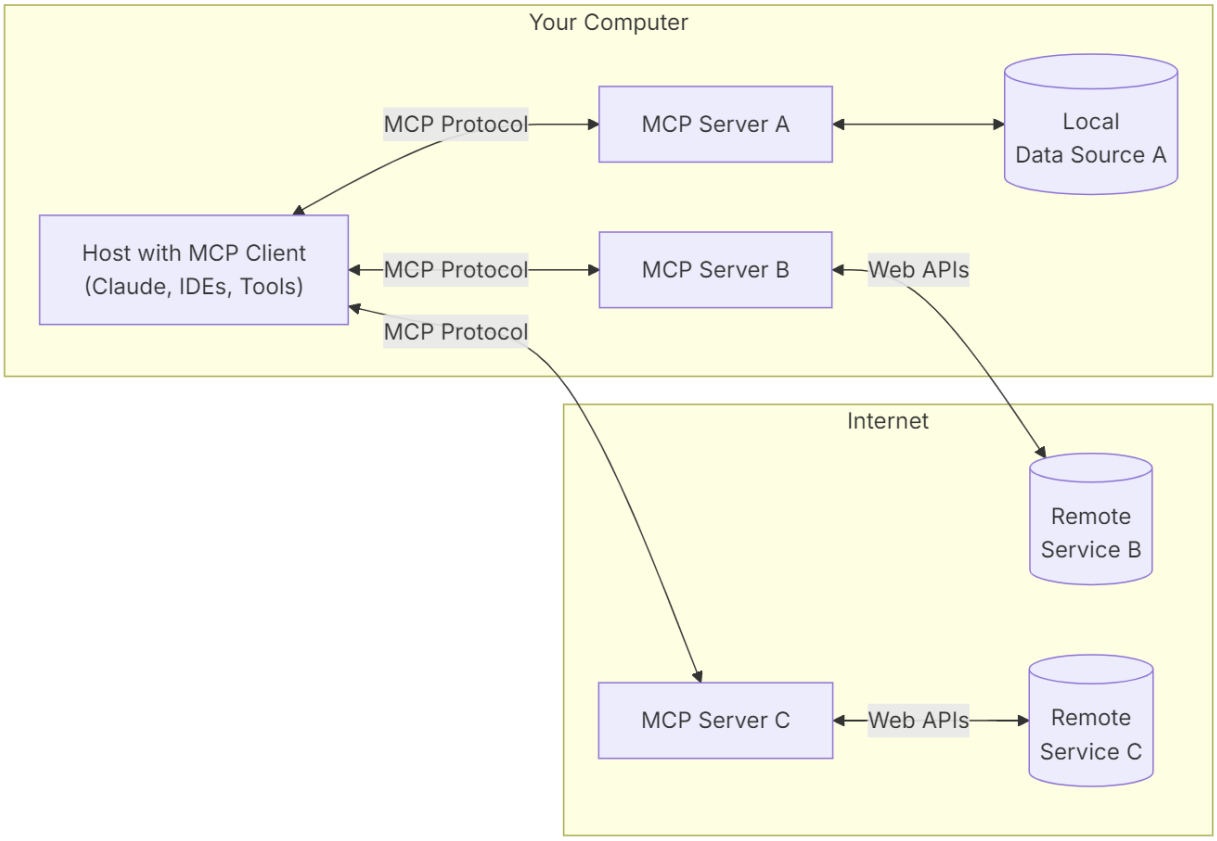

Рисунок 1 — Схема взаимодействия Host, Client и Server

Стек протоколов и передача данных. Основой для обмена сообщениями в MCP служит протокол удаленного вызова процедур JSON-RPC 2.0. Выбор данного протокола обусловлен его ориентацией на вызов процедур. В отличие от протоколов, оперирующих ресурсами (например, REST), JSON-RPC непосредственно сфокусирован на вызове методов, или «инструментов» в контексте LLM, что идеально соответствует парадигме Tool Calling. JSON-RPC обеспечивает надежный и структурированный обмен данными, где каждое сообщение запроса содержит поля «method» и «params», а ответ — «result» или «error».

Транспортный уровень MCP поддерживает два ключевых режима передачи данных. Первый, Stdio (использование стандартных потоков ввода-вывода), используется преимущественно для локальных серверов, работающих на той же машине, что и хост. Второй, более важный для распределенных сред, — Streamable HTTP, базирующийся на технологии Server-Sent Events (SSE). Технология SSE дает возможность серверу передавать данные клиенту в реальном времени через одно открытое HTTP-соединение. Это критически важно для длительных и ресурсоемких операций, таких как анализ больших объемов информации. Благодаря SSE, результат выполнения инструмента может поступать фрагментами, позволяя оркестратору LLM начать обработку контекста и формирование предварительного ответа до полного завершения процедуры.

Жизненный цикл соединения и типы сообщений. Взаимодействие Клиента и Сервера MCP начинается с обязательной фазы инициализации. Клиент отправляет запрос initialize, содержащий версию протокола и перечень своих возможностей (например, поддержка публикации логов). Сервер в ответ предоставляет полную спецификацию своих возможностей: список доступных инструментов (tools), шаблонов подсказок (prompts) и ресурсов (resources). Стандартный обмен сообщениями начинается только после того, как Сервер отправляет уведомление initialized, подтверждая свою готовность.

Спецификация MCP определяет три категории сообщений, циркулирующих между компонентами:

1. Requests: Сообщения, которые требуют обязательного ответа от получателя, например, прямой вызов определенного инструмента.
2. Responses: Содержат результат успешного выполнения запроса или подробное описание сбоя с кодом ошибки.

3. Notifications: Односторонние сообщения информационного характера, не требующие подтверждения. Примером может служить уведомление об изменении состояния внешнего файла, доступного агенту.
4. Механизм Tool Calling и интеграция с LLM

Механизм вызова инструментов является ключевой особенностью MCP, трансформирующей LLM из пассивного текстового генератора в активный, функционально нагруженный агент. Процесс Tool Calling через MCP API представляет собой итеративный цикл, состоящий из четырех последовательных этапов.

Стадия 1: Регистрация возможностей. На этом этапе оркестратор, управляющий моделью, при каждом новом запросе передает ей набор описаний доступных инструментов. Эти описания представлены в стандартизированном формате JSON Schema и включают имя инструмента, его описание, а также строгую структуру требуемых параметров.

Стадия 2: Генерация вызова. Если в процессе обработки запроса пользователя модель определяет необходимость получения внешних актуальных данных (например, для ответа на вопрос о погоде), она прекращает генерацию текста. Вместо этого модель формирует структурированный объект tool_calls с уникальным идентификатором вызова (ID) и аргументами, необходимыми для исполнения выбранного инструмента (Рисунок 2) .

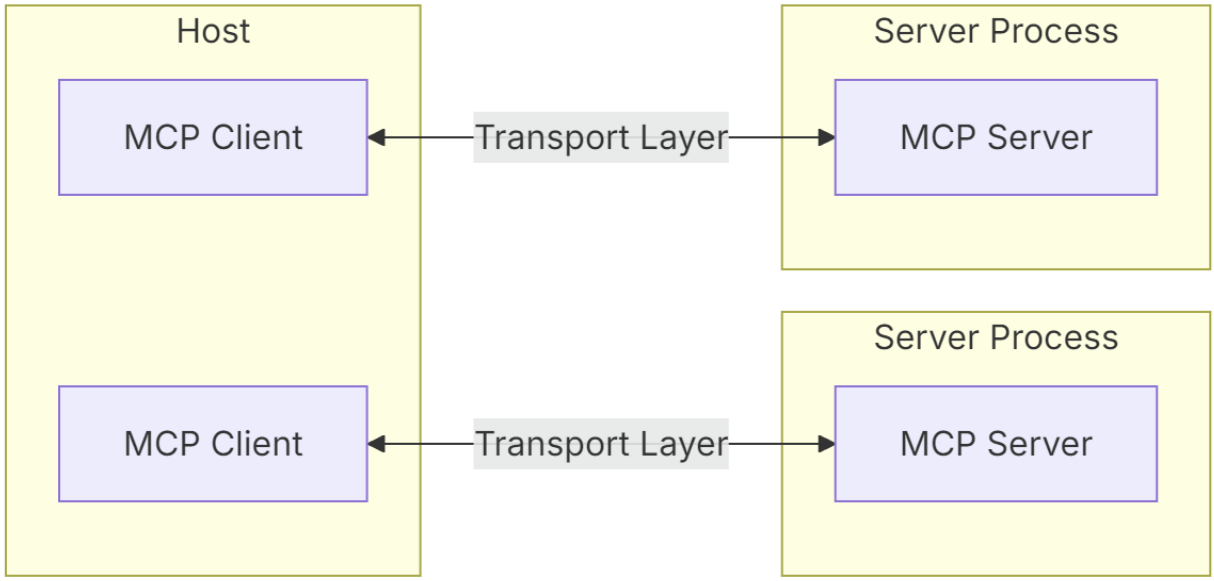

Рисунок 2 — Схема итерационного цикла Tool Calling

Стадия 3: Исполнение и инъекция (Output Injection). Сгенерированный вызов инструмента перехватывается внешним оркестратором. Оркестратор инициирует процедуру вызова соответствующего MCP-сервера, который выполняет фактическое действие (например, отправляет запрос на метеорологический API) и возвращает результат. Полученные данные (к примеру, JSON-объект с температурой и осадками) не направляются напрямую пользователю. Вместо этого они инъецируются обратно в контекст LLM с особой ролью tool и ссылкой на уникальный ID исходного вызова. Этот процесс, известный как «Output Injection», расширяет контекстное окно модели актуальной информацией.

Стадия 4: Финальная генерация. На финальном этапе модель повторно анализирует весь расширенный контекст, который теперь включает не только исходный запрос пользователя, но и полученные от инструмента данные. Основываясь на этой полной информации, модель формирует естественный, семантически корректный и информативный ответ для пользователя.

Структура и безопасность инструментов. Каждый инструмент, определенный в рамках MCP, дополняется аннотациями, которые повышают безопасность и эффективность его использования моделью. Аннотация readOnlyHint используется для указания на то, что инструмент не изменяет состояние внешней системы (например, поисковый запрос), что делает его безопасным для многократных вызовов. В то же время, аннотация destructiveHint служит предупреждением о потенциально необратимых действиях (например, удаление данных). Наличие этой аннотации позволяет оркестратору или пользовательскому интерфейсу запрашивать дополнительное подтверждение перед исполнением инструмента, что является важным элементом безопасности.

Заключение. Model Context Protocol устанавливает новый унифицированный стандарт для экосистемы искусственного интеллекта, обеспечивая стандартизированную и бесшовную интеграцию LLM с внешним информационным пространством. Использование процедурного JSON-RPC в сочетании с технологией потоковой передачи SSE обеспечивает высокую реактивность системы, четкую структуру обмена данными и способность эффективно обрабатывать длительные операции. Итеративный механизм Tool Calling, подробно описанный в спецификации, позволяет создавать сложные автономные ИИ-агенты, способные оперировать актуальной информацией в режиме реального времени. Внедрение MCP в образовательные, исследовательские и промышленные проекты позволит существенно снизить затраты на разработку интеллектуальных интерфейсов и повысить их функциональность.

Список использованных источников:

1. *MCP Specification [Electronic resource]. – Mode of access: – Date of access: 23.12.2025.*
2. *JSON-RPC 2.0 Specification [Electronic resource]. – Mode of access: – Date of access: 23.12.2025.*
3. *Badenkov, A.V. Development of AI agents based on function calling / A.V. Badenkov // Journal of Artificial Intelligence Systems. – 2024. – №3. – P. 45–52.*

UDC 004.89:004.7

EXTENDING CONTEXTUAL CAPABILITIES OF ARTIFICIAL INTELLIGENCE THROUGH THE MODEL CONTEXT PROTOCOL (MCP)

Maslakov S. A., student

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Zhvakina A.V. – PhD in Technical Sciences

Annotation. This article presents a detailed analysis of the Model Context Protocol (MCP)—an open standard developed to standardize the interaction of large-scale language models (LLM) with external data sources. It examines the principles of the Host-Client-Server architectural scheme, data transfer mechanisms using the JSON-RPC 2.0 procedural protocol, and the application of Server-Sent Events (SSE) streaming technology to achieve high system responsiveness. The full iterative cycle of external tool calling and the procedure for injecting the obtained results back into the model context to generate the final answer are described in detail.

Keywords. Model Context Protocol, MCP, JSON-RPC, LLM, contextual learning, function invocation, AI agent architecture.