PhenoTriple Model in Gosilang

Core Architecture

The **PhenoTriple Model** forms the foundation of phenomenological networking in Gosilang, consisting of three interconnected components:

```
// PhenoTriple: The fundamental unit of phenomenological data
@thread_safe(level=MAX)
actor PhenoTriple {
    token_type: PhenoTokenType,
    token_value: PhenoTokenValue,
    memory: PhenoMemory
}
```

1. PhenoTokenType

Definition: The categorical classification of network events and data within the phenomenological frame.

```
enum PhenoTokenType {
    // Node-level tokens
    NODE_IDENTITY,      // Unique node identifier
    NODE_STATE,         // Current operational state
    NODE_DEGRADATION,   // Degradation event marker

    // Cluster-level tokens
    CLUSTER_TOPOLOGY,   // Cluster formation token
    CLUSTER_CONSENSUS,  // Agreement protocol token
    CLUSTER_MIGRATION,  // Data movement token

    // Frame tokens
    FRAME_REFERENCE,    // Subjective context marker
    FRAME_TRANSFORM,    // Context shift event
    FRAME_COLLAPSE,     // Frame degradation event
}
```

2. PhenoTokenValue

Definition: The actual data payload carried within the phenomenological network, maintaining type safety and context.

```
@hardware_isolated
struct PhenoTokenValue {
    // Core value storage
    raw_data: Vec<u8>,
    encoded_data: Vec<u8>,  // AVL-Trie encoded form

    // Phenomenological metadata
    origin_frame: FrameID,
    degradation_score: f32,  // 0.0 = healthy, 1.0 = fully degraded
    timestamp: u64,

    // Thread-safe accessors
    fn get_typed<T>() -> Result<T, PhenoError> {
        // Type-safe extraction with frame validation
    }
}
```

3. PhenoMemory

Definition: The persistent, thread-safe memory model that maintains phenomenological state across network degradation events.

```
@constant_time(verified=true)
actor PhenoMemory {
    // AVL-Trie hybrid storage
    avl_root: Option<AVLNode>,
    trie_map: HashMap<PhenoPath, PhenoTriple>,

    // Degradation tracking
    degradation_events: RingBuffer<DegradationEvent>,
    recovery_snapshots: Vec<FrameSnapshot>,

    // Memory operations
    fn store(triple: PhenoTriple) -> Result<(), MemoryError> {
        // Thread-safe storage with AVL balancing
    }

    fn retrieve(path: PhenoPath) -> Option<PhenoTriple> {
        // O(log n) retrieval with trie optimization
    }

    fn handle_degradation(event: DegradationEvent) {
        // Graceful degradation without data loss
    }
}
```

Integration with AVL-Trie Structure

```
// AVL-Trie node for phenomenological data
struct PhenoAVLTrieNode {
    // AVL properties
    height: i32,
    balance_factor: i8,

    // Trie properties
    prefix: Vec<u8>,
    children: HashMap<u8, Box<PhenoAVLTrieNode>>,

    // Phenomenological data
    triple: Option<PhenoTriple>,
    frame_context: FrameReference,

    // P2P network properties
    peer_nodes: Vec<NodeID>,
    cluster_id: Option<ClusterID>,
}
```

Thread-Safe Event Handling

```
// Degradation event processing
@policy(#noghosting)
actor DegradationHandler {
    fn process_event(event: NetworkDegradationEvent) {
        // Create PhenoTriple for the event
        let triple = PhenoTriple {
            token_type: PhenoTokenType::NODE_DEGRADATION,
            token_value: PhenoTokenValue::from_event(event),
            memory: self.allocate_pheno_memory()
        };

        // Bubble up through topology
        self.bubble_to_cluster(triple);

        // No locks, just phenomenological consensus
        self.achieve_frame_consensus(triple);
    }
}
```

Example Usage

```
// Node-to-node communication with phenomenological awareness
let sender_triple = PhenoTriple {
    token_type: PhenoTokenType::NODE_IDENTITY,
    token_value: PhenoTokenValue::new(node_id, current_frame),
    memory: PhenoMemory::allocate_isolated()
};

// Cluster-level degradation handling
match network.detect_degradation() {
    Some(degradation) => {
        let degrade_triple = PhenoTriple {
            token_type: PhenoTokenType::FRAME_COLLAPSE,
            token_value: PhenoTokenValue::from_degradation(degradation),
            memory: cluster.shared_pheno_memory()
        };

        // Thread-safe propagation
        cluster.propagate_phenomenological_event(degrade_triple);
    },
    None => continue_normal_operation()
}
```

This PhenoTriple Model ensures that Gosilang maintains thread safety while handling complex network topologies and degradation events through phenomenological awareness.