# Abstract Agentic System Architecture Design

This document outlines the high-level architecture for a flexible, message-driven multi-agent system, capable of integrating both software and human agents. The core principle is **direct, asynchronous communication via dedicated mailboxes**, with agents actively listening for incoming messages.

## 1. The Agent (Node)

The fundamental building block of the system. Each Agent is an instance of a generic class, its specific role defined by its configuration and internal logic.

### 1.1. Core Attributes:

- **agent_id**: A unique identifier for the agent (e.g., "Agent1", "Puppetmaster", "HumanAgent_JohnDoe"). This serves as its network address.
- **input_dir (Mailbox)**: A dedicated, unique file system directory where the agent receives all its incoming messages. This is the agent's primary ingress point.
- **processed_dir**: A directory for archiving messages after they have been successfully processed by the agent, preventing re-processing and maintaining a history.

### 1.2. Core Capabilities:

- **Active Listening**: The agent continuously monitors its input_dir for new message files. This implies a long-running process (daemon) utilizing file system event monitoring or efficient polling.
- **Internal Logic (process_input Method)**: This is the agent's "brain." When a new message arrives in its mailbox, the agent reads it and executes specific logic based on the message's content (type, payload). This method is overridden by subclasses or configured to define the agent's unique behavior (e.g., objective framing, task execution, human review, feedback generation).
- **Communication Capability (_send_message Method)**: The agent can send messages to any other known agent in the network. This involves:
    1. Loading (or refreshing) the Central Mailbox Registry to get the recipient's status and mailbox_path.
    2. Constructing a standardized message file.
    3. Writing the message file directly into the recipient's mailbox.
- **Status Reporting (Heartbeat)**: Each active software agent is responsible for periodically updating its own status in the Central Mailbox Registry to reflect its current operational state.

## 2. The Central Mailbox Registry (Network Directory

# Service)

A globally accessible mechanism that maps agent_ids to their respective input_dir (mailbox) paths and provides their current operational status.

## 2.1. Implementation:

- **mailbox_map.json**: A JSON file located in a well-known, central directory (e.g., /var/lib/agent_mailboxes/mailbox_map.json).
- **Content Structure**: Each entry for an agent is now an object containing its details:

```
{
    "Agent1": {
        "mailbox_path": "/path/to/agent1_in",
        "status": "active",
        "last_heartbeat": "2025-07-20T17:00:00Z",
        "last_processed_task_id": "abc123xyz",
        "description": "Orchestrates tasks and evaluates results."
    },
    "Agent2": {
        "mailbox_path": "/path/to/agent2_in",
        "status": "inactive",
        "last_heartbeat": "2025-07-20T16:00:00Z",
        "last_processed_task_id": null,
        "description": "Executes specific computational tasks."
    },
    "Puppetmaster": {
        "mailbox_path": "/path/to/puppetmaster_in",
        "status": "active_human",
        "last_heartbeat": "2025-07-20T17:05:00Z",
        "last_processed_task_id": "def456uvw",
        "description": "Human oversight and high-level directive
issuance."
    }
}
```

- **Key Fields within Agent Entry**:
  - **mailbox_path**: (String) The absolute file system path to the agent's input directory.
  - **status**: (String) The current operational state of the agent.
    - active: Running and processing messages.
    - inactive: Not running, or intentionally paused.
    - error: Running but encountered a critical error.
    - maintenance: Temporarily offline for updates/maintenance.
    - active_human: Specifically for human agents who are actively monitoring their mailbox.
  - **last_heartbeat**: (Timestamp, ISO 8601 format) The last time the agent updated its status in the registry. Crucial for detecting unresponsive agents.
  - **last_processed_task_id**: (String, Optional) The ID of the last task successfully

processed by the agent. Useful for auditing and tracking progress.
    - **description**: (String, Optional) A brief human-readable explanation of the agent's purpose or role.
- **Role**: Provides the necessary lookup service for agents to address messages directly to their intended recipients, and offers **real-time operational awareness** of the network's state.

### 2.2. Mechanism for Updating Status:

- **Self-Reporting**: Each active software Agent instance is responsible for periodically updating its own entry in the mailbox_map.json (e.g., every 60 seconds) to reflect its status and last_heartbeat. This requires careful implementation of file locking to prevent data corruption during concurrent writes.
- **Human Agents**: Human agents update their status manually or via simple utility scripts provided by the system.
- **Monitoring**: A dedicated "Registry Monitor" (potentially a specialized Agent or a function within the Puppetmaster's tooling) can periodically check last_heartbeat values across the registry and mark agents as inactive or error if heartbeats cease or other issues are detected.

## 3. The Message (Packet)

The standardized unit of information exchanged between agents. These are JSON files written to mailboxes.

### 3.1. Key Attributes:

- **sender_id**: The agent_id of the message originator.
- **recipient_id**: The agent_id of the intended receiver. (Crucial for the receiving agent's process_input to confirm it's the intended recipient).
- **task_id**: A unique identifier for a specific task or conversational thread, allowing agents to track multi-step interactions.
- **type**: Categorizes the message's purpose (e.g., "request", "result", "feedback", "directive", "approval", "notification", "query"). This informs the receiving agent's processing logic.
- **payload**: The actual data or content of the message, highly variable based on the type.
- **timestamp**: ISO 8601 formatted timestamp of message creation for chronological ordering and latency analysis.
- **File Naming Convention**: [sender_id]_[task_id]_[timestamp]_[type].json (e.g., Agent1_abc123_20250720163000_request.json).

## 4. System Architecture as a Graph

The system can be visualized as a **directed graph**:

- **Nodes**: Each Agent instance.
- **Edges**: Represent potential communication paths. A message sent from Agent A to Agent B establishes a logical directed edge A → B. The Central Mailbox Registry defines all possible nodes and their addresses, enabling any node to initiate an edge to any other

node.

# 5. Integration of Human Agents

Human participants (including the Puppetmaster) are integrated as Agent instances within this framework:

- They have their own agent_id and dedicated input_dir (mailbox).
- They "actively listen" by monitoring their mailbox manually or via a simple notification tool.
- Their "processing" involves human cognition, decision-making, and manual creation/placement of response messages into other agents' mailboxes, adhering to the Message structure and using the Central Mailbox Registry.

# 6. Abstract Principles

- **Decentralized Processing, Centralized Discovery & Status**: Agents operate independently, but find each other and report their health via a shared, dynamic registry.
- **Asynchronous, Message-Driven**: Communication is entirely through messages written to mailboxes, processed when the receiving agent "sees" them.
- **Plug-and-Play Extensibility**: New agents (software or human) can be added by defining their ID, creating mailboxes, registering in the map, and implementing their process_input logic.
- **Role Agnostic Base**: The core Agent class is generic; specific roles (Orchestrator, Executor, Validator, Human Reviewer) emerge from the configured connections and process_input logic of each instance.
- **Enhanced Resilience**: The status mechanism allows for more intelligent message routing and error handling based on recipient availability.