Orchestrator Agent Technical Specification

Created by Kovuri, Ramesh (CWR), last modified 10 minutes ago

FOR DEVELOPMENT TEAM REFERENCE

IMPORTANT: All API development must strictly follow FEPOC API guidelines including naming conventions, security standards, authentication protocols, and response formats. This specification provides functional requirements - implementation must align with established FEPOC standards.

API
Guidelines: BCBSA_Web_Services_API_Guidelines_2022.docx

Overview

Orchestrator Agent is a Python-based intelligent routing service that uses LangGraph for workflow orchestration. It dynamically selects and invokes appropriate tools based on user prompts using LLM-powered decision making.

Architecture

- **Framework**: Python with LangGraph
- **Deployment**: Amazon EKS (Kubernetes) / AgentCore (In Future)
- **Web Server**: Python HTTP Server (FastAPI recommended, may need TA, not needed with AgentCore)
- **LLM Integration**: For tool selection and routing decisions
- **Configuration**: YAML-based tool definitions (with future MCP support for dynamic tool discovery

Tool Configuration

Current Implementation: YAML

File: `tools-config.yaml`

Note: YAML-based tool definitions provide static configuration for PI2 release. Future versions may support Model Context Protocol (MCP) for dynamic tool discovery and registration.

Future Enhancement: MCP Integration

Model Context Protocol (MCP) support planned for future releases to enable:

- **Dynamic Tool Discovery**: Automatic detection and registration of available tools

- **Dynamic Tool Discovery**: Automatic detection and registration of available tools
- **Runtime Tool Registration**: Tools can register/unregister without service restart
- **Standardized Tool Interface**: Consistent tool communication protocol
- **Enhanced Scalability**: Simplified addition of new tools and capabilities

Sample tools-config.yaml

```yaml
tools:
  - name: "IBTAgent"
    description: "Handles insurance benefit and coverage inquiries"
    endpoint: "https://ibt-service.internal/api/v1"
    capabilities:
      - "benefit inquiries"
      - "coverage questions"
      - "policy information"
    parameters:
      required:
        - "userPrompt"
        - "userName"
      optional:
        - "policyNumber"
        - "memberType"
    examples:
      - prompt: "What are my dental benefits?"
        reasoning: "User asking about specific benefit coverage"
      - prompt: "Is physical therapy covered?"
        reasoning: "Coverage inquiry for specific service"

  # Future tools can be added here
  - name: "ClaimsAgent"
    description: "Processes claims-related queries"
    endpoint: "https://claims-service.internal/api/v1"
    capabilities:
      - "claim status"
      - "claim submission"
      - "claim history"
    parameters:
      required:
        - "userPrompt"
        - "userName"
      optional:
        - "claimNumber"
    examples:
      - prompt: "What's my claim status?"
        reasoning: "User requesting claim information"
```

API Endpoints

1. Process Query/Prompt Endpoint

Endpoint: `POST /invocations  (Naming is compatible with AgentCore)`

Purpose: Receives user prompt from DXAIService InvokeAgent endpoint, determines appropriate tool, and returns processed response.

Request Body (Matches DXAIService InvokeAgent Request):

- `userPrompt` (string) - User's question or request (passed from DXAIService)
- `sessionId` (string) - Conversation session ID (passed from DXAIService)
- `context` (object) - Additional context information
    - `userName`: String - User identifier (passed from DXAIService)
    - `userType`: String - User type (passed from DXAIService)
    - `source` (string) - Request origin (passed from DXAIService)
    - promptId - (passed from DXAIService)

2. Health Check Endpoint

Endpoint: `GET /ping (Naming is compatible with AgentCore)`

Purpose: Service health monitoring and availability status.

LangGraph Workflow Design

Workflow Steps:

1. **Input Validation** - Validate request format and required fields
2. **Query Analysis** - Use LLM to understand user intent
3. **Tool Selection** - Determine best tool based on capabilities and query
4. **Guard Rails Check** - Validate confidence score and tool availability
5. **Tool Invocation** - Call selected tool with formatted parameters OR return generic message
6. **Response Processing** - Format and return tool response
7. **Error Handling** - Handle failures and fallback scenarios

Tool Selection Logic

PI2 Implementation Note

Important: For PI2 release, only the IBTAgent tool will be available. While LLM-based tool selection is not strictly necessary with a single tool, implementing this architecture provides several benefits:

- **Future-Proof Foundation:** Establishes the framework for multiple tools in future releases
- **Scalable Architecture**: Enables easy addition of new tools (ClaimsAgent, NavigationAgent, etc.) without major refactoring
- **Consistent Interface**: Maintains uniform prompt/query processing regardless of tool count
- **Testing & Validation:** Allows validation of LLM decision-making logic before multi-tool scenarios

PI2 Behavior: All valid queries will route to IBTAgent, with LLM providing confidence scoring and query understanding validation. Invalid or out-of-scope queries will trigger guard rails with generic responses.

LLM Decision Making Process:

Query Analysis Steps:

1. Parse user query for intent and key topics
2. Map query requirements to available tool capabilities
3. Evaluate parameter availability and completeness
4. Calculate confidence score based on matching accuracy
5. Apply guard rails and validation checks
6. Return tool selection with reasoning and confidence score

Decision Criteria:

- Tool capability alignment with query requirements
- Availability of required parameters for tool execution
- Query complexity and tool expertise match
- Historical success patterns for similar queries

LLM Prompt Template: Sample

```
Given the user query: "{user_query}"

Available tools and their capabilities:
{tools_info}

Analyze the query and select the most appropriate tool. Consider:
1. Query intent and topic
2. Tool capabilities and expertise
3. Required parameters availability

Respond with:
- Selected tool name (or "NO_TOOL" if no suitable tool found)
- Reasoning for selection
- Confidence score (1-10)

If confidence is below 7 or no tool matches, respond with "NO_TOOL".
```

Guard Rails & Validation

Confidence Threshold Logic:

- Minimum confidence score of 7.0 out of 10.0 required for tool selection
- Scores below threshold trigger fallback response mechanism
- Confidence calculation based on query-tool capability matching accuracy
- Multiple validation layers ensure appropriate tool routing

Fallback Strategy: Generic response for unmatched or low-confidence queries

Error Handling

Error Types:

- **Tool Unavailable**: Selected tool is down or unreachable
- **LLM Failure**: LLM service unavailable or error
- **Configuration Error**: Invalid tools configuration
- **Timeout Error**: Tool response timeout
- **Validation Error**: Invalid request format
- **Low Confidence**: Query understanding below threshold
- **No Tool Match**: No applicable tool found for query

Fallback Strategy:

1. **Low Confidence/No Tool Match**: Return generic helpful message
2. **Tool Failure**: Retry with exponential backoff
3. **All Tools Down**: Return service unavailable message
4. Log detailed error information for all scenarios

Generic Response Messages:

Note: All pre-defined response messages, including guard rail messages, are placeholder text and will be updated once business stakeholders provide final approved verbiage.

```
{
  "no_tool_found": "I'm sorry, I couldn't find the right resource to help with your question. Please try rephrasing your query or contact our support team for assistance.",
  "low_confidence": "I'm not entirely sure I understand your question. Could you please provide more details or rephrase your request?",
  "service_unavailable": "I'm currently experiencing technical difficulties. Please try again in a few moments or contact support if the issue persists."
}
```

Logging

- Request/response logging with correlation IDs
- Tool selection accuracy metrics
- Performance metrics (latency, throughput)
- Error rate tracking per tool