Model Meeting Protocol: Dynamic Protocol Evolution through Skill-Based Definition for Agent-to-Agent Communication

Masaomi Hatakeyama

1 February 2026 (v1.1)

Contents

0.1 Abstract

As AI agents proliferate across diverse platforms and frameworks, the need for standardized inter-agent communication becomes critical. While existing protocols such as Anthropic's Model Context Protocol (MCP) address LLM-to-tool communication and Google's Agent2Agent Protocol (A2A) enables agent interoperability, both treat protocol capabilities as static definitions. This paper introduces the Model Meeting Protocol (MMP), an open standard for agent-to-agent communication that implements a novel "Protocol as Skill" paradigm—where protocol definitions themselves are exchangeable skills that can evolve through agent interaction. MMP extends the KairosChain framework's skill management to distributed environments, enabling agents to discover each other, negotiate capabilities, and co-evolve their communication patterns while maintaining safety through layered constraints and human approval mechanisms. We argue that this approach provides the foundation for a future where AI agents autonomously develop and propagate specialized communication protocols within trust networks.

Keywords: agent-to-agent communication, protocol evolution, MCP, A2A, skill exchange, co-evolution, distributed AI systems

---

0.2 1. Introduction

The rapid advancement of Large Language Model (LLM) agents has created an ecosystem of increasingly capable AI systems operating across different platforms, frameworks, and vendors. These agents—whether implemented as MCP servers, autonomous coding assistants, or specialized domain experts—increasingly need to communicate with each other to accomplish complex tasks. However, the current landscape of agent communication protocols faces a fundamental limitation: protocol capabilities are defined statically, requiring specification updates to extend functionality.

Anthropic's Model Context Protocol (MCP) [1] has established a standard for how LLMs communicate with tool servers, using JSON-RPC 2.0 over stdio or HTTP. Google's Agent2Agent Protocol (A2A) [2], announced in April 2025, addresses inter-agent communication through Agent Cards that declare capabilities. While A2A enables agents from different vendors to collaborate, its AgentSkill definitions are static JSON structures embedded in Agent Cards—adding new protocol actions requires updating the specification itself.

This static approach conflicts with the observed behavior of multi-agent systems. Recent research on emergent communication [3] demonstrates that AI agents can spontaneously develop sophisticated communication protocols through a four-phase evolution: exploration, signal consolidation, protocol optimization, and protocol maturation. The question arises: can we design a protocol that supports this natural evolution while maintaining safety and auditability?

This paper introduces the Model Meeting Protocol (MMP), which addresses this challenge through a "Protocol as Skill" paradigm. In MMP, protocol definitions are not hardcoded specifications but

rather skill files—Markdown documents with structured metadata—that agents can exchange, adopt, and propagate. This approach enables:

1. **Dynamic extensibility**: New protocol actions can be added without specification changes
2. **Controlled co-evolution**: Agents can teach each other new capabilities within safety bounds
3. **Layered governance**: Core actions are immutable while extensions can evolve
4. **Privacy-preserving relay**: End-to-end encryption ensures communication privacy even through central relay points

MMP builds upon the KairosChain framework [4], which implements auditable AI skill evolution using blockchain-based recording and the Minimum-Nomic principle [5]. Where KairosChain addresses single-agent skill management, MMP extends this to distributed multi-agent environments.

0.3 2. Related Work

0.3.1 2.1 Model Context Protocol (MCP)

Anthropic's Model Context Protocol [1] provides a standardized way for LLM applications to communicate with external data sources and tools. Often described as "USB-C for AI applications," MCP uses JSON-RPC 2.0 messages for communication between hosts (LLM applications), clients (connectors), and servers (external services).

The MCP specification (Version 2025-11-25) covers base protocol lifecycle, transports, authorization, and server features including prompts, resources, and tools. MCP has achieved significant adoption, with OpenAI's Agents SDK now supporting MCP server integration [6].

Scope and Limitations: MCP focuses on LLM-to-tool communication, primarily over stdio for local integrations. While HTTP transport is supported, MCP does not address agent-to-agent communication or capability negotiation between peer agents. The protocol treats tool definitions as static configurations rather than exchangeable capabilities.

0.3.2 2.2 Agent2Agent Protocol (A2A)

Google's Agent2Agent Protocol [2], launched in April 2025 with support from over 50 technology partners, enables communication between independent AI agent systems. A2A is built on HTTP, JSON-RPC 2.0, and Server-Sent Events, supporting synchronous, streaming, and asynchronous interaction patterns.

Central to A2A is the **Agent Card**, a JSON metadata document describing an agent's identity, capabilities, skills, and authentication requirements. The **AgentSkill** structure defines specific capabilities:

```
{
  "id": "route-optimizer-traffic",
  "name": "Traffic-Aware Route Optimizer",
  "description": "Calculates optimal driving routes...",
  "tags": ["maps", "routing", "navigation"],
```

```
  "inputModes": ["application/json", "text/plain"],
  "outputModes": ["application/json", "application/vnd.geo+json"]
}
```

Scope and Limitations: While A2A enables robust agent interoperability, its approach to capabilities is fundamentally static. AgentSkill definitions are embedded in Agent Cards and cannot be dynamically extended. Adding new protocol actions (beyond the defined Task lifecycle and messaging) requires specification updates. The protocol does not support agents teaching each other new capabilities or evolving their communication patterns over time.

0.3.3 2.3 Emergent Communication in Multi-Agent Systems

Recent research on emergent communication provides theoretical grounding for protocol evolution. A January 2025 study [3] introduces the Multi-Agent Protocol Evolution Framework (MAPEF), identifying four phases in how AI agents spontaneously develop communication systems:

1. **Exploration**: Random signal generation with low success rates
2. **Signal Consolidation**: Pattern stabilization as agents converge on useful signals
3. **Protocol Optimization**: Development of efficient communication structures
4. **Protocol Maturation**: Error correction and context-dependent messaging

The research identifies "communication bottlenecks"—critical learning points where protocols either advance to higher sophistication or stabilize at simpler forms. Notably, artificial protocol evolution exhibits parallels with natural language development.

This research suggests that restricting agents to static protocol definitions may limit their collaborative potential. A protocol that supports controlled evolution could harness these emergent capabilities while maintaining safety.

0.3.4 2.4 KairosChain: Auditable AI Evolution

KairosChain [4] provides the conceptual and technical foundation for MMP. It implements a meta ledger that records AI skill evolution on a private blockchain, combining:

- **Pure Agent Skills**: Executable skill definitions using Ruby DSL and AST
- **Minimum-Nomic Principle**: Rules can be amended, but amendment history is immutable
- **Layered Architecture**: L0 (constitution), L1 (law), L2 (ordinance) for different constraint levels

KairosChain demonstrates that auditable evolution is achievable for AI capabilities. MMP extends this approach to inter-agent communication, treating protocol definitions as skills that can be exchanged between agents while maintaining the same layered governance model.

0.4 3. MMP Architecture

0.4.1 3.1 Design Principles

MMP is designed around four core principles:

Minimal Core: Only three actions are truly immutable: - `introduce`: Establish identity and declare capabilities - `goodbye`: End communication gracefully
- `error`: Report error conditions

Everything else is an extension that agents can choose to support.

Safety by Default: - Skill exchange defaults to Markdown-only (no executable code) - All relayed messages use end-to-end encryption - Capability changes require human approval for promotion to stable status

Extensibility through Skills: - Protocol extensions are defined as skill files (Markdown with YAML frontmatter) - Agents can request, receive, and adopt new extensions from peers - Extensions propagate through trust networks

Graceful Degradation: - Agents with different capabilities can still communicate using their common subset - Unknown actions return structured errors rather than failures

0.4.2 3.2 Relationship to MCP and A2A

MMP occupies a distinct position in the agent communication landscape:

Layer	Protocol	Scope
LLM ↔ Tools	MCP	Local, stdio/HTTP, static tools
Agent ↔ Agent	A2A	Network, HTTP/SSE, static skills
Agent ↔ Agent	MMP	Network, HTTP, evolvable protocol

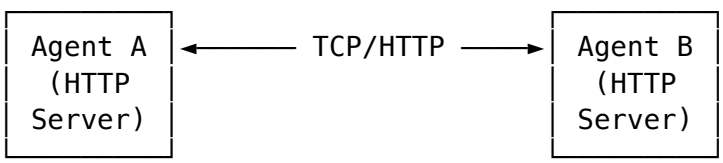
- **MCP** defines how an LLM talks to its tools (vertical integration)
- **A2A** defines how agents from different vendors collaborate (horizontal integration, static)
- **MMP** enables agents to evolve their collaboration patterns (horizontal integration, dynamic)

MMP is designed to complement rather than replace these protocols. An agent might use MCP for tool access, A2A for interoperability with enterprise systems, and MMP for capability co-evolution with trusted peers.

0.4.3 3.3 Communication Modes

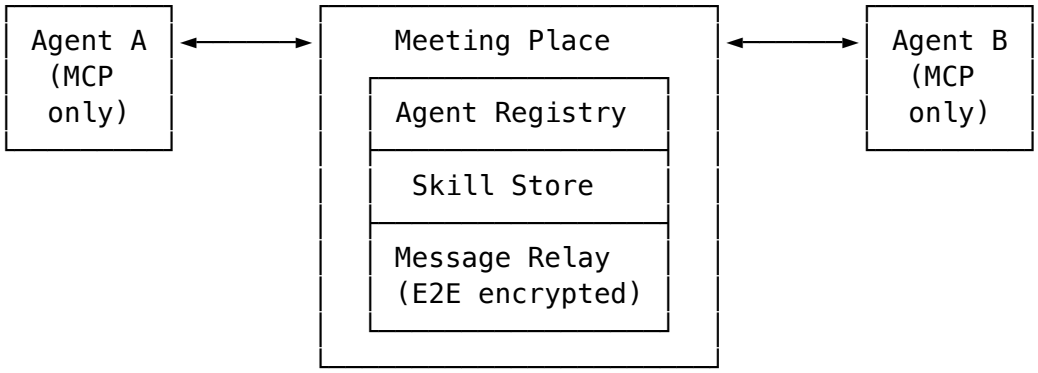
MMP supports two communication modes:

Direct Mode (P2P):



Agents communicate directly when network topology permits, using HTTP endpoints exposed by each agent's MMP server. This mode requires both agents to run HTTP servers and have publicly accessible endpoints.

Relay Mode (via Meeting Place):



In Relay Mode, agents do not need to run their own HTTP servers. The Meeting Place provides:

- **Agent Registry**: Track connected agents and their capabilities, with TTL-based cleanup for inactive agents
- **Skill Store**: Agents publish skills directly to the Meeting Place; other agents can browse and acquire skills without direct P2P communication
- **Message Relay**: Forward encrypted messages between agents
- **Ghost Agent Cleanup**: Automatic removal of unresponsive agents via TTL expiration, on-demand health checks, or administrator commands

Key differences between modes:

Aspect	Direct Mode	Relay Mode
Agent HTTP server required	Yes	No
Skill exchange mechanism	Direct request to peer	Via SkillStore on Meeting Place
Network requirements	Both agents publicly accessible	Only Meeting Place publicly accessible
Latency	Lower (direct)	Higher (via relay)
Operational complexity	Higher (run HTTP server)	Lower (MCP client only)

Critically, even in Relay Mode, the Meeting Place operates as a **router only** for messages—it stores and forwards encrypted blobs without access to message content. The SkillStore, however, stores skill metadata and content in cleartext to enable browsing and discovery. This design enables auditability (metadata logging) without surveillance (message content inspection).

0.4.4 3.4 Message Format

All MMP messages use a common envelope structure:

```
{
  "id": "msg_a1b2c3d4e5f6g7h8",
  "action": "introduce",
  "from": "agent_abc123",
  "to": "agent_xyz789",
  "timestamp": "2026-01-30T12:00:00Z",
  "protocol_version": "1.0.0",
  "payload": { ... },
  "in_reply_to": "msg_previous_id"
}
```

This format is intentionally similar to A2A's Message structure, facilitating potential bridging between protocols.

0.4.5 3.5 Layered Protocol Extensions

MMP implements a three-layer architecture for protocol definitions:

Layer	Name	Mutability	Blockchain Record	Example
L0	Core	Immutable	Full	`introduce`, `goodbye`, `error`
L1	Stable	Human approval	Hash reference	`skill_exchange`, `evolution`
L2	Experimental	Free modification	None	Custom domain extensions

This layered approach mirrors KairosChain's legal-system-inspired architecture, recognizing that different protocol elements require different levels of governance.

The full architecture diagrams are available in supplementary material: MMP Architecture Diagrams

---

0.5 4. Protocol as Skill: The Key Innovation

0.5.1 4.1 Comparison with A2A AgentSkill

The fundamental difference between MMP and A2A lies in how protocol capabilities are defined and managed:

Aspect	A2A AgentSkill	MMP Protocol Skill
Definition Format	JSON in Agent Card	Markdown with YAML frontmatter
Storage	Embedded in metadata	Separate skill files

Aspect	A2A AgentSkill	MMP Protocol Skill
Mutability	Static (spec change required)	Dynamic (skill exchange)
Extension Mechanism	New specification version	Adopt new skill file
Discovery	Agent Card query	`introduce` + capability negotiation
Evolution	Manual specification update	Agent-to-agent teaching
Governance	Specification committee	Layered (L0/L1/L2) with human approval

In A2A, if two agents want to support a new interaction pattern (e.g., structured debate), both must wait for a specification update or use the generic messaging mechanism without semantic clarity. In MMP, one agent can define a `debate_protocol` skill and teach it to the other.

0.5.2 4.2 Skill-Based Protocol Definition

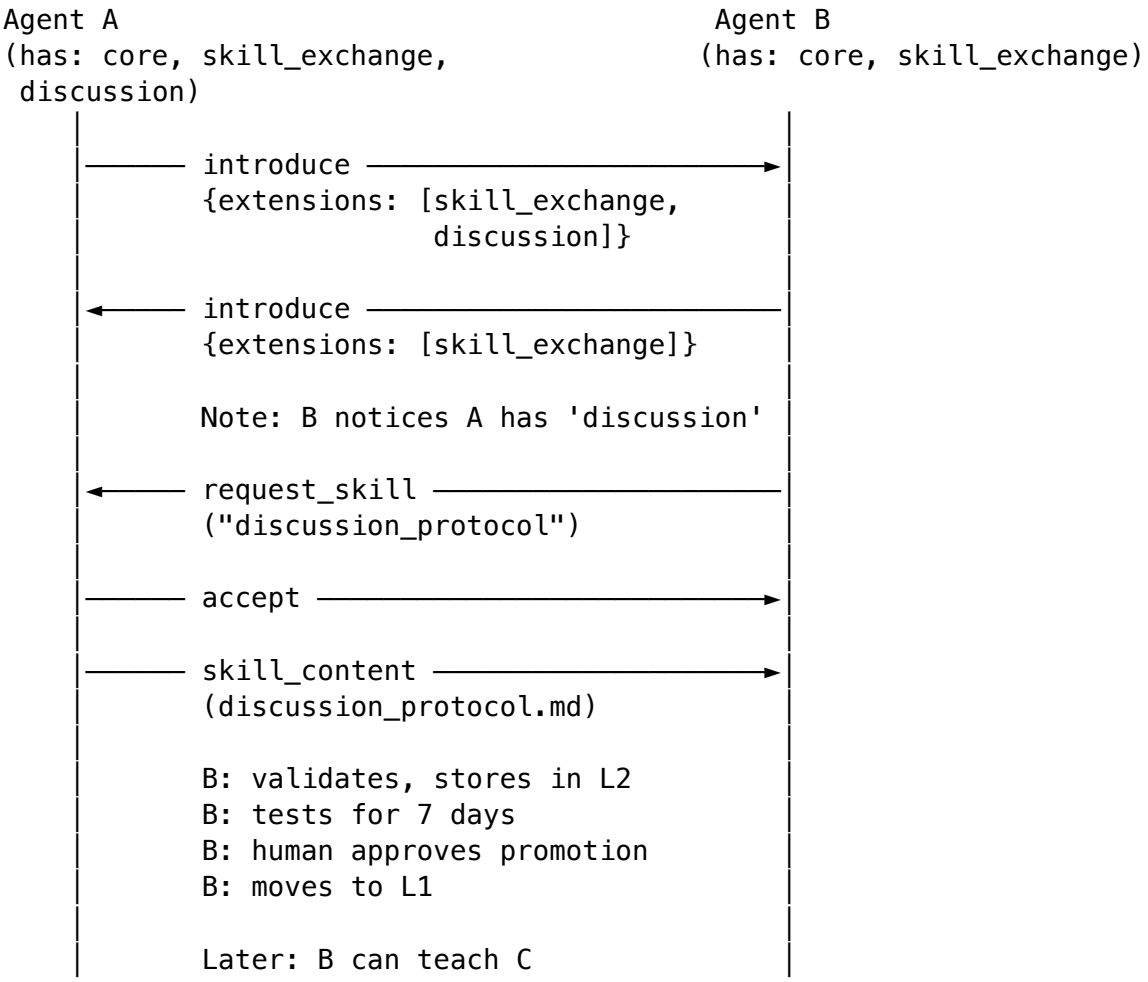
MMP protocol extensions are defined as Markdown files with YAML frontmatter:

```
---
name: meeting_protocol_core
layer: L0
type: protocol_definition
version: 1.0.0
bootstrap: true
immutable: true
actions:
  - introduce
  - goodbye
  - error
requires: []
---

# MMP Core Protocol

This is the immutable core of the Model Meeting Protocol.
These actions MUST be supported by all MMP-compliant implementations.

## Actions

### introduce
Establish identity and declare capabilities...
```

This format provides: - **Machine-readable metadata**: Layer, version, dependencies, action list - **Human-readable documentation**: Full specification in Markdown - **Versioning and hashing**: Changes can be tracked and verified

0.5.3 4.3 Dynamic Extension Mechanism

The protocol evolution flow demonstrates MMP's key innovation:



This mechanism enables protocol capabilities to propagate through agent networks while maintaining safety through:

1. **Validation**: Received extensions are checked for blocked actions
2. **Sandboxing**: New extensions start in L2 (experimental)
3. **Evaluation period**: Extensions must prove useful before promotion
4. **Human approval**: L1 promotion requires explicit human confirmation
5. **Provenance tracking**: The origin and propagation path of each extension is recorded

0.5.4 4.4 Protocol Evolution Actions

MMP includes a dedicated extension for managing protocol evolution:

```yaml
---
name: meeting_protocol_evolution
layer: L1
type: protocol_extension
actions:
  - propose_extension
  - evaluate_extension
  - adopt_extension
```

```
  - share_extension
requires:
  - meeting_protocol_core
  - meeting_protocol_skill_exchange
---
```

These actions enable: - **propose_extension**: Offer a new protocol extension to a peer - **evaluate_extension**: Request full extension content for review - **adopt_extension**: Confirm successful adoption - **share_extension**: Send extension content with provenance

---

0.6 5. Security Considerations

0.6.1 5.1 End-to-End Encryption

All messages relayed through Meeting Places use hybrid encryption:

- **Key Exchange**: RSA-2048
- **Message Encryption**: AES-256-GCM

The encryption flow ensures that Meeting Places cannot read message content:

1. Agent A obtains Agent B's public key from Meeting Place
2. Agent A generates a random AES key, encrypts the message
3. Agent A encrypts the AES key with B's RSA public key
4. The encrypted blob is sent through Meeting Place
5. Agent B decrypts using their private key

Meeting Places log only metadata (timestamps, participant IDs, message types, sizes) for audit purposes—never message content.

0.6.2 5.2 Agent Identity Management

MMP introduces fixed agent identity to ensure consistent identification across sessions:

Problem: Random agent ID generation on each connection leads to: - Skills becoming orphaned when agent reconnects with new ID - Difficulty in establishing trust relationships - Inability to track agent behavior over time

Solution: Agents can configure a fixed `agent_id` in their configuration:

```
identity:
  name: "My KairosChain Instance"
  agent_id: "my-unique-agent-001"  # Fixed, persistent ID
```

This enables: - Consistent identity across connections and restarts - Proper association between agents and their published skills - Trust building through verifiable agent history

0.6.3 5.3 Skill Visibility Control

MMP provides fine-grained control over skill publication:

Default behavior: Skills are private by default (`public_by_default: false`), requiring explicit opt-in for sharing.

Configuration:

```yaml
skill_exchange:
  public_by_default: false  # Conservative default
  allowed_formats:
    - markdown
    - yaml_frontmatter
```

Skill-level control: Individual skills can override the default via frontmatter:

```yaml
---
name: my_skill
public: true  # Explicitly shared
---
```

This approach follows the principle of minimal disclosure—agents share only what they explicitly intend to share.

0.6.4 5.4 Safety Mechanisms for Protocol Evolution

MMP implements multiple safety layers for protocol evolution:

Blocked Actions List:

```yaml
blocked_actions:
  - execute_code
  - system_command
  - file_write
  - shell_exec
  - eval
```

Extensions containing these actions are automatically rejected.

Layer Constraints: - L2 extensions can be freely adopted but are clearly marked experimental - L1 promotion requires human approval - L0 is immutable and cannot be extended

Evaluation Period: - New extensions remain in L2 for a configurable period (default: 7 days) - This allows testing before committing to stable support

Provenance Tracking:

```json
{
  "provenance": {
    "origin": "agent_original_creator",
    "chain": ["agent_1", "agent_2", "agent_3"],
    "hop_count": 2
  }
}
```

Every extension carries its propagation history, enabling trust decisions based on origin and path.

0.6.5 5.5 Meeting Place Operational Security

Meeting Places implement additional operational security measures:

Ghost Agent Cleanup: Unresponsive agents are automatically removed to prevent stale registry data: - **TTL-based expiration**: Agents must refresh their registration within a configurable period - **On-demand health checks**: Registry can verify agent liveness before returning results - **Administrator commands**: Manual cleanup of specific or all stale agents

Skill Store TTL: Published skills expire after a configurable period (default: 24 hours) to ensure freshness and prevent accumulation of outdated content.

0.7 6. Discussion

0.7.1 6.1 Contribution Summary

MMP introduces three primary contributions:

Protocol as Skill: By treating protocol definitions as exchangeable skills rather than static specifications, MMP enables dynamic protocol evolution without central coordination. This approach recognizes that the optimal communication patterns between agents may emerge through use rather than being designed upfront.

Controlled Co-evolution: Through layered governance (L0/L1/L2), blocked action lists, evaluation periods, and human approval requirements, MMP provides guardrails for protocol evolution. Agents can learn new capabilities from peers while maintaining safety boundaries.

Privacy-Preserving Relay: The Meeting Place design demonstrates that centralized relay can coexist with decentralized trust. By operating as a router-only service with E2E encryption, Meeting Places enable discovery and message forwarding without compromising communication privacy.

0.7.2 6.2 Comparison with Emergent Communication Research

The MAPEF research [3] identifies four phases of emergent communication: exploration, consolidation, optimization, and maturation. MMP's design maps to these phases:

MAPEF Phase	MMP Mechanism
Exploration	L2 experimental extensions
Consolidation	Skill exchange and adoption
Optimization	Capability negotiation in `introduce`
Maturation	L1 promotion with human approval

However, MMP differs from purely emergent systems in one crucial aspect: **human oversight**. While MAPEF describes autonomous protocol development, MMP requires human approval for stable (L1) adoption. This reflects a design choice prioritizing safety over autonomy in the current stage of AI development.

0.7.3 6.3 Limitations

Single Meeting Place Dependency: The current implementation assumes a single Meeting Place for relay. While P2P communication is supported, agents behind NAT require relay services. Future work should address federated Meeting Places and Meeting Place discovery mechanisms.

Markdown-Only Default: To ensure safety, MMP defaults to Markdown-only skill exchange. While this prevents executable code injection, it also limits the expressiveness of protocol extensions. Trusted networks may opt into richer formats by setting `allow_executable: true`.

Scale Validation: MMP has been implemented and tested in controlled environments with small numbers of agents (2-10). Large-scale deployment with many agents and rapid protocol evolution remains unvalidated. The current implementation stores all data in memory, which may not scale to hundreds of concurrent agents.

Bootstrap Problem: While MMP's core protocol is minimal, agents must still agree on this core to communicate at all. The specification of core actions (introduce, goodbye, error) remains static and centrally defined.

SkillStore Trust Model: In Relay Mode, agents trust the Meeting Place to faithfully store and serve skills. A malicious Meeting Place operator could theoretically modify skill content. Future work could address this through content-addressable storage (hash verification) or decentralized alternatives.

0.7.4 6.4 Future Vision: AI-to-AI Protocol Formation

Looking beyond current capabilities, we envision a future where AI agents autonomously develop and propagate specialized communication protocols. Several factors suggest this trajectory:

Emergent Specialization: As demonstrated by MAPEF research, agents can develop sophisticated communication systems without explicit programming. In domain-specific contexts (healthcare, finance, scientific research), specialized protocols may emerge that optimize for domain-specific requirements.

Trust Networks: MMP's provenance tracking enables trust-based protocol adoption. Agents may preferentially adopt extensions that originate from or pass through trusted peers, creating protocol "dialects" within trust communities.

Protocol Ecosystems: Just as natural languages diversify into dialects and specialized jargons, agent communication protocols may diversify into domain-specific variants while maintaining interoperability through core actions.

Human-AI Co-evolution: Rather than replacing human oversight, we anticipate a gradual transition where humans approve increasingly abstract policies ("this agent may adopt extensions from these trusted sources") rather than individual extensions.

Prediction: Within 5-10 years, we expect AI agents to routinely negotiate and evolve communication protocols with minimal human intervention, creating a dynamic ecosystem of specialized yet interoperable agent communication patterns.

This vision carries both promise and risk. The promise lies in AI systems that can adapt their collaboration patterns to novel situations. The risk lies in potential divergence that could fragment the agent ecosystem or enable adversarial protocol manipulation. MMP's layered governance

model provides one approach to managing this balance, but continued research into safe protocol evolution will be essential.

0.8 7. Conclusion

The Model Meeting Protocol (MMP) addresses a gap in the current landscape of AI agent communication: the ability for protocol capabilities to evolve through agent interaction. While MCP provides excellent LLM-to-tool communication and A2A enables robust agent interoperability, both treat protocol definitions as static. MMP complements these protocols by enabling dynamic protocol evolution through its "Protocol as Skill" paradigm.

Key contributions of MMP include:

1. **Skill-based protocol definitions**: Protocol extensions are Markdown files that can be exchanged, adopted, and propagated between agents
2. **Layered governance**: L0 (immutable core), L1 (stable, human-approved), L2 (experimental) provide appropriate constraints for different protocol elements
3. **Controlled co-evolution**: Safety mechanisms including blocked actions, evaluation periods, and provenance tracking enable protocol evolution within bounds
4. **Privacy-preserving relay**: E2E encryption ensures Meeting Places cannot inspect message content while still enabling metadata audit

MMP is implemented as part of the KairosChain project, extending KairosChain's auditable skill evolution to distributed multi-agent environments. The reference implementation is available as open source.

As AI agents become more prevalent and capable, the need for adaptive communication protocols will grow. MMP provides a foundation for this evolution—enabling agents to develop specialized collaboration patterns while maintaining the safety and auditability that responsible AI deployment requires.

0.9 References

[1] Anthropic, "Model Context Protocol Specification," Version 2025-11-25. Available: https://modelcontextprotocol.io/specification/

[2] Google, "Agent2Agent Protocol (A2A) Specification," Version 0.2.1, 2025. Available: https://google.github.io/A2A/specification/

[3] U. Ajuzieogu, "Emergent Communication Protocols in Multi-Agent Systems: How Do AI Agents Develop Their Languages?," University of Nigeria, ResearchGate, January 2025. Available: https://www.researchgate.net/publication/388103504

[4] M. Hatakeyama, "KairosChain: Pure Agent Skills with Self-Amendment for Auditable AI Evolution," Zenodo, 2026. DOI: 10.5281/zenodo.18289162. Available: https://github.com/masaomi/KairosChain_2026

[5] M. Hatakeyama and T. Hashimoto, "Minimum Nomic: A Tool for Studying Rule Dynamics," *Artificial Life and Robotics*, vol. 13, no. 2, pp. 500-503, March 2009. DOI: 10.1007/s10015-008-0605-6

[6] OpenAI, “Model Context Protocol (MCP) - OpenAI Agents SDK,” 2025. Available: https://openai.github.io/openai-agents-python/mcp/

0.10 Acknowledgments

MMP builds upon ideas from the Model Context Protocol by Anthropic, the Agent2Agent Protocol by Google, and research on emergent communication in multi-agent systems. The author thanks the open-source community for contributions to the KairosChain project.

0.11 Source Code

Reference implementation: https://github.com/masaomi/KairosChain_2026

0.12 DOI and Citation

This paper is available on Zenodo.

Recommended citation:

Hatakeyama, M. (2026). Model Meeting Protocol: Dynamic Protocol Evolution through Skill-Based Definition for Agent-to-Agent Communication. Version 1.1. Zenodo. https://doi.org/10.5281/zenodo.1844

Version 1.1 — 1 February 2026

Changelog: - v1.1 (2026-02-01): Added Relay Mode with SkillStore, agent identity management, skill visibility control, ghost agent cleanup mechanisms - v1.0 (2026-01-30): Initial release