NANDA Registry Architecture: Internet of Agents

Collaborative Infrastructure for Agent-Based Communication

Ayush Chopra, Pradyumna Chari, Abhishek Singh, Ramesh Raskar
Project NANDA, MIT Media Lab

Executive Summary: Project NANDA is building the foundational infrastructure for an Internet of Agents - a new communication paradigm where autonomous agents coordinate on behalf of humans to improve efficiency and reduce coordination overhead. Our live chat service at chat.nanda-registry.com demonstrates real-world agent interactions, while waitlist.nanda-registry.com allows early adopters to claim their agent identities in the NANDA namespace. The core insight is that human-to-human coordination is increasingly mediated by digital systems, but current approaches require humans to manually translate intent, search for options, and negotiate outcomes. An Internet of Agents handles this coordination layer automatically while keeping humans in control of decisions.

Part 1: Architecture Overview: NANDA's three-tier agent architecture mirrors proven internet infrastructure patterns while enabling new capabilities for dynamic, context-aware communication. An example scenario is visualized in Figure 1 and used to explain the architecture below.

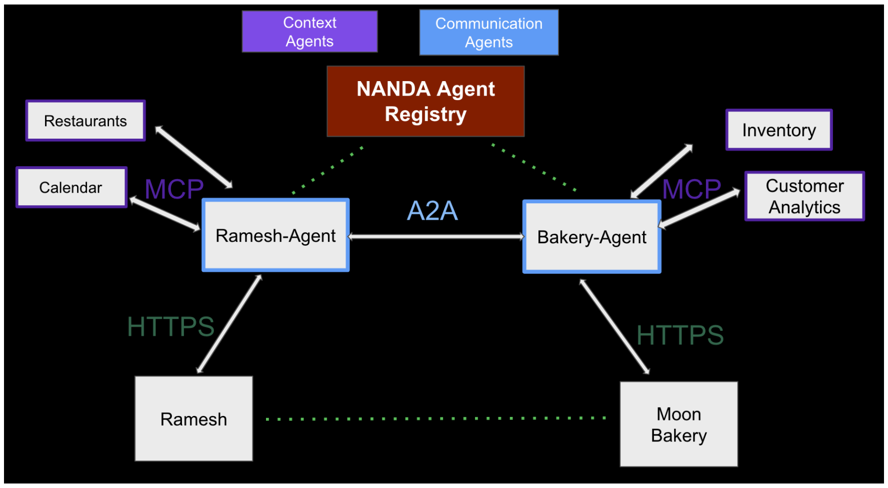

Figure 1: NANDA Architecture: *Ramesh sends "@moonbakery get me the usual cake for daughter's birthday" to his Client Agent. This simple request triggers coordination across multiple agent types registered in the NANDA Agent Registry: Communication Agents (blue)*

orchestrate the conversation via A2A protocol, while Context Agents (purple) provide specialized capabilities via MCP protocol. The Registry verifies agent identities and capabilities, enabling secure multi-agent coordination that transforms a casual human request into a fully contextualized business interaction.

1. Client Agents: The Identity Layer

Client Agents represent real-world entities (people, businesses) and serve as their verified digital identity in the agent ecosystem. Like domain names in traditional internet infrastructure, they provide a globally unique namespace (@ramesh, @moonbakery) that can be claimed and verified.

Key Characteristics:

- Communicate via standard HTTPS protocol for maximum compatibility
- Stored in distributed Client Registry for global verification
- Act as the authoritative source for routing agent communications
- Enable trusted identity without requiring centralized authentication

Example: Ramesh sends a simple message to his Client Agent: @moonbakery get me the usual cake for daughter's birthday. His Client Agent (@ramesh) establishes the authenticated session with Moon Bakery's Client Agent (@moonbakery) and routes this request through the appropriate Communication Agents.

2. Communication Agents: The Orchestration Layer

Communication Agents handle the bidirectional conversation flow between Client Agents. They're client-specific but not task-specific, meaning Ramesh might have separate Communication Agents for work, family, and personal contexts, each understanding his preferences and communication style for those domains.

Key Characteristics:

- Use A2A (Agent-to-Agent) protocol for rich, contextual communication
- Orchestrate complex multi-turn conversations
- Client-specific rather than task-specific (one per context/relationship)
- Handle context gathering and response formatting

Example: Ramesh-Agent receives the cake request (@moonbakery get me the usual cake for daughter's birthday), recognizes it needs timing context for "daughter's birthday," queries the Calendar Context Agent to find the specific date, discovers Ramesh's usual preferences, then formulates a detailed request to Bakery-Agent: "Please prepare Ramesh's usual chocolate cake for pickup on Saturday, June 1st for his daughter's birthday party."

3. Context Agents: The Capability Layer

Context Agents provide specialized, unidirectional services via the MCP (Model Context Protocol). They're task-specific and can combine multiple data sources, tools, and APIs to deliver rich context to Communication Agents.

Key Characteristics:

- Execute specific tasks (calendar lookup, inventory check, customer analytics)
- Can be registered directly (@pizzahut) or through other registries (@salesforce:dominos)
- Unidirectional: provide context when requested, don't initiate conversations
- Highly composable and specialized

Example: When Bakery-Agent needs to fulfill an order, it consults Customer-Analytics Agent (what does Ramesh usually order?) and Inventory Agent (what's available Saturday?) to provide informed options.

Part 2: Registry Infrastructure: NANDA's registry architecture follows a minimal core registry design that separates static identifiers from dynamic routing and metadata. Full technical details are available in our paper on NANDAs Verified AgentFacts by Raskar et al.

Key Architecture Principles

1. Lean Registry Core: Stores only essential static metadata (agent IDs, credential pointers, AgentFacts URLs) to minimize update frequency
2. AgentFacts Cards: Self-describing, cryptographically signed metadata documents that agents can update independently without registry modifications
3. Multi-Path Resolution: Supports direct access, privacy-preserving third-party hosting, and adaptive routing for different operational needs
4. Distributed Deployment: Federated registry model enabling enterprise, industry consortium, and public deployment scenarios

Finally, we have also designed our **NANDA Capabilities Schema**. This provides high-resolution capability definitions (over 32K dimensional basis) consistent with US Census skills classifications, enabling unified management of human and agent workforce capabilities. This is demonstrated by our Iceberg Index (iceberg.mit.edu) which is already enabling real-world impact of the NANDA Registry to help multiple US states quantify the emerging impact of internet of agents.

Part 3: Open Problems - Collaboration Opportunities

1. Content Generation Networks (CGNs): The Next Evolution of CDNs Traditional CDNs optimize delivery of static content, but the Internet of Agents requires dynamic content generation at the edge. When Ramesh's agent requests cake options, the response isn't retrieved from storage - it's generated by combining real-time inventory, customer preferences, scheduling constraints, and business logic. This creates fundamentally new infrastructure requirements.

Content Generation Networks would deploy lightweight agent processing capabilities at edge locations, enabling:

- **Ultra-low Latency Response Generation**: Agent conversations require sub-100ms response times for natural interaction flow
- **Context-Aware Edge Processing**: Customer Analytics and Inventory Context Agents could run at regional edges, reducing round-trips to origin servers
- **Dynamic Load Balancing**: Route requests to the most capable available Context Agent, not just the nearest one
- **Intelligent Caching of Generated Content**: Cache frequently requested agent capabilities and common conversation patterns

The parallel to CDN evolution is clear: just as static content delivery moved to the edge, dynamic agent-generated content will follow. Akamai's edge infrastructure provides the perfect foundation for CGN deployment.

2. Protocol Evolution: Beyond Request-Response Paradigms: Current agent interactions are forced into HTTP request-response patterns that don't match their natural communication needs. When Ramesh-Agent needs calendar, preferences, and inventory data, it makes three separate HTTP calls and waits for responses - but this is inherently a multicast query that should be answered collaboratively. Agent conversations also maintain persistent context across multiple interactions, but TCP's connection model maps poorly to "conversation sessions" that pause, resume, and involve multiple participants. Additionally, agent capabilities change in real-time, requiring routing decisions per-message rather than per-connection.

Solution Direction: Explore MQTT-style pub/sub protocols for agent communication, enabling efficient multicast queries and persistent conversation state management across distributed agent networks.

3. Enterprise DNS for Dynamic Agent Capabilities: Traditional DNS resolves static domain names to IP addresses, but agent interactions need dynamic capability resolution - @moonbakery should resolve to current capabilities, supported protocols, real-time load, and trusted context agents. Unlike web services with predictable interfaces, agent capabilities change based on context, availability, and learned preferences. Current DNS infrastructure can't handle capability queries like "which inventory agents near Boston can check cake availability for Saturday pickup?" or trust chain resolution for cross-registry agent verification.

Solution Direction: Design capability-aware DNS that combines traditional name resolution with real-time agent capability discovery, load balancing, and distributed trust verification.

4. Stateful Edge Computing for Agent Conversations: Traditional CDNs cache static content near users, but agent interactions require stateful processing at the edge. When multiple Context Agents collaborate on a response, current infrastructure forces round-trips to origin servers - but conversations like Ramesh's cake order should be processed entirely at regional edge locations using local Calendar, Customer Analytics, and Inventory agents. The challenge is maintaining conversation coherence when Context Agents are distributed across different edge locations while ensuring consistent agent state and capabilities.

Solution Direction: Develop stateful edge architectures that deploy lightweight Context Agents regionally while maintaining conversation coherence and cross-edge agent coordination protocols.