Software Buildr Team

- MCP - Introduction
- Agent-to-Agent (A2A) - Introduction
- Microsoft Agent Framework (MAF)

MCP - Introduction

Model Context Protocol (MCP)

What is MCP?

Model Context Protocol (MCP) is an open standard that enables Artificial Intelligence (AI) models, particularly Large Language Models (LLMs), to securely connect with external tools, applications, databases, and data sources. It acts as a common communication layer between AI agents and the systems they need to interact with.

Just as USB provides a standardized way for devices to connect to computers, MCP provides a standardized way for AI models to access external resources and perform actions beyond their built-in capabilities.

Why is MCP Used?

Traditional AI models are limited to the information they were trained on and cannot directly access real-time data or enterprise systems. MCP addresses this limitation by allowing AI applications to:

- Access live and up-to-date information.
- Connect to enterprise tools and business applications.
- Retrieve data from databases and APIs.
- Execute actions through external services.
- Reduce the need for custom integrations for every tool.

By using MCP, organizations can build more powerful and scalable AI solutions without creating separate connectors for each application.

Key Components of MCP

1. MCP Host

The MCP Host is the application that contains or uses the AI model. Examples include AI assistants, chatbots, IDEs, and enterprise AI platforms.

Responsibilities: Initiates requests; Manages communication with MCP servers; Presents results to users.

2. MCP Client

The MCP Client acts as an intermediary between the host and MCP servers.

Responsibilities: Sends requests to MCP servers; Receives responses; Maintains protocol communication.

3. MCP Server

An MCP Server exposes tools, resources, or services that AI models can access. Examples include Database servers, CRM systems, Ticketing platforms, Cloud services, and Internal business applications. The server provides standardized interfaces so that AI models can interact with these systems consistently.

4. Resources

Resources represent data that AI models can read, such as Documents, Database records, Configuration files, and Knowledge bases.

5. Tools

Tools are functions or actions that the AI model can execute, such as creating a support ticket, running a database query, sending an email, or generating a report.

How MCP Works

1. A user submits a request to an AI application.
2. The AI determines that external information or functionality is required.
3. The MCP Client sends a request to an MCP Server.
4. The MCP Server provides access to the required resource or tool.
5. The result is returned to the AI model.
6. The AI generates a response using the retrieved information.

This process enables AI systems to perform tasks that would otherwise be impossible using only their training data.

Benefits of MCP

Standardized Integration: Developers can connect AI models to multiple systems using a common protocol instead of building custom integrations for each application.

Scalability: New tools and services can be added without modifying the AI model itself.

Reusability: The same MCP server can be used by multiple AI applications and agents.

Security: MCP supports controlled access to resources and tools, helping organizations manage permissions and data access.

Improved AI Capabilities: AI models can access real-time data, execute actions, and interact with enterprise systems, making them significantly more useful.

Common Use Cases

- AI-powered customer support systems
- IT service management and ticketing automation
- Enterprise knowledge assistants
- Software development assistants
- Database querying and reporting
- Cloud infrastructure management
- Workflow automation

Conclusion

Model Context Protocol (MCP) is becoming a key standard for connecting AI models with external systems and tools. By providing a unified and secure communication framework, MCP enables AI applications to access real-time information, perform actions, and integrate seamlessly with

enterprise environments. As organizations increasingly adopt AI-driven solutions, MCP plays a crucial role in making AI systems more powerful, scalable, and practical for real-world use.

Agent-to-Agent (A2A) - Introduction

Agent-to-Agent (A2A)

Agent-to-Agent (A2A) refers to a paradigm in artificial intelligence where multiple AI agents communicate, coordinate, and collaborate directly to accomplish a shared task. Rather than relying on a single monolithic model, A2A distributes intelligence across specialised agents — each with a defined role such as planning, execution, validation, or summarisation — interacting in structured, collaborative workflows.

Why Agent-to-Agent Systems Matter

Scalability of Intelligence

A single AI agent struggles with complex, multi-dimensional tasks. A2A solves this by distributing work: each agent focuses on a defined subtask, and the system scales by simply adding more agents as complexity grows — no redesign needed.

Improved Reliability and Validation

Agents cross-validate each other — one generates a solution, a second checks accuracy, a third flags risks. This built-in validation layer significantly reduces errors, especially in high-stakes domains like finance, healthcare, and legal analysis.

How A2A Works: Common Architectures

Sequential Pipeline

Agents operate in a strict chain. Each agent passes its output as input to the next, forming a linear workflow from planning through to validation.

```
[ Planning Agent ] ⟶ [ Executor Agent ] ⟶ [ Validation Agent ]
```

Master-Slave Architecture

A central Master Agent coordinates Slave Agents running in parallel, then consolidates their results into a unified output.

```
            MASTER AGENT (Coordinator)
          ▼            ▼            ▼
    SLAVE 1 /     SLAVE 2 /     SLAVE 3 /
    Planner       Executor      Validator
```

Collaborative Mesh (Shared Context)

Every agent has full visibility of the shared context bus. Agents run in parallel and communicate bidirectionally — enabling real-time re-planning and cross-agent awareness.

```
              MASTER AGENT (Orchestrator)
              ▼ Shared Context Bus ▼

  A1 / Planner ↔ A2 / Coder ↔ A3 / Validator ↔ A4 / Summarizer

  All agents share full context — enabling dynamic
  re-planning and real-time cross-agent awareness
```

Real-World Applications

In software development, a planner designs architecture, a coder writes, and a tester validates autonomously. In finance, agents gather data, build models, and generate risk reports in parallel. Healthcare deployments cross-reference patient data and flag contraindications. In content production, research, drafting, editing, and fact-checking agents collaborate at enterprise scale.

Challenges and Considerations

A2A introduces orchestration complexity, inter-agent communication latency, and risk of cascading errors. Federated deployments require robust authentication and sandboxing protocols to prevent data leakage or unauthorised agent actions.

Conclusion

Agent-to-Agent communication represents a fundamental shift in how AI systems are designed and deployed. By enabling intelligent agents to coordinate autonomously, A2A unlocks capabilities far beyond what any single model can achieve. As the technology matures, A2A architectures are set to become a cornerstone of the next generation of enterprise AI solutions — driving speed, accuracy, and resilience at scale.

Microsoft Agent Framework (MAF)

Keywords: Agentic AI, Microsoft Agent Framework, Semantic Kernel, AutoGen, Multi-Agent Systems, Enterprise AI

1. Introduction

Large Language Models (LLMs) have transformed software development by enabling natural language understanding and generation. Traditional AI assistants are highly effective at answering questions and generating content but remain largely reactive in nature.

Modern enterprises require AI systems capable of making decisions, coordinating actions, interacting with external services, and executing complex workflows autonomously. This need has led to the emergence of Agentic AI, where intelligent agents pursue goals rather than simply responding to prompts.

Microsoft has progressively evolved its AI ecosystem to support this vision through Semantic Kernel, AutoGen, and Microsoft Agent Framework (MAF).

2. What is Agentic AI?

Agentic AI refers to intelligent systems capable of independently pursuing objectives through reasoning, planning, memory retention, and tool utilization.

Unlike traditional chatbots that focus on response generation, agentic systems focus on achieving outcomes by performing actions and coordinating multiple tasks.

Key Characteristics

- Goal-Oriented Execution
- Autonomous Decision Making
- Tool and API Integration
- Persistent Memory

- Multi-Step Task Execution
- Multi-Agent Collaboration

3. Evolution of Microsoft's Agent Ecosystem

Microsoft's journey toward Agentic AI can be viewed in three major phases.

Phase	Technology	Purpose
1	Semantic Kernel	AI orchestration
2	AutoGen	Multi-agent collaboration
3	Microsoft Agent Framework	Enterprise-grade agent systems

- Semantic Kernel
  - Semantic Kernel introduced orchestration capabilities that connected LLMs with plugins, memory, and external services. It provided developers with a structured method to integrate AI capabilities into applications.
- AutoGen
  - AutoGen expanded these capabilities through collaborative intelligence, enabling multiple specialized agents to communicate and solve tasks together through distributed reasoning.
- Microsoft Agent Framework
  - Microsoft Agent Framework combines orchestration, workflows, memory management, governance, and observability into a unified platform designed for enterprise-scale agentic systems.

4. Microsoft Agent Framework Architecture

Microsoft Agent Framework provides a layered architecture for building enterprise-grade AI systems. A user request flows into the MAF orchestration platform, which coordinates specialized agents (Planner, Search, and Data agents) that connect to external APIs and enterprise systems. A shared memory layer manages context, history, and state, while a workflow engine handles planning, routing, and orchestration. Security and governance (authentication, authorization, compliance, audit trails), a human-in-the-loop path (approval, feedback, manual intervention), and observability and monitoring (logs, metrics, traces) wrap the platform before a final response is

returned.

5. Semantic Kernel vs Microsoft Agent Framework

Feature	Semantic Kernel	Microsoft Agent Framework
Focus	AI Orchestration	Agent Systems
Agents	Limited	Native
Multi-Agent	Basic	Advanced
Workflows	Simple	Enterprise-Grade
Governance	Limited	Comprehensive
Observability	Basic	Extensive
Production Readiness	Moderate	High

Semantic Kernel established the foundation for AI orchestration, while Microsoft Agent Framework extends these capabilities through native support for multi-agent collaboration, workflow automation, governance, and enterprise deployment.

6. Conclusion

The evolution from Semantic Kernel to AutoGen and ultimately Microsoft Agent Framework reflects Microsoft's vision for enterprise Agentic AI. Semantic Kernel introduced orchestration, AutoGen enabled collaborative intelligence, and MAF unified these concepts into a production-ready framework.

As organizations increasingly adopt autonomous systems, agentic architectures will become a fundamental design pattern for enterprise automation. Microsoft Agent Framework provides the necessary capabilities to build secure, scalable, and intelligent applications that can reason, collaborate, and execute complex workflows with minimal human intervention.