Agentic Workflow Architecture

Overview

Hybrid workflow system combining:

Agent Orchestration & SLA Management

Agent Orchestration

- Agents are orchestrated per workflow using configurable strategies:
    - Execution modes: parallel, sequential, conditional, priority-based.
    - Red flag handling: early termination, escalation, or enhanced review.
    - Enrichment: agent outputs enrich entity metadata for downstream rules.
- Orchestration strategy is set per workflow and can be dynamically adjusted.

SLA Management

- SLA (Service Level Agreement) is tracked for each approval step and agent task.
- SLA breaches trigger escalation, notifications, or auto-approval/rejection.
- SLA configuration includes:
    - Per-agent and per-approver timeouts.
    - Escalation rules (who, when, how).
    - Fallback actions (auto-escalate, auto-reject, notify).
- SLA status is visible in workflow state and audit logs.

Example: Agent Orchestration & SLA in Workflow Template (YAML)

```yaml
workflow:
  version: 2.2
  agents:
    - agentId: "kyc-fraud-agent"
      type: "MCP"
      config:
        endpoint: "http://fraud-service:8090"
        model: "claude-3-7-sonnet"
        tools: ["analyze_fraud", "check_id"]
        timeout: 20000
      orchestration:
        mode: "ASYNC_RED_FLAG"
        priority: 1
        slaHours: 6
        escalation:
          after: "PT6H"
          escalateTo: "FRAUD_TEAM"
          notify: true
    - agentId: "document-ocr-agent"
```

```
      type: "CUSTOM"
      config:
        endpoint: "http://ocr-service:8096"
        tools: ["extract_text", "validate_format"]
        timeout: 10000
      orchestration:
        mode: "SEQUENTIAL"
        priority: 2
        slaHours: 2
        escalation:
          after: "PT2H"
          escalateTo: "OPS_TEAM"
          notify: true
  slaManagement:
    globalTimeout: "PT24H"
    breachAction: "ESCALATE"
    notifyOnBreach: true
```

Example: Agent Orchestration & SLA in Workflow Template (JSON)

```
{
  "templateId": "DOCUMENT_VERIFICATION_V2",
  "version": "2.2",
  "entityType": "DOCUMENT_VERIFICATION",
  "agentConfig": {
    "enableAgents": true,
    "agents": [
      {
        "agentId": "kyc-fraud-agent",
        "type": "MCP",
        "config": {
          "endpoint": "http://fraud-service:8090",
          "model": "claude-3-7-sonnet",
          "tools": ["analyze_fraud", "check_id"],
          "timeout": 20000
        },
        "orchestration": {
          "mode": "ASYNC_RED_FLAG",
          "priority": 1,
          "slaHours": 6,
          "escalation": {
            "after": "PT6H",
            "escalateTo": "FRAUD_TEAM",
            "notify": true
          }
        }
      },
      {
        "agentId": "document-ocr-agent",
        "type": "CUSTOM",
        "config": {
          "endpoint": "http://ocr-service:8096",
          "tools": ["extract_text", "validate_format"],
          "timeout": 10000
        },
        "orchestration": {
          "mode": "SEQUENTIAL",
          "priority": 2,
          "slaHours": 2,
          "escalation": {
            "after": "PT2H",
            "escalateTo": "OPS_TEAM",
            "notify": true
          }
        }
      }
    ],
    "slaManagement": {
      "globalTimeout": "PT24H",
      "breachAction": "ESCALATE",
      "notifyOnBreach": true
    }
  },
  "decisionTables": [ ... ]
}
```

Decision Flow Patterns

Pattern 1: Rule-Based DMN (Traditional)

```
Submit → Validate → Evaluate DMN Rules → Assign Approvers → Human
Approval
```

Pattern 2: Async Agent Red Flag Detection

```
Submit → Validate → Launch Agents (async) → Red Flag? → TERMINATE
                         ↓
                    No Red Flags → Continue to DMN → Human Approval
```

Pattern 3: Sync Agent + Rule Hybrid

```
Submit → Validate → Agent Analysis → Extract Insights → Evaluate DMN
with Agent Data → Human Approval
```

Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│    agentConfig: {                                               │
│      enableAgents: true,                                        │
│      agents: ["fraud-detection", "risk-assessment", "graph-rag"]│
│      mode: "ASYNC_RED_FLAG"  // or "SYNC_ENRICHMENT"            │
│    }                                                            │
│  }                                                              │
└─────────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────────┐
│              Agent Orchestration Layer                          │
│                                                                 │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐           │
│  │ MCP Agent    │  │ MCP Agent    │  │ GraphRAG     │           │
│  │ Fraud Detect │  │ AML Check    │  │ Knowledge    │           │
│  │              │  │              │  │ Retrieval    │           │
│  └──────────────┘  └──────────────┘  └──────────────┘           │
│         ↓                  ↓                  ↓                 │
│  ┌──────────────────────────────────────────────────┐           │
│  │           Agent Decision Aggregator              │           │
│  │  - Collect agent outputs                         │           │
│  │  - Check for red flags                           │           │
│  │  - Enrich entity metadata                        │           │
│  └──────────────────────────────────────────────────┘           │
└─────────────────────────────────────────────────────────────────┘
                              ↓
         ┌────────────────────┴────────────────────┐
         ↓                                         ↓
┌──────────────────┐                    ┌──────────────────────┐
│ RED FLAG FOUND   │                    │ NO RED FLAGS         │
│ - Terminate      │                    │ - Enrich metadata    │
│ - Auto-reject    │                    │ - Evaluate DMN       │
│ - Notify         │                    │ - Human approval     │
└──────────────────┘                    └──────────────────────┘
```

Agent Types

1. MCP Agent (Model Context Protocol)

Purpose: Real-time analysis using external AI services

Example: Fraud Detection Agent

```
{
  "agentId": "fraud-detection-mcp",
  "type": "MCP",
```

~~~
  "config": {
    "mcpServerUrl": "http://fraud-detection-service:8090",
    "timeout": 30000,
    "model": "claude-3-7-sonnet",
    "tools": [
      "analyze_transaction_patterns",
      "check_device_fingerprint",
      "verify_behavioral_biometrics"
    ]
  },
  "mode": "ASYNC_RED_FLAG",
  "redFlagConditions": {
    "fraudProbability": "> 0.7",
    "suspiciousPatterns": "count > 3",
    "deviceMismatch": "true"
  },
  "enrichmentOutputs": [
    "fraudScore",
    "riskFactors",
  ]
}

### 2. GraphRAG Agent

```json
  "agentId": "compliance-graph-rag",
  "type": "GRAPH_RAG",
  "config": {
    "embeddingModel": "text-embedding-3-large",
    "retrievalDepth": 3,
      "regulations",
      "high_risk_countries",
    ]
  "mode": "SYNC_ENRICHMENT",
    {
      "cypher": "MATCH (c:Customer)-[:RELATED_TO]->(s:SanctionedEntity)
WHERE c.id = $customerId RETURN s"
    },
    {
      "name": "get_jurisdiction_rules",
      "cypher": "MATCH (j:Jurisdiction {country: $country})-
[:HAS_REGULATION]->(r:Regulation) RETURN r"
    }
  ],
  "enrichmentOutputs": [
    "sanctionsMatch",
    "applicableRegulations",
    "complianceRequirements",
    "historicalRiskIndicators"
  ]
}
~~~

3. MCP Reasoning Agent

Purpose: Multi-step reasoning for complex decisions

Example: Risk Assessment Agent

```
{
  "agentId": "risk-assessment-mcp",
  "type": "MCP",
  "config": {
    "mcpServerUrl": "http://risk-service:8091",
    "model": "claude-3-7-sonnet",
    "tools": [
    ],
      {
        "step": "gather_financial_data",
        "tool": "get_financial_statements"
      },
      {
        "step": "analyze_ratios",
        "tool": "calculate_financial_ratios"
      },
      {
        "step": "assess_industry_risk",
        "tool": "get_industry_trends"
      },
      {
        "step": "compute_final_score",
        "tool": "aggregate_risk_score"
      }
    ]
  },
  "mode": "SYNC_ENRICHMENT",
  "enrichmentOutputs": [
    "creditRiskScore",
    "riskCategory",
    "recommendedLimits",
    "mitigationSuggestions"
  ]
}
```

Example Workflows

Workflow 1: Pure Rule-Based (Traditional DMN)

Use Case: Simple Product Configuration

No agents, just decision table evaluation.

```
{
  "templateId": "SIMPLE_PRODUCT_CONFIG",
  "entityType": "PRODUCT_CONFIGURATION",
  "agentConfig": {
    "enableAgents": false
  },
  "decisionTables": [
    {
      "name": "Product Approval Rules",
      "inputs": [
        {"name": "productType", "type": "string"},
        {"name": "priceVariance", "type": "number"}
      ],
      "outputs": [
        {"name": "approvalRequired", "type": "boolean"},
        {"name": "approverRoles", "type": "array"}
      ],
      "rules": [
        {
          "conditions": {"priceVariance": "<= 10"},
          "outputs": {"approvalRequired": false}
        },
        {
          "conditions": {"priceVariance": "> 10"},
          "outputs": {
            "approvalRequired": true,
            "approverRoles": ["PRODUCT_MANAGER"]
          }
        }
      ]
    }
  ]
}
```

Flow:

```
Submit → Validate → Evaluate DMN → Assign Approver (if needed) → Done
```

Workflow 2: Async Agent Red Flag Detection

Use Case: Customer Onboarding with Fraud Detection

Critical: If agents detect fraud/AML issues, immediately terminate workflow.

```
{
  "templateId": "CUSTOMER_ONBOARDING_WITH_FRAUD_DETECTION",
  "entityType": "CUSTOMER_ONBOARDING",
  "agentConfig": {
    "enableAgents": true,
    "mode": "ASYNC_RED_FLAG",
    "agents": [
      {
        "agentId": "fraud-detection-mcp",
        "type": "MCP",
        "config": {
          "mcpServerUrl": "http://fraud-detection:8090",
          "model": "claude-3-7-sonnet",
          "tools": ["analyze_fraud_patterns", "check_device_risk"]
        },
        "redFlagConditions": {
          "fraudProbability": "> 0.75",
          "deviceRiskScore": "> 80",
          "identityMismatch": "true"
        },
        "redFlagAction": {
          "action": "TERMINATE_WORKFLOW",
          "autoReject": true,
          "reason": "High fraud risk detected by AI agent",
          "notifyRoles": ["FRAUD_TEAM", "COMPLIANCE_OFFICER"]
        }
      },
      {
        "agentId": "aml-sanctions-mcp",
        "type": "MCP",
        "config": {
          "mcpServerUrl": "http://aml-service:8092",
          "tools": ["check_sanctions_lists", "screen_pep"]
        },
        "redFlagConditions": {
          "sanctionsMatch": "true",
          "pepMatch": "true",
          "highRiskJurisdiction": "true"
        },
        "redFlagAction": {
          "action": "TERMINATE_WORKFLOW",
          "autoReject": true,
          "reason": "AML sanctions or PEP match found",
          "escalateTo": "AML_OFFICER"
        }
      }
    ],
    "timeout": 60000,
    "continueOnTimeout": false
  },
  "decisionTables": [
    {
      "name": "Onboarding Approval Rules",
      "note": "Only evaluated if NO red flags from agents",
      "inputs": [
        {"name": "customerType", "type": "string"},
        {"name": "accountValue", "type": "number"},
        {"name": "agentFraudScore", "type": "number"},
        {"name": "agentAmlRisk", "type": "string"}
      ],
      "outputs": [
        {"name": "approverRoles", "type": "array"},
        {"name": "approvalCount", "type": "number"}
      ],
      "rules": [
        {
          "conditions": {
            "customerType": "INDIVIDUAL",
            "accountValue": "<= 100000",
            "agentFraudScore": "<= 30",
            "agentAmlRisk": "LOW"
          },
          "outputs": {
            "approverRoles": ["ONBOARDING_SPECIALIST"],
            "approvalCount": 1
          }
        },
        {
          "conditions": {
            "agentFraudScore": "> 30 && <= 60",
            "agentAmlRisk": "MEDIUM"
          },
          "outputs": {
            "approverRoles": ["ONBOARDING_SPECIALIST",
"COMPLIANCE_OFFICER"],
            "approvalCount": 2
          }
        }
      ]
    }
  ]
}
```

Flow:

```
Submit Customer Data
    ↓
Validate
    ↓
Launch Agents in Parallel (async)
    ↓
    ├─→ Fraud Detection Agent
    │   ├─→ RED FLAG? → TERMINATE → Auto-reject → Notify Fraud Team
    │   └─→ OK → Continue
    │
    └─→ AML Sanctions Agent
```

```
        ├─→ RED FLAG? → TERMINATE → Auto-reject → Escalate to AML
Officer
        └─→ OK → Continue
    ↓
Aggregate Agent Results
    ↓
Enrich entityMetadata with:
  - agentFraudScore: 25
  - agentAmlRisk: "LOW"
    ↓
Evaluate DMN with enriched data
    ↓
Result: Single approval needed (ONBOARDING_SPECIALIST)
    ↓
Human Approval
```

Workflow 3: Sync GraphRAG + MCP + Final DMN

Use Case: Complex Loan Application with Knowledge Retrieval

Pattern: Agents run synchronously to enrich data, then DMN evaluates with agent insights.

```
{
  "templateId": "LOAN_APPLICATION_INTELLIGENT",
  "entityType": "LOAN_APPLICATION",
  "agentConfig": {
    "enableAgents": true,
    "mode": "SYNC_ENRICHMENT",
    "agents": [
      {
        "agentId": "credit-risk-graphrag",
        "type": "GRAPH_RAG",
        "config": {
          "graphDbUrl": "neo4j://credit-graph:7687",
          "embeddingModel": "text-embedding-3-large",
          "queries": [
            {
              "name": "get_customer_history",
              "cypher": "MATCH (c:Customer {id: $customerId})-
[:HAS_LOAN]->(l:Loan) RETURN l.status, l.defaulted, l.amount"
            },
            {
              "name": "get_industry_risk",
              "cypher": "MATCH (i:Industry {name: $industry})-
[:HAS_DEFAULT_RATE]->(r:RiskMetric) RETURN r.rate, r.trend"
            },
            {
              "name": "find_similar_customers",
              "cypher": "MATCH (c1:Customer {id: $customerId})-
[:SIMILAR_TO]->(c2:Customer)-[:HAS_LOAN]->(l:Loan) WHERE l.defaulted =
true RETURN count(l) as similarDefaults"
            }
          ]
        },
        "enrichmentOutputs": [
          "historicalDefaultRate",
          "industryRiskTrend",
          "similarCustomerDefaults",
          "graphRiskScore"
        ]
      },
      {
        "agentId": "financial-analysis-mcp",
        "type": "MCP",
        "config": {
          "mcpServerUrl": "http://financial-analysis:8093",
          "model": "claude-3-7-sonnet",
          "tools": [
            "analyze_financial_statements",
            "calculate_dscr",
            "assess_collateral_value",
            "predict_cash_flow"
          ],
          "reasoningSteps": [
            {
              "step": "parse_financials",
              "tool": "analyze_financial_statements",
              "input": "$.entityData.financialStatements"
            },
            {
              "step": "compute_ratios",
              "tool": "calculate_dscr"
            },
            {
              "step": "evaluate_collateral",
              "tool": "assess_collateral_value",
              "input": "$.entityData.collateral"
            },
            {
              "step": "forecast_repayment",
              "tool": "predict_cash_flow"
            }
          ]
        },
        "enrichmentOutputs": [
          "debtServiceCoverageRatio",
          "collateralCoverageRatio",
          "cashFlowStability",
          "financialHealthScore"
        ]
      },
      {
        "agentId": "regulatory-compliance-graphrag",
        "type": "GRAPH_RAG",
        "config": {
          "graphDbUrl": "neo4j://compliance-graph:7687",
          "queries": [
            {
              "name": "check_lending_limits",
              "cypher": "MATCH (r:Regulation {jurisdiction:
$jurisdiction})-[:SETS_LIMIT]->(l:LendingLimit) WHERE l.loanType =
$loanType RETURN l.maxAmount, l.maxLTV"
            },
            {
              "name": "verify_documentation",
              "cypher": "MATCH (r:Regulation)-[:REQUIRES_DOCUMENT]->
(d:DocumentRequirement) WHERE r.jurisdiction = $jurisdiction RETURN
d.documentType, d.mandatory"
            }
          ]
        },
        "enrichmentOutputs": [
          "regulatoryMaxLoanAmount",
          "maxLoanToValue",
          "requiredDocuments",
          "complianceStatus"
        ]
      }
    ],
    "executionMode": "SEQUENTIAL",
    "failOnAgentError": false,
    "timeout": 120000
  },
  "decisionTables": [
    {
      "name": "Loan Approval Decision",
      "note": "Evaluates using both original data AND agent-enriched
insights",
      "inputs": [
        {"name": "loanAmount", "type": "number"},
        {"name": "creditScore", "type": "number"},
        {"name": "loanType", "type": "string"},
        {"name": "graphRiskScore", "type": "number", "source": "agent"},
        {"name": "debtServiceCoverageRatio", "type": "number", "source":
"agent"},
        {"name": "collateralCoverageRatio", "type": "number", "source":
"agent"},
        {"name": "historicalDefaultRate", "type": "number", "source":
"agent"},
        {"name": "complianceStatus", "type": "string", "source":
"agent"}
      ],
      "outputs": [
        {"name": "decision", "type": "string"},
        {"name": "approverRoles", "type": "array"},
        {"name": "approvalCount", "type": "number"},
        {"name": "conditions", "type": "array"}
      ],
      "rules": [
        {
          "ruleId": "AUTO_REJECT_COMPLIANCE_FAIL",
          "priority": 100,
          "conditions": {
            "complianceStatus": "FAILED"
          },
          "outputs": {
            "decision": "AUTO_REJECT",
            "approverRoles": [],
            "conditions": ["Regulatory compliance requirements not met"]
          }
        },
        {
          "ruleId": "AUTO_REJECT_HIGH_RISK",
          "priority": 99,
          "conditions": {
            "graphRiskScore": "> 80",
            "historicalDefaultRate": "> 15"
          },
          "outputs": {
            "decision": "AUTO_REJECT",
            "approverRoles": [],
            "conditions": ["High risk based on historical patterns and graph analysis"]
          }
        },
        {
          "ruleId": "AUTO_APPROVE_EXCELLENT",
          "priority": 95,
          "conditions": {
            "creditScore": ">= 750",
            "debtServiceCoverageRatio": ">= 1.5",
            "collateralCoverageRatio": ">= 1.3",
            "graphRiskScore": "<= 30",
            "loanAmount": "<= 500000"
          },
          "outputs": {
            "decision": "AUTO_APPROVE",
            "approverRoles": [],
            "approvalCount": 0,
            "conditions": ["Excellent credit metrics and low AI-assessed risk"]
          }
        },
        {
          "ruleId": "SINGLE_APPROVAL_GOOD",
          "priority": 80,
          "conditions": {
            "creditScore": ">= 680",
            "debtServiceCoverageRatio": ">= 1.25",
            "graphRiskScore": "<= 50"
          },
          "outputs": {
            "decision": "APPROVE_WITH_CONDITIONS",
            "approverRoles": ["CREDIT_OFFICER"],
            "approvalCount": 1,
            "conditions": ["Standard approval required"]
          }
        },
        {
          "ruleId": "DUAL_APPROVAL_MODERATE_RISK",
          "priority": 75,
          "conditions": {
            "creditScore": ">= 620",
            "graphRiskScore": "> 50 && <= 70",
            "debtServiceCoverageRatio": ">= 1.15"
          },
          "outputs": {
            "decision": "APPROVE_WITH_CONDITIONS",
            "approverRoles": ["CREDIT_OFFICER",
"SENIOR_CREDIT_OFFICER"],
            "approvalCount": 2,
            "conditions": [
              "Enhanced monitoring required",
              "Consider additional collateral"
            ]
          }
        },
        {
          "ruleId": "COMMITTEE_APPROVAL_HIGH_VALUE",
          "priority": 90,
          "conditions": {
            "loanAmount": "> 5000000"
          },
          "outputs": {
            "decision": "COMMITTEE_APPROVAL",
            "approverRoles": ["CREDIT_COMMITTEE"],
            "approvalCount": 1,
            "conditions": [
              "Credit committee review required for high-value loans"
            ]
          }
        }
      ],
      "hitPolicy": "PRIORITY"
    }
  ]
}
```

Flow:

```
Submit Loan Application
    ↓
Validate
    ↓
Execute Agents Sequentially:
    ↓
1. Credit Risk GraphRAG Agent
   - Query graph for customer history
   - Query graph for industry risk
   - Find similar customer patterns
   → Output: graphRiskScore: 45, historicalDefaultRate: 8%,
similarCustomerDefaults: 3
    ↓
2. Financial Analysis MCP Agent
   Step 1: Parse financial statements
   Step 2: Calculate DSCR = 1.4
   Step 3: Assess collateral coverage = 1.5
   Step 4: Predict cash flow stability = "HIGH"
   → Output: debtServiceCoverageRatio: 1.4, collateralCoverageRatio: 1.5
    ↓
3. Regulatory Compliance GraphRAG Agent
   - Check lending limits for jurisdiction
   - Verify required documents
   → Output: complianceStatus: "PASSED", maxLoanToValue: 80%
    ↓
Aggregate Agent Results & Enrich Metadata:
  {
    loanAmount: 750000,
    creditScore: 720,
    loanType: "COMMERCIAL",
    graphRiskScore: 45,                  ← from GraphRAG
    debtServiceCoverageRatio: 1.4,       ← from MCP
    collateralCoverageRatio: 1.5,        ← from MCP
    historicalDefaultRate: 8,            ← from GraphRAG
    complianceStatus: "PASSED"           ← from GraphRAG
  }
    ↓
Evaluate DMN with Enriched Data:
  - Rule AUTO_REJECT_COMPLIANCE_FAIL: NO
  - Rule AUTO_REJECT_HIGH_RISK: NO (graphRiskScore=45 < 80)
  - Rule AUTO_APPROVE_EXCELLENT: NO (loanAmount=750k > 500k)
  - Rule SINGLE_APPROVAL_GOOD: YES ✓
    ↓
Result:
  {
    decision: "APPROVE_WITH_CONDITIONS",
    approverRoles: ["CREDIT_OFFICER"],
    approvalCount: 1,
    conditions: ["Standard approval required"]
  }
    ↓
Assign CREDIT_OFFICER
```

```
    ↓
Human Approval with AI Insights Displayed
```

Agent Decision Models

AgentDecision

```java
@Data
@Builder
public class AgentDecision {
    private String agentId;
    private String agentType;  // "MCP", "GRAPH_RAG"
    private LocalDateTime executedAt;
    private Duration executionTime;

    // Red flag detection
    private boolean redFlagDetected;
    private String redFlagReason;
    private RedFlagSeverity severity;
    private RedFlagAction recommendedAction;

    // Enrichment outputs
    private Map<String, Object> enrichmentData;

    // Agent reasoning trace
    private List<AgentReasoningStep> reasoningSteps;

    // Confidence and metadata
    private double confidenceScore;
    private String model;
    private Map<String, Object> agentMetadata;
}

public enum RedFlagSeverity {
    LOW,
    MEDIUM,
    HIGH,
    CRITICAL
}

public enum RedFlagAction {
    CONTINUE,           // Log but continue
    ENHANCE_REVIEW,     // Add additional approver
    TERMINATE_REJECT,   // Auto-reject
    ESCALATE           // Escalate to senior
}
```

AgentReasoningStep

```java
@Data
@Builder
public class AgentReasoningStep {
    private int stepNumber;
    private String stepName;
    private String tool;
    private Map<String, Object> input;
    private Map<String, Object> output;
    private String reasoning;
    private LocalDateTime timestamp;
    private Duration duration;
}
```

MCPAgentConfig

```java
@Data
@Builder
public class MCPAgentConfig {
    private String mcpServerUrl;
    private String model;
    private List<String> tools;
    private List<ReasoningStep> reasoningSteps;
    private int timeout;
    private Map<String, String> headers;
    private RetryPolicy retryPolicy;
}

@Data
public class ReasoningStep {
    private String step;
    private String tool;
    private String input;  // JSONPath expression
    private Map<String, Object> parameters;
}
```

GraphRAGAgentConfig

```java
@Data
@Builder
public class GraphRAGAgentConfig {
    private String graphDbUrl;
    private String embeddingModel;
    private int retrievalDepth;
    private List<String> entities;
    private List<CypherQuery> queries;
    private SemanticSearchConfig semanticSearch;
}
```

```
@Data
public class CypherQuery {
    private String name;
    private String cypher;
    private Map<String, String> parameters;
    private String resultMapping;
}

@Data
public class SemanticSearchConfig {
    private boolean enabled;
    private String questionField;
    private int topK;
    private double similarityThreshold;
}
```

Agent Execution Models

AgentExecutionPlan

```
@Data
@Builder
public class AgentExecutionPlan {
    private ExecutionMode mode;
    private List<AgentTask> tasks;
    private int timeout;
    private boolean failOnError;
    private AgentOrchestrationStrategy strategy;
}

public enum ExecutionMode {
    ASYNC_RED_FLAG,      // Parallel, terminate on red flag
    SYNC_ENRICHMENT,     // Sequential, enrich metadata
    HYBRID               // Mix of both
}

public enum AgentOrchestrationStrategy {
    PARALLEL,            // All agents execute simultaneously
    SEQUENTIAL,          // One after another
    CONDITIONAL,         // Based on previous results
    PRIORITY_BASED       // High priority first
}
```

AgentTask

```
@Data
@Builder
```

```
public class AgentTask {
    private String agentId;
    private AgentType type;
    private Object config;  // MCPAgentConfig or GraphRAGAgentConfig
    private ExecutionMode mode;

    // Red flag configuration
    private Map<String, String> redFlagConditions;
    private RedFlagAction redFlagAction;

    // Enrichment configuration
    private List<String> enrichmentOutputs;
    private String outputMapping;

    // Execution settings
    private int priority;
    private int timeout;
    private RetryPolicy retryPolicy;
}

public enum AgentType {
    MCP,
    GRAPH_RAG,
    CUSTOM
}
```

Async Red Flag Pattern

Workflow Behavior

```
// In Temporal workflow
@WorkflowMethod
public WorkflowResult execute(WorkflowSubject subject) {

    // Step 1: Launch all agents
    List<Promise<AgentDecision>> agentPromises = new ArrayList<>();

    for (AgentTask task : subject.getAgentConfig().getTasks()) {
        Promise<AgentDecision> promise = Async.function(() ->
            executeAgent(task, subject)
        );
        agentPromises.add(promise);
    }

    // Step 2: Wait for first red flag OR all complete
    Selector selector = new Selector();

    for (Promise<AgentDecision> promise : agentPromises) {
        selector.addCondition(() -> promise.isCompleted() &&
                                   promise.get().isRedFlagDetected(),
            () -> {
                AgentDecision decision = promise.get();
                // RED FLAG DETECTED - TERMINATE
                return WorkflowResult.builder()
                    .success(false)
                    .resultCode("RED_FLAG_DETECTED")
                    .message(decision.getRedFlagReason())
                    .agentDecision(decision)
                    .build();
            });
    }

    // All agents complete without red flags
    selector.addCondition(() ->
Promise.allOf(agentPromises).isCompleted(),
        () -> {
            // Aggregate agent results
            List<AgentDecision> decisions = agentPromises.stream()
                .map(Promise::get)
                .collect(Collectors.toList());

            // Enrich metadata
            Map<String, Object> enrichedMetadata = enrichMetadata(
                subject.getEntityMetadata(),
                decisions
            );

            // Evaluate DMN with enriched data
            ComputedApprovalPlan plan = evaluateRules(
                template,
                enrichedMetadata
            );

            return continueWorkflow(plan);
        });

    selector.select();
}
```

Integration with MCP Servers

MCP Tool Invocation

```
public class MCPAgentExecutor {

    public AgentDecision execute(MCPAgentConfig config, WorkflowSubject
subject) {

        List<AgentReasoningStep> steps = new ArrayList<>();
        Map<String, Object> context = new HashMap<>
(subject.getEntityData());

        // Execute reasoning steps sequentially
        for (ReasoningStep step : config.getReasoningSteps()) {

            // Invoke MCP tool
            MCPToolRequest request = MCPToolRequest.builder()
                .tool(step.getTool())
                .input(extractInput(step.getInput(), context))
                .parameters(step.getParameters())
                .build();

            MCPToolResponse response = mcpClient.invokeTool(
                config.getMcpServerUrl(),
                request
            );

            // Record reasoning step
            steps.add(AgentReasoningStep.builder()
                .stepNumber(steps.size() + 1)
                .stepName(step.getStep())
                .tool(step.getTool())
                .input(request.getInput())
                .output(response.getResult())
                .reasoning(response.getReasoning())
                .build());

            // Add to context for next step
            context.putAll(response.getResult());
        }

        // Extract enrichment data
        Map<String, Object> enrichment = extractEnrichment(
            context,
            config.getEnrichmentOutputs()
        );

        // Check for red flags
        boolean redFlag = checkRedFlags(enrichment,
config.getRedFlagConditions());

        return AgentDecision.builder()
            .agentId(config.getAgentId())
            .agentType("MCP")
            .redFlagDetected(redFlag)
            .enrichmentData(enrichment)
            .reasoningSteps(steps)
            .model(config.getModel())
            .build();
    }
}
```

Benefits

1. **Intelligent Automation**

- AI agents pre-screen applications
- Auto-reject obvious fraud/risk cases
- Auto-approve low-risk cases

2. **Context-Aware Decisions**

- GraphRAG retrieves relevant historical patterns
- Knowledge graph provides regulatory context
- MCP agents apply complex reasoning

3. **Flexible Execution**

- Async red flag detection for early termination
- Sync enrichment for rule enhancement
- Hybrid approaches for complex scenarios

4. **Explainable AI**

- Full reasoning trace from agents
- DMN rules remain interpretable
- Combined AI + human judgment

5. **Extensible Agent Framework**

- Add new MCP tools without code changes
- Configure GraphRAG queries via templates
- Plugin architecture for custom agents

Configuration Example

```yaml
# application.yml
workflow:
  agents:
    mcp:
      default-timeout: 30000
      retry-attempts: 2
      servers:
        fraud-detection: http://fraud-service:8090
        financial-analysis: http://financial-service:8093
        aml-screening: http://aml-service:8092

    graph-rag:
      neo4j-url: neo4j://graph-db:7687
      embedding-service: http://embedding-service:8094
      default-retrieval-depth: 3
      cache-ttl: 3600

    orchestration:
      max-parallel-agents: 5
      default-timeout: 120000
      fail-fast: true
```

This hybrid approach combines the best of rule-based and AI-driven decision-making!