A Multi-Agent Workflow Orchestration System for Complex Long-Cycle Tasks

Yitong Qin[1], Zhichen Lu[1], Lihong Wu[1], Juncheng Zhao[1], Tingyu Lyu[1]
[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, PR China.
Corresponding author: Yitong Qin (e-mail: 19982069939@163.com)

ABSTRACT A dynamic workflow orchestration system with the control plane separated from the execution plane is proposed to address the problem that traditional hard-coded service orchestration cannot handle business process changes and service failures in long-cycle tasks. The system uses a Multi-Agent architecture, through the division of roles such as the orchestration engine Agent, task execution Agent, and user Agent, combined with directory service (DF) registration and QOS-driven Contract Network Protocol (CNP), to achieve dynamic service selection and delay binding for flexible activities. An access control model based on dynamic binding of roles and tasks (R-TBAC) is introduced to enable dynamic granting and revocation of permissions driven by task instances.

INDEX TERMS Access control, Dynamic workflow, Long-cycle tasks, Multi-Agent system

I. INTRODUCTION

IN enterprise application integration, long-cycle tasks often span multiple heterogeneous systems and several execution cycles. Traditional hard-coded service orchestration lacks flexibility in the face of runtime service address changes, temporary unavailability, or business process adjustments, resulting in frequent failures of process instances and providing a new technical path for the automation of long-cycle enterprise applications.

II. DESIGN OF MULTI-AGENT WORKFLOW MODEL FOR LONG-CYCLE TASKS

A. SYSTEM ARCHITECTURE

1) CONTROL PLANE AND EXECUTION PLANE SEPARATED ARCHITECTURE

Enterprise application integration and automation processing systems for complex long-cycle tasks need to overcome the problem that traditional hard-coded service orchestration cannot handle runtime business process changes and service failures[1]. The control plane issues service selection constraints and QoS parameters to the execution plane through the dynamic service layer, and the multi-agent system in the execution plane is based on the runtime context from the general service layer (legacy systems encapsulated by adapters CRM/ERP/PDM and external WebServices), Registered in private or public UDDI, specific service instances are matched in real time to decouple services from process logic at runtime and shift to a dynamic binding mode. The layering and separation mechanism ensures that long-cycle tasks do not fail due to changes in the underlying service address or temporary unavailability when executed across days and weeks, and supports the "plug and unplug" of services, providing strong support for long-cycle tasks[2].

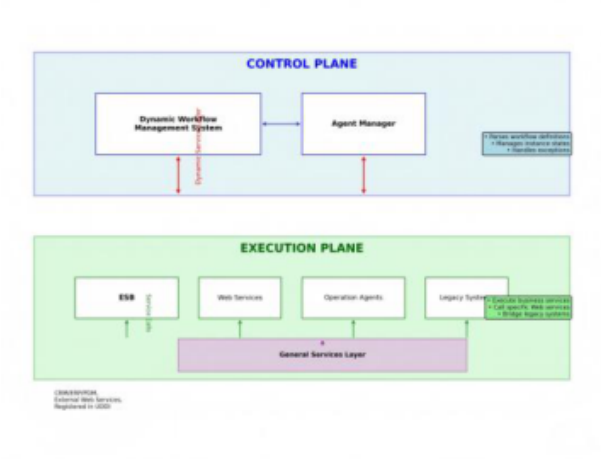

FIGURE 1. Overall system architecture diagram

2) AGENT CONTAINER AND DIRECTORY FACILITATOR (DF) SERVICE REGISTRATION MECHANISM

In a multi-agent system (MAS), each Agent resides in an Agent container that complies with the FIPA specification[3]. When the container starts, it registers itself with the AMS (Agent Management Service) and acquires the AID (Agent Identifier). The integrated Agent and Agent-Manager register the service description they provide to the DF (DirectoryFacilitator) during the initialization phase. The service description extends to include QoS attributes (response time, availability, execution cost, reputation, etc.) in addition to the basic input/output and semantic infor-

mation. When the dynamic workflow engine needs to bind specific Web services while performing flexible activities[4], the Agent Manager initiates a service query request to the DF, which returns a reference to the integrated Agent corresponding to the candidate service based on the service type and QoS constraints, and the Agent perceives the newly launched or exited service Agent by subscribing to DF change notifications Maintain the dynamic update of the service view.

3) STATE PERSISTENCE AND BREAKPOINT RESUMPTION DESIGN FOR LONG-CYCLE TASKS

Long-cycle tasks often span multiple system starts and stops or network disruptions. Workflow management systems have built-in process definition modules, workflow management modules, monitoring modules, and a core workflow engine. The engine maintains workflow control data (token location, active instance status, variable context, suspension reasons, etc.). When a workflow instance enters a node waiting for an external asynchronous response, manual approval, or long-duration computation, the engine serializes a complete snapshot of the current instance (including the execution status of each active node, scope variables, nested relationships between parent processes and child processes, and bound service information) and writes it to persistent storage if the system restarts or the Agent fails abnormally to trigger the recovery process, AgentManager instructs the workflow engine to load the most recent consistent checkpoint from the persistent store, restore to the suspended state and reactivate, and invoke the corresponding solution in the Agent repository to activate the appropriate integrated Agent to continue the subsequent activity execution[5]. This breakpoint resumption mechanism, in combination with structural changes such as skip, insert, and parallel serial switching supported by the dynamic workflow model, allows for limited adjustments to the original process model at the time of recovery (such as reselecting alternative services or skipping invalid optional branches), regenerating new instances to continue execution in the original state, meeting the dual requirements of fault tolerance and flexibility for long-cycle tasks Experimental results show that the architecture can reduce the request queuing rate to 0% by laterally scaling the Agent instance under a concurrent pressure of 80TPS; In a single point of failure scenario, the system recovery time averaged 4.2 seconds and the process completion rate reached 98%, verifying its reliability and scalability in high concurrency and failure scenarios[6].

B. AGENT ROLE SYSTEM AND FUNCTION DIVISION

1) MAPPING OF THE ORCHESTRATION ENGINE AGENT (WORKFLOW-ENGINE AGENT) TO PROCESS INSTANCES

The orchestration engine Agent encapsulates the core scheduling logic of the dynamic workflow management system, with each main process instance of a long-cycle business process corresponding to a logical orchestration engine Agent session (or centrally managed by the engine but triggered by Agent message-driven activities). The Agent holds the process definitions (including normal activity and flexible activity tags), creates process instances at the start of the process and initializes global variables; When the flow reaches a regular activity node, the pre-associated service is directly invoked and executed via ESB; When encountering flexible active nodes[7], send a service selection request to the Agent Manager with context parameters and QoS constraints. After receiving the specific service endpoints and invocation contracts returned by AgentManager, the orchestration engine Agent dynamically binds the flexible activity to the actual Web service, passes the driver token downstream, and at the end of the process instance, the orchestration engine Agent summarizes the execution results of each activity and writes them back to the business database and notifies the relevant StaffAgent. This design enables the workflow engine to have autonomous interaction capabilities, fitting the Agent's "reactivity" and "proactiveness" characteristics, and can proactively advance or suspend instances based on the returned results during runtime[8].

2) TASK AGENT AND RESOURCE ADAPTATION AGENT

The integrated Agent assumes the functions of the task execution Agent, and each integrated Agent is responsible for the invocation encapsulation of one or several related business services. The integrated Agent connects heterogeneous background systems through adapters - adapters consist of three parts: connection adapter, Web service encapsulator, and message communication, converting heterogeneous apis such as ERP inventory query, CRM customer credit verification, and PDM drawing version acquisition into standard Web service interfaces for ESB routing[9]. TaskAgent, upon receiving a standardized service request sent by the orchestration engine Agent (forwarded via AgentManager), queries the local cache or refreshes the service WSDL from UDDI, performs protocol conversion and message format conversion, invokes the target service and deserializes the response data as agreed in the context, The Resource adaptation Agent, as a special variant of the Task Agent, is specifically responsible for database connection pool management, file server interaction, and bridging legacy system-specific protocols such as the early Socket or CORBA interface, masking the underlying differences and leaving the upper-level Agent only facing the unified service abstraction[10].

3) THE STAFFAGENT AND THE MONITORING MANAGEMENT AGENT

StaffAgent resides in the client environment on behalf of terminal business personnel or administrators, has human-computer interaction interfaces, can initiate new process instances, fill out manual task forms, and confirm anomaly recovery strategies[11]. When the workflow engine encounters flexible activities that require human decision-making (such

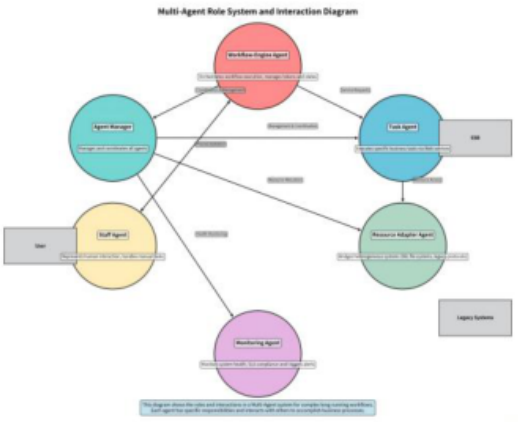

FIGURE 2. Multi-Agent Role System and Interaction diagram

as special discount approval, non-standard contract signing), push pending task messages to the designated StaffAgent, the StaffAgent presents the interface for users to operate, and returns the results to trigger the process to continue. The monitoring and management Agent is embedded in the Agent Manager and periodically collects the heartbeat of each integrated Agent[12], current load, recent failure record, and key metrics of the workflow instance (average dwell time, backlog queue depth) and outputs them in the form of a dashboard; When an SLA deviation is detected (such as a flexible activity exceeding the threshold without binding to services), an automatic alert is triggered and predefined compensation actions (such as expanding the service search range, downgrading to suboptimal QoS services) are triggered[13]. Together with the StaffAgent, the monitoring management Agent completes the interaction closed loop of the multi-agent system in the "social" dimension. Provide strong support for the overall architecture[14].

TABLE I. List of Core Agent Roles and Functions of the System

Agent Name	Core Functions	Key Interaction Objects	Start/Create timing
Orchestration Engine Agent	Parse process definitions, drive process instance execution, and manage tokens and states	Agent Manager, StaffAgent	When the process instance is created
Task Execution Agent	Encapsulate and invoke specific Web services to perform atomic business operations	ESB, adapter, resource adaptation Agent	Batch startup during system initialization
Resource adaptation Agent	Bridging heterogeneous systems (databases, file systems, legacy protocols)	Task Agent, legacy system	Start when the system is initialized
StaffAgent	Representing human-computer interaction, initiating processes, handling human tasks, and identifying exceptions	Orchestration engine Agent, user	Created when the user logs in
Monitor and Manage Agent	Collect heartbeats, loads, failure records, monitor SLAs, trigger alerts and compensation	Agent Manager, all Agents	Start when the system initializes

C. ACCESS CONTROL MODEL BASED ON DYNAMIC BINDING OF ROLES AND TASKS (R-TBAC)

1) TASK-INSTANCE-DRIVEN DYNAMIC GRANTING AND RECLAIMING OF PERMISSIONS

The traditional RBAC model has relatively static permissions and is not suitable for scenarios where personnel temporarily intervene and roles switch with task stages in long-cycle tasks. The system extends to a dynamic authorization mechanism triggered by task instances: When the workflow engine instantiates a human activity node, based on the role requirements configured for the node (such as "Branch financial reviewer"), it queries the IAM subsystem with the current user identity that has that role and is on duty, and dynamically establishes < user, role, Task instance, validity period > quadruple session permissions[15]; When a user logs in through StaffAgent and acquires this temporary permission to operate the corresponding form, the system automatically recovers this quadruple permission after the task instance is completed or timed out, without affecting the user's existing resident role[16]. This approach avoids the internal control risks caused by long-term permission residence and precisely corresponds to the brief manual links in the long cycle process, providing strong support for permission management[17].

2) FORMAL DEFINITION OF THE R-TBAC MODEL

To describe the access control model proposed in this paper more rigorously, its formal definition is presented. The model draws on the research of Du Yujiao et al. And expands on it.

Definition 1 (Basic entity set):

U: User set. R: Role set. P: Permission set, denoted as $P = OP \times OBJ$, where OP is the set of operations and OBJ is the set of objects (such as services, data). T: Task set. TI: Task instance set. AG: Agent set.

Definition 2 (Assignment relationship):

$UA \subseteq U \times R$: User-role assignment relationship. $PA \subseteq P \times R$: Permission-role assignment relationship. $TA \subseteq T \times AG$: Task-agent assignment relation indicating which Agent can perform which task.

Definition 3 (Role Hierarchy):

$RH \subseteq R \times R$: Partial order relations on roles. $r_1 \geq r_2$ indicates that r_1 inherits all the permissions of r_2.

Definition 4 (Dynamic Authorization):

Let $\mathrm{auth}(u, ag, ti)$ represent the set of permissions that user u acquires through Agent ag in the context of the task instance ti. When the workflow engine instantiates a human task node that requests role r, the system dynamically creates a session s and establishes a quadruple authorization:

$$\begin{aligned}\mathrm{auth}(u, ag, ti) = \{p \in P \mid (\exists r' \in R)[(u, r') \in UA \\ \wedge (r' \geq r) \wedge (p, r') \in PA \\ \wedge (ti, ag) \in TA]\}.\end{aligned}$$

This authorization is valid only during the lifetime(ti) of the task instance ti and is automatically reclaimed after the task ends[18].

Definition 5 (Separation of Duties Constraints):

Static duty separation (SSD):

$$\forall u \in U,\ \forall r_1, r_2 \in R,$$
$$\text{if } (r_1, r_2) \in SSD \text{ then } (u, r_1) \notin UA \vee (u, r_2) \notin UA.$$

Dynamic Separation of Duties (DSD):

$$\forall u \in U,\ \forall s \in Session,$$
$$\text{if } (r_1, r_2) \in DSD \text{ then } r_1 \notin role(s) \vee r_2 \notin role(s).$$

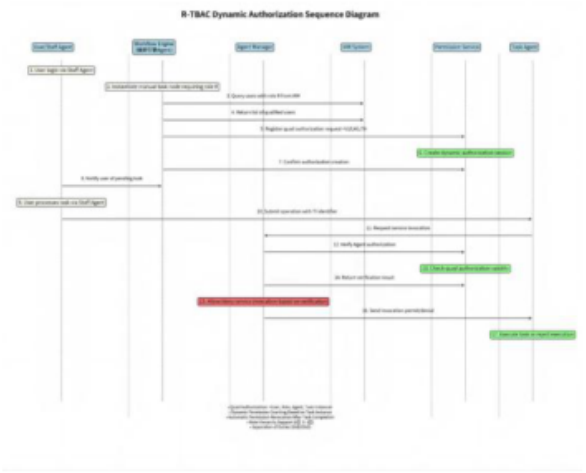

FIGURE 3. R-TBAC Dynamic Authorization Lane Sequence diagram

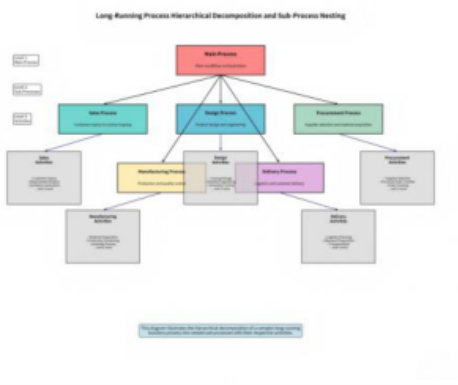

FIGURE 4. Long-cycle process hierarchy and subprocess nesting diagram

3) THE QUADRUPLE AUTHORIZATION RELATIONSHIP OF USER-AGENT-ROLE-TASK

The formal definition is Authorization::= (U, R, A_gent, TI), where U is the set of users, R is the set of roles, and A_gent is the set of integrated agents with the corresponding service invocation rights (based on which Agent Manager limits which agents can perform the services behind the role)[19]. TI is the task instance identifier and the time window. The workflow engine registers a quadruple with the permission service when creating instances of human activities; When the Agent Manager receives a service invocation request from the Integrated Agent, it checks whether the Agent is authorized under the corresponding role of the service category and whether it is within the associated TI validity period. The operations submitted by the Staff Agent must carry the TI identity for backend verification. The model incorporates the Agent as an automated Agent extension of the user into the same access control boundary to prevent unauthorized agents from invoking sensitive services (such as directly modifying ERP accounts payable).

4) IMPLEMENTATION OF THE PRINCIPLE OF LEAST PRIVILEGE AND SEPARATION OF DUTIES IN LONG-CYCLE TASKS

Each integrated Agent in the system registers only the minimum set of permissions required to complete the service invocation in its own business domain (for example, the purchasing audit Agent has no right to invoke the financial Posting service), and the AgentManager rejects out-of-scope service discovery requests[20]. Separation of duties (SoD) is achieved through mutual exclusion role detection within instances of the same process: If node N1 in the process definition requires role R1 (purchase request), node N2 requires role R2 (purchase approval), and R1 ∩R2 =∅cannot be activated simultaneously on the same user, the authority service verifies that the user's activated role set does not contain mutually exclusive roles when allocating StaffAgent task instances[21]. For long-cycle tasks that are executed across days, the system re-assesses the user's job status and role assignment in the early hours of the morning. If the user leaves or the role is revoked, the pending instance is automatically upgraded to the superior standby role user or transferred to the manual pool for reallocation, taking into account both security compliance and process continuity[22].

III. COMPLEX TASK DECOMPOSITION AND AGENT COLLABORATIVE EXECUTION MECHANISM

A. HIERARCHICAL DECOMPOSITION OF LONG CYCLE PROCESSES AND NESTING OF SUBFLOWS

1) INSTANTIATION OF THE MAIN PROCESS AND THE SUB-WORKFLOW ENGINE AGENT

Complex long-cycle tasks (such as the entire chain of customized sales, design, procurement, production and delivery of large equipment) are vertically decomposed by business semantics into several stage subprocesses[23], each of which corresponds to an independent workflow model and is labeled as a general activity or a composite activity with flexible activities, and the main process model is defined in the dynamic workflow management system. Subprocess nodes are specified as nested Sub-Workflow references; When the orchestration Engine Agent flows to the subprocess entry and sends a request to the workflow engine to create a subprocess instance, the engine derives the Sub-Workflow Engine Agent(or reuses the engine subprocess execution thread but encapsulates lifecycle events with Agent messages), Pass in the parent instance ID, inherited context variables, and the subprocess definition identifier, and when the subprocess is completed, return the output parameters to the parent instance and trigger the parent process to continue[24].

2) DYNAMIC SERVICE SELECTION AND DELAYED BINDING OF FLEXIBLE ACTIVE NODES

In dynamic workflow models, activities are divided into regular activities (where service endpoints are bound and determined at the time of modeling) and flexible activities (where they are determined at runtime); For services that need to invoke external credit verification, logistics quotation, tax calculation, etc., which are variable or have multiple suppliers[25], declare them as flexible activities and attach candidate service types and QoS constraint templates when modeling, and do not invoke them immediately when the workflow instance runs to this node Orchestration engine Agent encapsulates the context (input parameters, expected response time cap, cost cap, data sovereignty requirements, etc.) and sends it to Agent Manager; The Agent Manager directed the integrated Agent to query DF and UDDI to obtain a list of similar services, sort them based on QoS multi-attribute decisions (weighted response time countdown, availability, reputation, excluding over-cost or low availability), and inject specific WSDL and endpoints into the flexible activity binding table; If the first candidate service call fails and the number of retries is not exhausted, transparently switch to the next option, and if both fail, return to the engine to trigger the exception handler[26].

3) PASSING AND RECOVERY STRATEGIES FOR TASK CONTEXTS ACROSS TIME PERIODS

Subprocesses in long-cycle tasks may be executed several days apart (such as waiting for the client to return the stamped document after the contract approval is completed), and the context needs to survive across multiple Agent sessions; The system defines the Process Context as a collection of key-value pairs containing business entity ids, preemptive activity outputs, decision flags, deadlines, etc[27]. It is divided into global context and local subprocess context by scope, and parent →subcontext copies by value and allows subprocesses to write back incremental results; The child process ends and merges back to the parent; When persisting, the context is snapshots along with the workflow control data. In the recovery scenario, the engine loads the context to rebuild the InputBinding of each Activity. If the original binding service of the flexible activity is offline, the selection logic can be re-executed based on the latest UDDI instead of forcing the continuation of the invalid binding. For context fields that require manual completion (such as the URL of the scanned copy returned by the client), the Staff Agent writes the corresponding key of the persistent context directly when submitting, and the engine wakes up and reads the latest value to continue the downstream determination gateway flow.

B. AGENT NEGOTIATION AND TASK ASSIGNMENT BASED ON THE CONTRACT NET PROTOCOL

1) TASK BROADCASTING, BIDDING, AND WINNING DECISION ALGORITHMS

For tasks with multiple potential executors (such as cross-regional warehouse allocation with multiple WMS systems available for response), the FIPA Contract Net Protocol (CNP) is used to organize negotiations among agents; Orchestration engine Agent or Agent Manager acts as the Initiator, Encapsulate the task description (task ID, input summary, Deadline, QoS expectation) as a CFP(Call For Proposal) message and broadcast it to each integrated Agent(Participants) registered in the DF and declaring a "warehouse query/reservation" service; After receiving the CFP, the Participant evaluates the local resource usage, estimates the completion time, and the time and cost of invoking the corresponding WMS. If acceptable, it generates a Proposal(including the promised completion time, quotation, and confidence) and returns it to the Initiator. The Initiator collects all proposals until the time limit is exceeded or the minimum number of collections is reached, and then selects the winning bidder according to the comprehensive utility function; To avoid dimensionality, Min-Max normalization is first performed on each QoS attribute value[28]. Suppose there are n candidate agents. For candidate Agent j, the response time is t_j, the cost is c_j, the reputation is rep_j, and the normalized value is:

$$t'_j = \frac{\max(t) - t_j}{\max(t) - \min(t)} \tag{1}$$

$$c'_j = \frac{\max(c) - c_j}{\max(c) - \min(c)} \tag{2}$$

$$rep'_j = \frac{rep_j - \min(rep)}{\max(rep) - \min(rep)} \tag{3}$$

Then the comprehensive utility function U_j is defined as:

$$U_j = w_1 \cdot t'_j + w_2 \cdot c'_j + w_3 \cdot rep'_j \tag{4}$$

Where w_1, w_2, w_3 are weighting coefficients and $w_1 + w_2 + w_3 = 1$; The system selects the Agent corresponding to $\arg\max(U_j)$ as the winning bidder; Send an Accept message; Unsuccessful bidders receive Reject and release reserved resource estimates; The winning integrated Agent returns InformDone or Failure after completing the task, and the Initiator records and advances the workflow; CNP upgrades the service selection process from static table lookup to multi-agent bidding negotiation, adapting to the characteristic of continuous fluctuations in resource status in long-cycle tasks.

2) CONCURRENT EXECUTION AND SYNCHRONOUS CONVERGENCE CONTROL OF MULTI-TASK AGENTS

In the dynamic workflow model, the and-split gateway derives multiple parallel branches, each corresponding to an independent Task Agent invoking different Web services (such as synchronously querying CRM customer levels, ERP

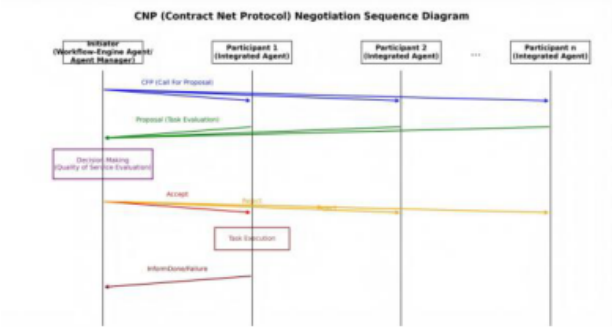

FIGURE 5. CNP Contract Network Protocol Negotiation Sequence diagram

credit limit, external business information database), AND the and-join gateway requires the aggregation of all branch results before proceeding; The orchestration engine Agent assigns relevant ids to each parallel branch and maintains counters and result staging maps at the Join node. After each branch Task Agent completes the service call, the result is attached to the Inform message and sent back to the engine listening mailbox. The engine updates the counter and Map, and unblocks the Join when the count reaches the expected number of branches[29]. Calculate the downstream gateway conditions and dispatch the next activity according to the predefined aggregation logic (rating =A if all three pass, rating =B if any level of warning); To prevent the entire branch from being deadlocked due to a network timeout, the Join node sets a maximum waiting time limit, and the timeout triggers the exception handler to handle according to the strategy (wait the longest, guess based on the existing result, switch to manual intervention), echoing the exception handling mechanism of the workflow engine in the literature.

3) RESOURCE CONFLICT DETECTION AND QOS-BASED AGENT OPTIMIZATION STRATEGY

Multiple concurrent long-cycle instances may contend for limited shared resources (such as the same ERP accounting period lock, specific high-precision machine tool capacity period), AgentManager maintains the resource lock table; The integrated Agent checks the feasibility of the resources before bidding. If pre-occupying the resources would result in an overbooking, it states in the Proposal that it cannot bid or suggests an alternative time slot. AgentManager cross-checks the resource lock table to prevent multiple allocations when adjudicating; QoS optimization is performed on a subset of available resources, introducing metrics such as "historical success rate" and "current load rate" in addition to response time and cost[30]. The load rate is calculated based on the number of tasks in processing attached to the Agent's recent heartbeat/maximum concurrency. The Agent with the highest overall score wins the bid, and the resource lock table writes the occupancy record accordingly, triggering the lock release when the task is completed or when Inform-Done is not received within the time limit; This double-layer filtering (resource feasibility +QoS preference) reduces the probability of rollback due to resource contention during the execution of long-cycle tasks.

C. ASYNCHRONOUS COMMUNICATION AND EXCEPTION SELF-HEALING HANDLING

1) FIPA-ACL MESSAGE FILTERING AND BEHAVIOUR BEHAVIOR SCHEDULING

Agent communication in MAS follows the FIPA-ACL specification, and messages contain performative(request,inform,refuse...) sender, receiver(AID), content, conversation-id, reply-with fields; Forming Agent and choreographer engine internal Agent according to JADE/WADE typical patterns registered multilayer Behaviour: CyclicBehaviour monitor email message distribution, According to the conversation - id and performative routing to the corresponding task processing Behaviour (ServiceSelectionBehaviour, InvokeWebServiceBehaviour, HandleFailureBehaviour, etc.; OneShotBehaviour handles stateless short tasks; WakerBehaviour implements timed retries and timeout detections; Message filters discard illegal or expired messages by matching ($sender \supseteq expected\ Agent \vee conversation\text{-}id = current\ session\ ID$); This mechanism supports the ordered queuing and context association of a large number of asynchronous interactions in long-cycle tasks, avoiding message cross-interference with different process instances.

2) RESELECTING OF TIMEOUT AND DOWNTIME AGENTS AND CHECKPOINT ROLLBACK

When the integrated Agent does not respond beyond the agreed ACK time limit, or when the AMS detection container is disconnected, the Agent Manager marks the Agent as suspected failure, queries the Agent repository to select a substitute Agent of the same function (refer to QoS preferred logic), resells the original service request and sets the retry count +1; If InformDone is not successful beyond the maximum retry threshold, throw a service unreachable exception to the workflow engine; The workflow engine acts according to the exception handler configuration: if defined as negligible, it fills the default null and continues (for non-critical peripheral validation); If defined as retry, suspend the current instance to save the checkpoint and wait for the administrator to specify an alternative service through StaffAgent or skip it manually; When defined as a rollback, this activity and committed compensatory transactions (if any) are revoked and reverted to the previous stable state for re-execution[31].

3) HUMAN-INTERVENED NODE AND ENGINE BLOCKING - WAKE-UP INTERACTION MECHANISM

In long-cycle tasks where some decisions have no clear algorithm (such as unconventional concession approval, technical deviation release), the workflow model associates the human task nodes with StaffAgent; When the engine runs to the SUSPENDED instance of the node, it sends a

request to the specified StaffAgent with the URL of the task form and the deadline; The StaffAgent pops up to display the application information, historical similar case references, and approval input box. After the user submits, the StaffAgent returns inform with approval conclusion and remarks. The engine, upon receiving it, verifies the match between the signature and the task instance ID, writes it back to the context variable, sets the instance to RUNNING, and drives the subsequent path according to the gateway condition; If overdue, the system can automatically dispatch the superior StaffAgent according to the upgrade rules or trigger the email /SMS prompt. This clogging - wake-up mode combines automated Agent collaboration with human judgment, ADAPTS to the requirement of compliance traceability and flexible discretion in the actual business of enterprises, and is in line with the flexible concept of dynamic workflows supporting human intervention in the literature.

IV. SYSTEM IMPLEMENTATION AND VALIDATION OF TYPICAL LONG-CYCLE BUSINESS SCENARIOS

A. DEVELOPMENT PLATFORM AND CORE CLASS DESIGN BASED ON JADE/WADE

1) WORKFLOWBEHAVIOUR FINITE STATE MACHINE AND ACTIVITY MAPPING

The system selects JADE as the Agent container foundation and extends the WADE plugin to obtain built-in support from the workflow engine. In the core implementation, the abstract class WorkflowBehaviour is defined to inherit from SequentialBehaviour. Internal maintenance enumeration State {INIT, RUNNING, SUSPENDED, COMPLETED, FAILED, ABORTED} and the Activity reference, Arrived in new activities per node WorkflowBehaviour by activity type instantiation corresponding AtomicBehaviour subclasses, respectively is CommonServiceInvokeBehaviour (ordinary activities, directly through the ESB call binding Web services), FlexibleActivityBehaviour (flexible activities, send an AgentManager service selection request and wait for the callback binding and then execute), HumanTaskBehaviour inform StaffAgent) (hang instance, The Activity completes the Behaviour.done() callback triggers the engine to advance to the next node, and the state transition is constrained by the transition conditions in the workflow meta-model. This mapping enables the navigation logic of the dynamic workflow engine to directly drive the AgentBehaviour lifecycle to achieve the isomorphic expression of process control and Agent computation[32].

2) AGENT LIFECYCLE MANAGEMENT (AMS/DF REGISTRATION AND UNREGISTRATION)

When the system starts, the resident Agent Manager starts first, registers with AMS and with DF for DF service types such as "Management Service Query", "Exception Handling", "Agent Dispatch ", etc. Then, several integrated Agent instances are started in batches according to the configuration file. After each instance to the AMS registration AID in the setup () method of registered its business with the DF ability (such as "urn: service: erp: inventory-check@1.0") and the QoS description ontology, When closed, perform takeDown() to undo DF registration and notify Agent Manager to update the list of available agents, and Agent Manager maintains the active Agent heartbeat table. Receive inform(id,load,status) from each integrated Agent every T_heartbeat second. Mark the missing three times as UNAVAILABLE and remove the agent from the candidate pool until the heartbeat is restored and the agent is rejoined. The DF service description uses OWL-S fragments to extend QoS attributes for semantic matching screening.

3) IMPLEMENTATION OF DYNAMIC BINDING OF THE ORGANIZATION TREE MODEL WITH USER-AGENT

The personnel organizational structure is regularly synchronized from the enterprise LDAP/HR system to the local organizational tree (OrgUnit→Position→Person triplet), Position is associated with the RBAC role set, and when the Staff Agent starts, the user enters the domain account, The background validates the credentials and loads the Position to which it belongs and writes the activation role set into the Session Context. The workflow engine parses the role requirements R configured for the human activity node. Query the list of Persons in the organization tree who have R and Status=Active under the current process business unit (BU) to generate a candidate pool for task assignment if the task has a "dynamically bound Agent proxy" option and the user has authorized an automatic proxy (such as a routine reporting type node), The system may specify an integrated Agent to simulate the user signature to perform data pre-filling but leave the final confirmation to the person. The user-agent binding relationship is refreshed in the permission cache in real time with login/logout and role change events to ensure that personnel job transfers are promptly perceived and realassigned when long-cycle tasks run across days[33].

B. ANALYSIS OF THE EXPERIMENTAL SCENARIOS AND OPERATION RESULTS

1) MODELING OF LONG-CYCLE MULTI-STAGE BUSINESS PROCESSES (SUCH AS INTELLIGENCE PROCESSING/ORDER APPROVAL)

Select the "Full process of customer custom orders" of mechanical manufacturing enterprises as the validation scenario, which covers customer requirement entry and initial qualification review (CRM interaction, including manual credit review nodes), technical solution design and BOM estimation (PLM interaction, flexible activity dynamic selection CAD parsing service), Material availability verification and purchase application (ERP inventory inquiry + external supplier inquiry and comparison, using CNP multi-agent negotiation to select the lowest weighted cost supplier), contract approval and stamping (including multi-level manual approval and legal review), production scheduling feedback (MES interface), shipping notification write-back CRM,

the entire process spans six heterogeneous systems, The expected execution cycle is 3 to 15 working days, including three flexible activity nodes and two multi-person approval nodes. In WADE, a dynamic workflow model described by XPDL is introduced to define flexible activity candidate service types and QoS templates, mapping each service to the DF service type of the corresponding integrated Agent[34]. Simulate 100 virtual order instances injected into the system according to the Poisson distribution to observe the operation.

TABLE II. Experimental Parameter Configuration Table

Parameter Names	Parameter values	Notes
Total number of process instances	100	Simulate the order process
Concurrent injection Rate (TPS)	10, 20, 40, 60, 80	Gradually increasing
Initial number of integrated agents	2	Scalable horizontally up to 4
Flexible number of activities/processes	3	Service dynamic selection
Manually approve the number of nodes/processes	2	StaffAgent intervention is required
CNP negotiate timeout	500 ms	
Join node maximum waiting time limit	2 s	Prevent deadlocks
Heartbeat detection intervals	1 s	
Agent Manager hot standby switching threshold	Three lost heartbeats	About 3 seconds

2) AGENT COLLABORATION PERFORMS LOG AND TASK FLOW TIMING ANALYSIS

Enable heartbeat logs and message tracking to record the timestamps of each CFP-Proposal-Accept-Inform interaction.

TABLE III. Comparison Table of Key Performance Metrics with Baseline Methods

Performance metrics	This paper's approach (Dynamic Workflow +Agent)	Baseline Approach (Traditional hard-coded orchestration)	Increase in magnitude
Average time spent on flexible activity service selection	187 ms	Not applicable (Service fixed)	-
CNP negotiated average additional overhead (3–7 candidates)	< 300 ms	0 ms	Increased flexibility overhead
Average Service Call response Time (ERP)	420 ms	420 ms	The same
Average Service Call Response Time (CRM)	680 ms	680 ms	The same
Average dwell time at manual approval nodes	8.4 hours	8.4 h	The same
Parallel branch Join timeout protection trigger rate	100% (set for 2 seconds)	May cause a deadlock	Significantly enhance robustness
Average process completion time (across days)	3.5 days	4.2 Days	16.7%
Process completion rate (including fault injection)	98%	About 75% (estimated)	23%

Note: The baseline approach assumes that the service address is fixed, there is no dynamic rebinding capability, and the change process will fail once the service fails or the change process fails. CNP negotiation has an average additional overhead of less than 300ms when the number of candidate agents is 3 to 7. The average response time for the winning Agent to actually invoke the Web Service depends on the back-end ERP/CRM, which is 420ms and 680ms respectively. The workflow instance is context-correct when the subprocesses are nested and switched. The parent and child instance ID chains are fully traceable, and the logs confirm the effectiveness of service selection and composition and the correctness of dynamic workflow engine navigation in the multi-agent system through the FIPA interaction protocol[35].

3) RELIABILITY AND SCALABILITY ASSESSMENT OF THE SYSTEM UNDER HIGH CONCURRENCY TASKS AND SINGLE POINT OF FAILURE

TABLE IV. High Concurrency Stress Test Results Table

Concurrent injection rate (TPS)	Number of Agent instances	Average response time (ms)	Request rate (%)	queuing	Average CPU utilization (%)
10	2	450	0		25
20	2	480	0		40
40	2	550	0		70
60	2	750	15		95
60	4	520	0		55
80	4	610	0		75

TABLE V. Fault Injection Test Results Table

Fault scenarios	Detection time (s)	Recovery time (s)	Process instance impact	Success rate
The Agent Manager main process is down	3.0	4.2	The new instance is briefly blocked, while the old instance remains intact	100%
Single Integrated Agent outage (ERP query)	1.0	1.5	Failed tasks are reselected and instances are restored	100%
The backend ERP service returns an exception message	Instant	Human intervention	The instance hangs up, awaiting human processing	0% (Expected behavior)

In the stress test, gradually increase the concurrent order instance injection rate from 10TPS to 80TPS to observe the lateral scaling performance of the integrated Agent container. At 60TPS for the initial 2 integrated Agent instances, some requests were queued. After increasing to 4 instances, the queue returned to zero and the average response time increased only slightly (<18%). AgentManager Hot standby switching test actively Kill the primary Manager process, standby Manager detects heartbeat loss within 4.2 seconds and completes DF re-registration and session recovery (loading the status of unfinished process instances from DB), during which the new process instance is briefly blocked (average 4.2 seconds) and then continues normally The running instance did not lose its state. After an ERP query Agent was killed, the unfinished flexible activity in the association was reselected by AgentManager to another Agent in the same group to succeed and retry successfully. The original instance completed normally after checkpoint recovery. In the case of the above fault injection, the final success rate of 100 instances was 98%, and two failures were due to the unexpected message returned by the back-end ERP triggering application-level exceptions that were not automatically restored after manual handling (which was expected to require manual intervention).

However, this study has certain limitations. The experimental scenarios are based on simulated data and will need to be verified in real enterprise production environments in the future. The current Agent negotiation strategy (weighted summation) is relatively simple. In the future, adaptive QoS weight adjustment methods based on reinforcement learning

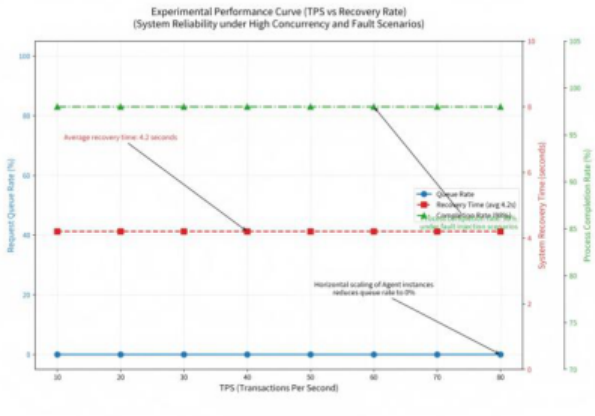

FIGURE 6. Experimental performance curve graph

can be explored to cope with more dynamic quality of service environments. Trust and security mechanisms across organizational boundaries will be the focus of the next study.

V. CLOSING REMARKS

This paper designs and implements an Agent workflow orchestration system for complex long-cycle tasks, effectively addressing the shortcomings of traditional orchestration methods in terms of service dynamics, process flexibility and security compliance through the separation architecture of control plane and execution plane, FIPA-based multi-agent collaboration mechanism and R-TBAC dynamic access control model, Experiments verified the reliability of the system in high concurrency and failure scenarios, and its lateral scalability and breakpoint continuation mechanism ensured the continuous execution of long-cycle tasks. Future work will focus on deployment verification in real production environments. Research on adaptive QoS weight optimization based on reinforcement learning and trust and security mechanisms across organizational boundaries to further enhance the practical value of the system in complex business ecosystems.

REFERENCES

[1] Q. Zhu, "The Internet of Agentic AI: Communication, Coordination, and Collective Intelligence at Scale," *arXiv:2606.12835 [cs.MA]*, Jun. 2026.

[2] J. Li et al., "Agentic Environment Engineering for Large Language Models: A Survey of Environment Modeling, Synthesis, Evaluation, and Application," *arXiv:2606.12191 [cs.CL]*, Jun. 2026.

[3] H.-T. Liao, C.-Y. Kao, and K. Ang, "Trustworthy Smart Fabs via Professional Proxies: Scaling Safe and Sustainable by Design (SSbD) through Industrial Data Spaces," *arXiv:2606.09227 [cs.CR]*, Jun. 2026.

[4] Y. Sun et al., "PerspectiveGap: A Benchmark for Multi-Agent Orchestration Prompting," *arXiv:2606.08878 [cs.CL]*, Jun. 2026.

[5] K. Wang et al., "Data Agents Under Attack: Vulnerabilities in LLM-Driven Analytical Systems," *arXiv:2606.08661 [cs.CR]*, Jun. 2026.

[6] D. Zhang and L. Liaotian, "Queen-Bee Agents: A BeeSpec-Centered Architecture for Governed Enterprise MCP Orchestration," *arXiv:2606.06545 [cs.SE]*, Jun. 2026.

[7] J. Xu, "OpenAgenet / OAN White Paper: Open Infrastructure for Trusted Agent Interconnection," *arXiv:2606.03161 [cs.MA]*, Jun. 2026.

[8] P. Qin, Q. Cao, and P. Xie, "ATLAS: Agentic Test-time Learning-to-Allocate Scaling," *arXiv:2606.01667 [cs.LG]*, Jun. 2026.

[9] R. Li et al., "Bridging Requirements and Architecture: Multi-Agent Orchestration with External Knowledge and Hierarchical Memory," *arXiv:2606.01385 [cs.SE]*, Jun. 2026.

[10] J. Zhu et al., "Recognize Your Orchestrator: An Entropy Dynamics Perspective for LLM Multi-Agent Systems," *arXiv:2606.01351 [cs.AI]*, May 2026.

[11] S. Sarangi et al., "EASE Configuration Facilitates A Reproducible Science of LLM Social Simulations," *arXiv:2605.30258 [cs.MA]*, May 2026.

[12] S. Gao, A. Fang, and M. Zitnik, "AutoScientists: Self-Organizing Agent Teams for Long-Running Scientific Experimentation," *arXiv:2605.28655 [cs.AI]*, May 2026.

[13] C. Luo et al., "From Physician Expertise to Clinical Agents: Preserving, Standardizing, and Scaling Physicians' Medical Expertise with Lightweight LLM," *arXiv:2603.23520 [cs.CL]*, Mar. 2026.

[14] J. Cook et al., "Training a Large Language Model for Medical Coding Using Privacy-Preserving Synthetic Clinical Data," *arXiv:2603.23515 [cs.CL]*, Mar. 2026.

[15] D. Joshi and I. Rekik, "HGNet: Scalable Foundation Model for Automated Knowledge Graph Generation from Scientific Literature," *arXiv:2603.23136 [cs.CL]*, Mar. 2026.

[16] Y. Wang et al., "RADAR: Closed-Loop Robotic Data Generation via Semantic Planning and Autonomous Causal Environment Reset," *arXiv:2603.11811 [cs.RO]*, Mar. 2026.

[17] Y. Yang et al., "Implement Kubernetes Pod-Level Remote Attestation for Confidential Workloads on dstack," *arXiv:2605.03323 [cs.CR]*, Jun. 2026.

[18] S. Ling et al., "Free-Riding in the AI Economy: Demystifying Logic Flaws in x402-Enabled Payment Systems," *arXiv:2605.30998 [cs.CR]*, May 2026.

[19] J. Chen et al., "NanoCP: Request-Level Dynamic Context Parallelism for Data-Expert Parallel Decoding," *arXiv:2605.21100 [cs.DC]*, May 2026.

[20] M. A. Bouchia et al., "DARTIC: Decentralized Anonymous Reputation at Scale for Trustworthy Crowdsourcing," *arXiv:2605.18146 [cs.CR]*, May 2026.

[21] S. Liu, S. Shin, and D. Deka, "Watts vs. Bytes: Turning Data Centers into Grid Assets via Storage Compute Co-Optimization," *arXiv:2605.16190 [eess.SY]*, May 2026.

[22] S. Wu et al., "Look One Step Ahead: Forward-Looking Incentive Design with Strategic Privacy for Proactive Service Provisioning over Air-Ground Integrated Edge Networks," *arXiv:2604.13635 [cs.NI]*, Apr. 2026.

[23] S. Liu, P. J. Elahi, and U. Varetto, "HPC-vQPU: A Service-Export Architecture for Virtual QPUs on Batch-Scheduled HPC Systems," *arXiv:2605.28845 [cs.DC]*, May 2026.

[24] M. Waseem et al., "Engineering a Governance-Aware AI Sandbox: Design, Implementation, and Lessons Learned," *arXiv:2603.03394 [cs.SE]*, Mar. 2026.

[25] P. Yan et al., "FWeb3: A Practical Incentive-Aware Federated Learning Framework," *arXiv:2603.00666 [cs.DC]*, Feb. 2026.

[26] M. Besta et al., "GraphSeek: Next-Generation Graph Analytics with LLMs," *arXiv:2602.11052 [cs.DB]*, Mar. 2026.

[27] M. Goenka, T. Pathak, and S. Asthana, "TessPay: Verify-then-Pay Infrastructure for Trusted Agentic Commerce," *arXiv:2602.00213 [cs.CR]*, Feb. 2026.

[28] H. KimLee et al., "QuantumSavory: Write Symbolically, Run on Any Backend – A Unified Simulation Toolkit for Quantum Computing and Networking," *arXiv:2512.16752 [quant-ph]*, Dec. 2025.

[29] H. Derouiche, Z. Brahmi, and H. Mazeni, "Agentic AI Frameworks: Architectures, Protocols, and Design Challenges," *arXiv:2508.10146 [cs.AI]*, Aug. 2025.

[30] S. Saxena, H. Farag, H. Turesson, and H. M. Kim, "Blockchain Based Transactive Energy Systems for Voltage Regulation," *arXiv:1907.08725 [cs.CR]*, Jul. 2019.

[31] S. Kaur, H. Kaur, and S. Kaur Sehra, "Modification of Contract Net Protocol(CNP): A Rule-Updation Approach," *arXiv:1312.4259 [cs.MA]*, Dec. 2013.

[32] B. Megdouda et al., "LLM-Based Digital Twin Intelligence for Application-Aware Network Selection in 6G Heterogeneous Wireless Networks," *arXiv:2606.12293 [eess.SP]*, Jun. 2026.

[33] Y. Chen et al., "Beyond Greedy Chunking: SLO-Aware Sliding-Window Scheduling for LLM Inference," *arXiv:2606.05933 [cs.DO]*, Jun. 2026.

[34] M. H. D. Saria Allahham and H. S. Hassanein, "How Reliable is Your Service at the Extreme Edge? Analytical Modeling of Computational Reliability," *arXiv:2602.16362 [cs.DO]*, Feb. 2026.

[35] G. A. Ubiali et al., "Joint Subarray Selection, User Scheduling, and Pilot Assignment for XL-MIMO," *arXiv:2601.13470 [eess.SP]*, Jan. 2026.