# Pizza Ordering System using MCP with Agent-to-Agent Communication

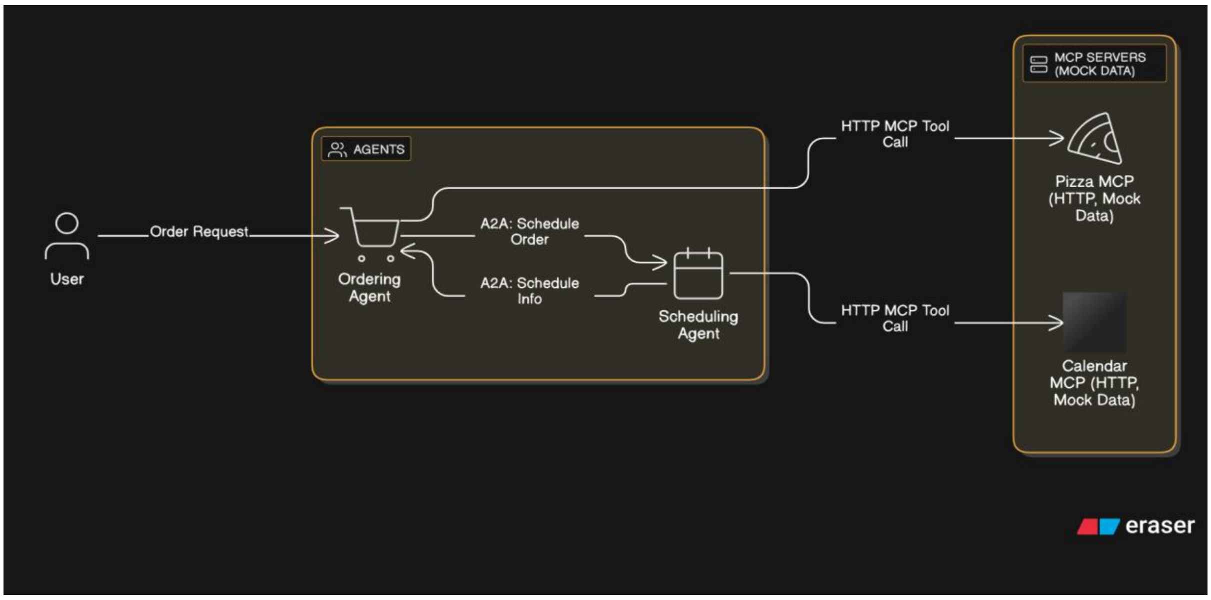
1. **Problem Statement**: In this project, I designed a pizza ordering system that demonstrates how agents can work together using Model Context Protocol (MCP) and agent -to-agent communication. Rather than making direct API calls, the system converts traditional OpenAPI into MCP servers, allowing agents to interact through tools in a structured way.

   This project demonstrates:
   - MCP server generation from an OpenAPI specification.
   - Autonomous agents with clear and structured instructions given.
   - Agent-to-Agent (A2A) communication.
   - Explicit handling of ambiguities and assumptions.
2. **High-Level Architecture:**
   The system is divided into three main parts:
   1.MCP Server
   2.Ordering Agent
   3.Scheduling Agent

   The MCP server exposes tools that are generated from OpenAPI YAML file. The agents do not directly call APIs or embed server logic.Instead, they reason about the user request and interact with the system only through MCP tools.

   I kept the architecture simple and modular so that each component has a clear responsibility and can be extended later.

**Simple Architecture Diagram**:



The diagram shows the high-level architecture of the system.
The Ordering Agent receives the user request and coordinates the flow.

When delivery scheduling is required, it communicates with the scheduling agent using Agent-to-Agent communication.

Both agents interact with their MCP server through HTTP, and the MCP servers use mock data to stimulate backend behavior.

## 3. MCP Server Generation:

The MCP server is created using the **Open API YAML specification** as the source of truth.

The Open Api file defines:

- Available tools(endpoints)
- Input parameter and schemas
- Output structure
- Validation rules

I choose open API because it provides a clean contract between agents and the server. If a new tool needs to be added or an existing on changes, the MCP server can be updated without changing the agent logic.

This approach keeps the system flexible and aligned with MCP’s tool-based interaction model.

**Pizza MCP Server -> Ordering Agent**: The Pizza Order MCP server exposes tools that enable the Ordering Agent to manage pizza orders without directly calling any REST API.

Calendar MCP Server->Scheduling Agent: The Scheduling Agent interacts with the Calendar MCP Server to Schedule delivery using MCP tools.

## 4. Agent Design:

### 4.1 Ordering Agent:

The Ordering Agent is responsible for understanding the user’s intent.

Its main tasks are:

- Interpreting the user request.
- Deciding which MCP tool to call
- Preparing the required arguments for the tool.

The agent does’nt hard code workflows. Instead, it reasons dynamically based on the user input.

This makes the agent more autonomous and easier to adapt to new scenarios.

### 4.2 Scheduling Agent:

The Scheduling Agent focuses only on delivery time and scheduling decisions.

I separated this logic from the ordering agent to avoid mixing concerns.

This separation made the system easier to reason about and closer to how real distributed systems are designed.

We can add additional agents, such as when payments are needed later, they could be added without modifying the existing agent.

## 5. Agent-to-Agent (A2A) Communication :

Agent-to-Agent communication is used whenever a task needs to be delegated.

For example, when an order requires delivery scheduling, the Ordering Agent communicates with the Scheduling Agent instead of handling scheduling itself.

This approach:

- Keeps agents focused on single responsibility
- Avoids duplicated logic
- Makes interaction explicit and easier to debug.

The communication is clear between the two agents rather than implicit.

## 6. Protocol Fidelity (MCP & A2A):

I made sure to follow MCP principles throughout the project.

Key points:

- Agents interact with the system only through MCP tools.
- No direct API calls are made inside agent business logic.
- Tool inputs and outputs are structured and validated.
- Agent communication follows clear boundaries.

I used this method to avoid shortcuts to ensure the design stays aligned with MCP concepts.

## 7. Handling Ambiguites & Design Assumption:

The problem statement intentionally leaves several details unspecified such as:

- Default values when user input is incomplete.
- If the user orders a pizza not in the MCP server, it will give a default pizza **and size.**

**8. Code Structure & Quality**:

The codebase is organized to keep concerns clearly separated:

- MCP server logic
- Agent logic
- Configuration setup
- Requirements.txt
- Created venv environment

**9. End-to-End flow Summary:**

The overall flow of the system is:

1. User submits a request.
2. Ordering Agent interprets intent
3. Ordering Agent selects the appropriate MCP tool.
4. Scheduling Agent is consulted if needed.
5. MCP server executes the tool.
6. A structured response is returned.

This flow demonstrates agent autonomy, MCP tool usage, and agent-to-agent collaboration.