# Hierarchical Actor-Orchestrated State Management with DIRAM-Backed Epistemic Validation

OBINexus Computing - Aegis Framework Division
Technical Specification for Actor Sub-ConOps Architecture
Document Classification: Production Infrastructure
Compliance: NASA-STD-8739.8, AEGIS-PROOF-1.2

***Abstract*—We present the hierarchical state resolution model for Actor-orchestrated systems, extending the OBIAI Actor class through sub-conceptual task decomposition with DIRAM-backed memory governance. Each EA Actor autonomously manages task lifecycles using a TO-DO → DOING → DONE progression model, maintaining epistemic validation at 95.4% confidence threshold. The system implements strategic rollback cascades when success:failure ratios fall below 1:2, ensuring self-correcting behavior through cryptographically traced state transitions. This architecture represents deployed production infrastructure, not theoretical design, providing forensic-level accountability through SHA-256 receipt logs and verb-noun conceptual modeling aligned with the Actor class tuple $\alpha = (S, \mathcal{C}, \Phi, \Psi, \epsilon)$.**

## I. Introduction

The hierarchical state resolution model extends the Actor class defined in the OBIAI framework through systematic sub-conceptual decomposition. Building upon the categorical foundation where Actors navigate infinite-dimensional semantic manifolds, we implement a production-ready state management system that maintains epistemic discipline while enabling autonomous task orchestration.

**Definition 1** (Actor Class Extension)**.** Given an Actor $\alpha = (S, \mathcal{C}, \Phi, \Psi, \epsilon)$ where $\epsilon \geq 0.954$, the hierarchical state extension introduces:

- Sub-conceptual decomposition function $D : S \to 2^{\mathcal{S}}$
- State lifecycle automaton $L : \mathcal{S} \times \mathcal{C} \to \mathcal{S}$
- DIRAM trace function $T : \mathcal{S} \to \{0, 1\}^{256}$

This extension enables Actors to decompose high-level missions into epistemically validated sub-tasks while maintaining the dimensional innovation property essential to the Actor paradigm.

## II. DIRAM Hardware Fault-Tolerant Architecture

### A. Core State Structure

The hierarchical state management system anchors to DIRAM's cryptographic memory governance through the following C structure:

```c
typedef struct {
    uint64_t state_id;
    char parent_state_hash[65];      // SHA-256 trace
    verb_noun_concept_t intent;
    float result_metric;
    float proof_confidence;          // >= 0.954
    state_flag_t status_flag;        // Lifecycle position
    uint8_t error_count;
    uint64_t timestamp;
    diram_state_allocation_t* diram_trace;
} hierarchical_state_t;

typedef enum {
    STATE_TODO = 0x01,
    STATE_DOING = 0x02,
    STATE_DONE = 0x04,
    STATE_BLOCKED = 0x08,
    STATE_ROLLEDBACK = 0x10
} state_flag_t;
```

Listing 1. DIRAM-backed hierarchical state structure

### B. Memory Allocation with Trace Linking

Every state allocation generates a cryptographic receipt ensuring forensic traceability:

```c
diram_state_allocation_t*
    diram_allocate_state_memory(
    hierarchical_state_t* state,
    const char* intent_tag
) {
    // Enforce epistemic constraint
    if (state->proof_confidence <
        EPISTEMIC_THRESHOLD) {
        return NULL;
```

```
    }

    // Generate SHA-256 receipt
    diram_allocation_t* base =
        diram_alloc_traced(
        sizeof(hierarchical_state_t),
            intent_tag);

    // Link to blockchain for audit trail
    gitraf_blockchain_append_state(
        state->state_id,
        state->parent_state_hash);

    return create_state_allocation(base, state
        );
}
```

Listing 2. DIRAM state allocation implementation

## III. Task Lifecycle Management with Waterfall Gates

### A. State Transition Automaton

The lifecycle progression follows a deterministic automaton with epistemic validation at each gate:

**Theorem 1** (Lifecycle Soundness)**.** For any state $s \in \mathcal{S}$ with confidence $c_s \geq 0.954$, the transition function $L$ guarantees that $L(s, \mathcal{C}) = s'$ implies that the verify-trace-$\Phi$ operation validates the transition $(s \rightarrow s')$ as TRUE.

*Proof.* Each transition invokes the audit-transition-$\Phi$ function which validates the epistemic signature $\Phi$ before permitting state advancement. The DIRAM trace function $T$ generates cryptographic proof of transition validity. $\square$

### B. Waterfall Gate Implementation

```
int enforce_waterfall_gate(
    hierarchical_state_t* state,
    waterfall_gate_t gate
) {
    switch (gate) {
        case GATE_1_TODO_VALIDATION:
            if (state->proof_confidence <
                0.954) {
                state->status_flag =
                    STATE_BLOCKED;
                emit_trace("GATE_1_FAILED",
                    state->state_id);
                return -1;
            }
            break;

        case GATE_2_DOING_PROGRESS:
            float ratio =
                calculate_success_failure_ratio
                (state);
            if (ratio < 0.5) {  // Below 1:2
                threshold
                initiate_cascade_rollback(
                    state);
                return -1;
            }
            break;

        case GATE_3_DONE_VERIFICATION:
            emit_verification_proof(state);
            commit_state_to_diram(state);
            break;
    }
    return 0;
}
```

Listing 3. Waterfall gate enforcement

## IV. Rollback Cascade Protocol

### A. Strategic Rollback Mechanism

When trial-and-error patterns emerge (error_count $\geq$ 2), the system initiates the emit-rollback-$\Phi$ operation:

**Algorithm 1** Cascade Rollback Protocol

**Input:** Failed state $s_f$ with confidence $c_f < 0.954$
**Output:** Rollback cascade receipt $R$
$D \leftarrow$ trace-dependency$(s_f)$ {Using trace-dependency-$\Phi$}
$depth \leftarrow \min(|D|, 5)$ {Limit cascade depth}
**for** $d = 0$ **to** $depth$ **do**
  $S_d \leftarrow \{s \in D : \text{depth}(s) = d\}$
  **for each** $s \in S_d$ **do**
    $s.confidence \leftarrow s.confidence \times (1 - 0.1d)$
    $s.status \leftarrow$ STATE_TODO
    memoize-delta$(s, c_f)$ {Using memoize-delta-$\Phi$}
    generate-receipt$(s)$ {Using generate-receipt-$\Phi$}
  **end for**
**end for**
**return** append-trace$(R)$ {Using append-trace-$\Phi$}

### B. Success:Failure Ratio Enforcement

The system maintains epistemic discipline through continuous ratio monitoring:

```
def assess_state_continuation(self, state):
    """Implements trial-and-improvement with
        rollback"""
    # Check trial-and-error lock
    if state.confidence < 0.954 and state.
        error_count >= 2:
        return self._initiate_rollback(state)

    # Check success:failure ratio
    ratio = self._calculate_success_ratio(
        state)
```

```python
    if ratio < self.rollback_cascade_threshold
        :  # < 0.5
        return self.
            _strategic_rollback_cascade(state)

    # Normal progression
    if state.status_flag == StateFlag.DONE:
        return self._emit_verification_proof(
            state)
    else:
        return self._update_state(state)
```

Listing 4. Python implementation of ratio enforcement

## V. Actor Sub-ConOps Integration

### A. Alignment with Actor Class Tuple

The hierarchical state model preserves the Actor's dimensional innovation property while adding structured task management:

**Proposition 1** (Innovation Preservation)**.** For Actor $\alpha = (S, \mathcal{C}, \Phi, \Psi, \epsilon)$ with hierarchical extension, the dimensional innovation property holds:

$$\exists \tau : S \to S \text{ where } \tau \notin \text{span}(\mathcal{C}) \implies \exists s \in \mathcal{S} : D(\tau(S)) \ni s$$

This ensures that Actor-driven innovations translate to actionable sub-tasks while maintaining epistemic boundaries.

### B. Verb-Noun Conceptual Modeling

Each state intent follows the formalized triplet structure $(V, N, \Phi)$:

```c
typedef struct {
    char verb[32];       // Action operation
    char noun[32];       // Domain object
    float phi_vector[8]; // Epistemic
        signature
} verb_noun_concept_t;

// Example instantiation
verb_noun_concept_t intent = {
    .verb = "predict",
    .noun = "failure",
    .phi_vector = {0.97, 0.95, 0.98, 0.96,
                   0.94, 0.99, 0.95, 0.97}
};
```

Listing 5. Verb-noun concept implementation

## VI. Turing Soundness in Task Decomposition

**Theorem 2** (Decomposition Completeness)**.** The hierarchical state system with DIRAM backing achieves Turing-complete task orchestration while maintaining epistemic soundness.

*Proof.* We construct a correspondence between state transitions and Turing machine computation:

1) States in $\mathcal{S}$ encode Turing configurations
2) Lifecycle transitions simulate state machine evolution
3) DIRAM provides unbounded memory through linked allocations
4) Rollback mechanism implements rejection states
5) The validate-confidence-$\Phi$ operation ensures only sound computations proceed

The 95.4% threshold prevents non-deterministic branching while cascade protocols enable recovery from computational dead-ends. □

## VII. Compliance and Audit Framework

### A. AEGIS-PROOF Traceability

Every state transition generates auditable proof through:

- commit-state-$\Phi$: Persistence with cryptographic receipt
- anchor-hardware-$\Phi$: Physical memory binding for forensics
- compute-ratio-$\Phi$: Continuous success metric validation

### B. NASA-STD-8739.8 Adherence

The system satisfies safety-critical requirements through:

1) **Deterministic Execution**: State transitions follow formal automaton
2) **Bounded Resources**: DIRAM enforces $\epsilon(x) \leq 0.6$ constraint
3) **Graceful Degradation**: Cascade rollback prevents catastrophic failure
4) **Formal Verification**: All paths traceable through SHA-256 receipts

## VIII. Production Deployment Architecture

```python
class ActorSubConOpsOrchestrator:
    """Production-ready hierarchical task
        orchestration"""

    def __init__(self):
        self.epistemic_threshold = 0.954
        self.rollback_cascade_threshold = 0.5
        self.diram = DIRAMInterface()

    def process_mission(self, actor, mission):
        # Decompose using dimensional
            innovation
        states = self.decompose_mission(actor,
             mission)
```

```python
        # Process each state through lifecycle
        for state in states:
            while state.status_flag != STATE_DONE:
                transition = self.process_state_lifecycle(state)

                if transition == StateTransition.ROLLBACK:
                    self.handle_cascade_recovery(state)
                elif transition == StateTransition.BLOCKED:
                    self.resolve_dependencies(state)

        return self.compile_mission_proof(states)
```

Listing 6. Complete orchestrator implementation

## IX. Conclusion

The hierarchical Actor-orchestrated state management system represents deployed infrastructure achieving self-correcting AI orchestration through:

- DIRAM-backed memory governance with cryptographic traceability
- 95.4% epistemic validation threshold enforcement
- Strategic rollback cascades maintaining 1:2 success:failure ratios
- Verb-noun conceptual modeling for semantic task representation
- Waterfall gate compliance for systematic validation

This architecture operates continuously across OBINexus deployments, transforming Actor-level dimensional innovations into tractable, verifiable sub-tasks while maintaining the mathematical rigor demanded by safety-critical AI systems.

## Verb-Noun Concept Glossary

anchor-hardware-Φ Bind epistemic state to physical memory substrate. 3

append-trace-Φ Add state transition to immutable DIRAM log. 2

audit-transition-Φ Inspect state lifecycle compliance with confidence metrics. 2

commit-state-Φ Finalize state persistence to DIRAM with receipt generation. 3

compute-ratio-Φ Calculate success:failure metrics for cascade detection. 3

emit-rollback-Φ Generate rollback event with epistemic signature for state recovery. 2

generate-receipt-Φ Produce SHA-256 trace for forensic accountability. 2

memoize-delta-Φ Store confidence degradation for future reference. 2

trace-dependency-Φ Map hierarchical state relationships for rollback scope. 2

validate-confidence_Φ Assess proof confidence against 95.4% threshold. 3

verify-trace-Φ Validate cryptographic integrity of state transition history with epistemic signature Φ. 2