Gactory Agent Simulation Scenarios for AI and Unity Agents in RabbitMQ

Overview

This document describes the simulation scenarios for the communication between AI agents and Unity agents in a factory metaverse environment using RabbitMQ. The setup includes assessment, quality, and master agents for both AI and Unity, and scenarios for message exchange and logging.

Scenario Descriptions

Scenario 1: Batch0001 - All Steps OK

- **Steps:** 10 steps with timestamps 20 to 40 seconds apart.
- **Flow:**
  i. Assessment Agent processes the log and confirms all steps are OK.
  ii. Sends a message to Quality AI agent with the batch number.

Scenario 2: Batch0002 - Missing Step

- **Steps:** 9 steps with one step missing.
- **Flow:**
  i. Assessment Agent processes the log and detects a missing step.
  ii. Sends a message to Quality AI agent with the batch number.
  iii. Quality AI agent instructs its Unity counterpart to inform the test bench about the missing step.

Scenario 3: Batch0003 - Abnormal Time Gap

- **Steps:** 10 steps with a 20-minute gap between step 8 and step 9.
- **Flow:**
  i. Assessment Agent processes the log and detects the abnormal time gap.
  ii. Sends a message to Quality AI agent with the batch number.
  iii. Sends a message to Master AI agent with the batch number.
  iv. Quality AI agent instructs its Unity counterpart to inspect the relevant part.
  v. Master AI agent instructs its Unity counterpart to query the test bench and logs a message to call the supervisor via MS Teams.

Exchange and Bindings

What is an Exchange?

In RabbitMQ, an exchange is a message routing agent that routes messages to queues based on routing keys and exchange types. When a message is sent to an exchange, the exchange decides which queue(s) to deliver the message to.

Types of Exchanges

- **Direct Exchange:** Routes messages with a specific routing key to the queue(s) that are bound with the same key.
- **Fanout Exchange:** Routes messages to all of the queues bound to it, regardless of routing keys.
- **Topic Exchange:** Routes messages to queues based on wildcard matches between the routing key and the routing pattern specified in the queue binding.
- **Headers Exchange:** Uses message headers to route messages.

What is a Binding?

A binding is a link between an exchange and a queue. It tells the exchange to send messages to the queue based on certain criteria (routing keys). Bindings can be configured to route messages from exchanges to queues in various ways.

Our Setup

In this simulation, we use two types of exchanges:

1. **agent_exchange (Direct Exchange):**
    - Purpose: Routes messages to specific agents based on routing keys.
    - Bindings: Each agent's queue is bound to this exchange with a specific routing key (e.g., `assessment`, `ai_quality`, `ai_master`, `unity_quality`).
2. **log_exchange (Fanout Exchange):**
    - Purpose: Broadcasts messages to all bound queues for logging purposes.
    - Bindings: The `log_queue` is bound to this exchange to receive all messages for auditing and live display.

Files and Their Functions

setup_rabbitmq.py

- **Purpose:** Sets up the necessary RabbitMQ exchanges, queues, and bindings for the simulation.
- **Details:** Creates the `agent_exchange` for direct communication and `log_exchange` for logging, and binds the appropriate queues to these exchanges.

simulate_scenarios.py

- **Purpose:** Simulates the three scenarios by sending structured messages to the Assessment Agent via RabbitMQ.
- **Details:** Reads scenarios from a `scenarios.json` file and sends messages every 10 seconds to simulate different batch processes.

AssessmentAIAgent.py

- **Purpose:** Processes incoming messages, checks for missing steps or abnormal time gaps, and forwards relevant information to Quality AI and Master AI agents.
- **Details:** Extracts logs and batch information, performs checks, and sends structured messages to other agents based on the findings.

QualityAIAgent.py

- **Purpose:** Processes messages from the Assessment Agent to handle quality-related issues.
- **Details:** Instructs the Unity Quality agent to take specific actions based on the assessment results.

MasterAIAgent.py

- **Purpose:** Processes messages from the Assessment Agent to handle master-level issues.
- **Details:** Instructs the Unity Master agent to take specific actions, such as querying the test bench and logging calls to the supervisor.

app.py

- **Purpose:** provide a web based view on Queues and simulation run

log_observer.py

- **Purpose:** Logs all communications happening through the `log_exchange` for auditing and live display.
- **Details:** Consumes messages from the log queue and writes them to a log file, while also printing them to the console.

Running the Simulation

1. **Set up RabbitMQ exchanges and queues:**
   Run the `setup_rabbitmq.py` script to create necessary exchanges and queues.

2. **Start the agents:**
   Run `AssessmentAIAgent.py`, `QualityAI.py`, and `MasterAIAgent.py` in separate terminals.

3. **Start the log observer:**
   Run the `log_observer.py` script to start logging communications.

4. **Simulate the scenarios:**
   Run the `simulate_scenarios.py` script to simulate the batch processes and trigger the agents' actions.

This setup simulates the communication between AI and Unity agents in a factory metaverse environment, handling quality checks, master-level issues, and logging all communications for auditing purposes.

Queues

Binding route : Queue

- unity_assessment: unity_assessment_queue:
- ai_assessment: ai_assessment_queue:
- unity_quality: unity_quality_queue:
- ai_quality: ai_quality_queue:
- unity_master: unity_master_queue:
- ai_master: ai_master_queue:
- call_ms_teams: call_ms_teams_queue:
- DigitalPokaYoke_bot: DigitalPokaYoke_bot_queue

Environment Setup

Create a `.env` file with the following content:

```
# RabbitMQ configuration
RABBITMQ_HOST=68.221.122.91
RABBITMQ_PORT=5672

# Unity Agents credentials
UNITY_USER=UnityAgent
UNITY_PASS=

# AI Agents credentials
AI_USER=AIAgent
AI_PASS=
```