SOFTWARE SPECIFICATION AUTONOMOUS AI AGENTS FOR COLLABORATIVE DECISION-MAKING

Version: 1.0.0 | **Date:** October 26, 2023 | **Status:** Final Draft

1. SYSTEM OVERVIEW

This specification defines the architecture for a multi-agent system (MAS) capable of autonomous operation and collaborative decision-making. The system enables independent AI agents to perceive their environment, communicate via established protocols, form consensus, and execute coordinated actions to solve complex problems.

2. CORE COMPONENTS

- **Agent Engine:** The runtime environment managing agent lifecycle, state, and local execution loop.
- **Decision Framework:** Logic modules enabling agents to evaluate options using utility functions and probabilistic reasoning.
- **Collaboration Layer:** Messaging infrastructure implementing ACL (Agent Communication Language) for negotiation and consensus.
- **Knowledge Base:** Distributed storage for shared world state, ontologies, and historical decision logs.

3. FUNCTIONAL REQUIREMENTS

- **Autonomy:** Agents must initiate actions without direct human intervention based on assigned goals.
- **Scalability:** System must support dynamic registration and deregistration of 100+ concurrent agents.
- **Interoperability:** Agents must exchange structured messages (JSON/Protobuf) adhering to FIPA standards.
- **Conflict Resolution:** Built-in mechanisms for voting or bidding to resolve resource contention.

- **Auditability:** Full traceability of decision paths and inter-agent communication logs.

4. TECHNICAL ARCHITECTURE

The system follows a microservices-based event-driven architecture.

- **Language:** Python 3.10+ (Core Logic), Go (Messaging Layer).
- **Communication:** gRPC for internal agent RPC; RabbitMQ/Kafka for asynchronous event bus.
- **Persistence:** Redis (Hot State), PostgreSQL (Transactional Data), Vector DB (Semantic Memory).
- **Deployment:** Docker containers orchestrated via Kubernetes.

5. DATA MODELS

Note: Presented in pseudo-schema format.

5.1 Agent Entity

```
struct Agent {
    id: UUID
    role: Enum(LEADER, WORKER, OBSERVER)
    status: Enum(IDLE, BUSY, OFFLINE)
    capabilities: List<String>
    utility_function: Function
    local_state: Map<String, Any>
}
```

5.2 Task Entity

```
struct Task {
    id: UUID
    priority: Integer(1-10)
    constraints: Map<String, Any>
    deadline: Timestamp
    required_capabilities: List<String>
    assigned_agents: List<UUID>
}
```

5.3 Decision Record

```
struct Decision {
    id: UUID
    context_hash: String
    options_considered: List<Option>
    selected_option: Option
    rationale: String
    consensus_score: Float
    timestamp: Timestamp
}
```

6. APIS & INTERFACES

6.1 Agent Control API (gRPC)

- `RegisterAgent(AgentConfig) returns (AgentID)`
- `UpdateGoal(AgentID, GoalDefinition) returns (Ack)`
- `GetAgentState(AgentID) returns (StateSnapshot)`

6.2 Collaboration Interface (Message Bus)

- **Topic:** `agent.negotiation.{proposal_id}`
  - Payload: `Proposal | CounterProposal | Accept | Reject`
- **Topic:** `system.broadcast`
  - Payload: `GlobalStateUpdate | EmergencyHalt`

7. AGENT CAPABILITIES

- **Perception:** Agents poll environment state or subscribe to event streams to update their local Knowledge Graph.
- **Reasoning (LLM Integrated):** Usage of chain-of-thought prompting via LLM APIs to generate high-level plans.
- **Learning:** Reinforcement Learning (RL) module updates utility weights based on task success/failure feedback.
- **Protocol Adherence:** Strict enforcement of turn-taking and message formatting during negotiation phases.

8. DECISION-MAKING FRAMEWORK

8.1 Individual Decision

Agents maximize local utility: `U(action) = w1*P(success) + w2*resource_cost + w3*time_efficiency`.

8.2 Collaborative Consensus (Voting)

For group decisions, a weighted voting mechanism is employed:

1. **Proposal Phase:** Leader agent broadcasts a plan.
2. **Evaluation Phase:** Worker agents simulate outcome locally.
3. **Voting Phase:** Agents cast votes (Yes/No/Abstain) weighted by their domain expertise score.
4. **Execution Phase:** If `weighted_yes > threshold`, plan executes. Otherwise, back to Proposal.

9. PERFORMANCE REQUIREMENTS

Metric	Target	Condition
Agent Response Latency	< 200ms	95th percentile, internal logic processing
Consensus Time	< 2 seconds	Groups of up to 10 agents
Throughput	10,000 msgs/sec	Message bus capacity
Recovery Time	< 5 seconds	Agent restart after crash

10. SECURITY & COMPLIANCE

- **Identity:** Mutual TLS (mTLS) for all inter-agent communication.
- **Authorization:** Role-Based Access Control (RBAC) enforced at the API Gateway level.
- **Sandboxing:** Agents execute logic in isolated containers with restricted network egress.
- **Data Protection:** All persistent data encrypted at rest (AES-256).
- **Governance:** "Kill switch" functionality to immediately suspend all agent autonomous actions.