Technical Specification: Active Brain Computing Interface via Puppet Protocol Relay

Semantic Decoherence Mitigation via Polyglot Gossip Programming

Version: 1.0
Status: Draft Technical Specification
Project: OBINexus - Active BCI with Human-in-the-Loop Coherence
Date: October 2025

Executive Summary

This specification defines an **Active Brain-Computer Interface (BCI)** system that leverages the **Puppet Method Protocol** as a relay mechanism to mitigate semantic decoherence in actor-based distributed systems. The architecture addresses a fundamental challenge: maintaining interpretive coherence when context flows across neural, physical, and computational boundaries through polyglot gossip programming patterns.

Core Innovation

The system detects **pre-conscious neural activity** (200-300ms before conscious awareness) and relays this through tangible **puppet interfaces** to create a human-interpretable feedback loop that prevents semantic drift in distributed actor systems.

System Name

OBINexus NeuroSpark BCI with Puppet Protocol Relay

---

1. System Architecture

1.1 Three-Layer Architecture

```
┌─────────────────────────────────────────┐
│ Layer 1: Neural Detection (BCI)         │
│ - EEG Pre-conscious Signal Capture      │
│ - 200-300ms Pre-awareness Window        │
│ - eze/uche/motor pattern recognition    │
└─────────────────┬───────────────────────┘
                  │
                  ▼
┌─────────────────────────────────────────┐
│ Layer 2: Puppet Protocol Relay          │
│ - Physical Identity Anchor              │
│ - Haptic/Tactile Feedback               │
│ - Human-Interpretable State Translation │
└─────────────────┬───────────────────────┘
```

```
         │
         ▼
┌─────────────────────────────────────┐
│  Layer 3: Actor System (Computational)  │
│  - Polyglot Gossip Message Passing     │
│  - Error Bubbling (not propagation)    │
│  - Active State Machine with Memory    │
└─────────────────────────────────────┘
```

1.2 System Components

```python
class ActiveBCISystem:
    """
    Distributed BCI with Puppet Protocol relay for semantic coherence
    """
    def __init__(self):
        self.neural_layer = NeuralDetectionModule()
        self.puppet_relay = PuppetProtocolInterface()
        self.actor_system = PolyglotGossipActors()
        self.coherence_monitor = CoherenceOperatorEngine()
        self.error_bubbler = ActiveStateMachine()
```

---

2. Neural Detection Layer (BCI)

2.1 Hardware Specifications

EEG Headset Design:

- **Type:** Dry electrode array (64-channel configuration)
- **Wireless Protocol:** Bluetooth 5.2 + proprietary ultra-low latency link
- **Sampling Rate:** 2048 Hz minimum (sub-millisecond temporal resolution)
- **Resolution:** 24-bit ADC for high signal fidelity
- **Safety Features:**
  - Isolated circuitry preventing electrical hazards
  - Spark-proof charging system
  - Active thermal management (<40°C operating temperature)
- **Power:** Rechargeable Li-ion, 8+ hours continuous operation

2.2 Pre-Conscious Neural Detection

Temporal Detection Window:

- **Target Range:** 200-300ms before conscious awareness
- **Rationale:** Capture neural "ignition" patterns before thought consolidation

Three Detection Modalities:

1. **eze (Visual Thought Monitoring)**
   - Monitors visual cortex activity
   - Detects image formation patterns
   - Gamma band (30-80 Hz) primary indicator
2. **uche (Knowledge Access/Internal Monologue)**
   - Tracks internal dialogue initiation
   - Language processing area monitoring
   - Alpha/theta band (4-13 Hz) correlation
3. **Motor Intent Prediction**
   - Pre-motor cortex activation detection
   - Action intention signals before movement
   - Beta band (13-30 Hz) desynchronization

2.3 Signal Processing Pipeline

```
Raw EEG Signal (64 channels @ 2048 Hz)
  ↓
[Noise Filtering]
  ├── 50/60 Hz notch (power line rejection)
  ├── 0.5-100 Hz bandpass
  └── Adaptive artifact removal (blink/muscle)
  ↓
[Feature Extraction]
  ├── Time-domain: Event-related potentials (ERPs)
  ├── Frequency-domain: Power spectral density
  └── Spatial: Independent component analysis (ICA)
  ↓
[Pattern Classification]
  ├── CNN for visual cortex patterns
  ├── RNN for temporal sequence (internal monologue)
  └── SVM for motor imagery
  ↓
[Neural Spark Detection]
  └── Pre-conscious intention identified
  ↓
[Relay to Puppet Protocol Layer]
```

2.4 Performance Targets

- **Detection Latency:** <50ms from neural event to classification
- **Classification Accuracy:** >90% for trained patterns
- **False Positive Rate:** <5% per minute
- **Battery Life:** 8+ hours continuous monitoring

---

3. Puppet Protocol Relay Layer

3.1 Conceptual Foundation

The **Puppet Method Protocol** transforms abstract neural signals into tangible, human-interpretable states through physical proxy objects (puppets). This creates a **semantic anchor** that prevents drift between internal experience and external expression.

3.2 Clinical Implementation: Non-Verbal Autism Therapy

Real-World Use Case: A 10-year-old non-verbal child with severe autism uses the system to communicate basic needs and emotional states.

Setup Process:

1. **Hardware Configuration**
   - Child fitted with lightweight dry-electrode EEG headset
   - Soft tactile puppet selected (e.g., favorite animal representation)
   - Parent dashboard initialized for real-time monitoring
2. **Relay Therapy Protocol (RTP)**
   - EEG detects subtle neural patterns (excitement, frustration, calm states)
   - System identifies brainwave signatures:
     - Delta waves (0.5-4 Hz): Deep calm/comfort states
     - Theta waves (4-8 Hz): Drowsiness/internal focus
     - Alpha waves (8-13 Hz): Relaxed alertness
     - Gamma waves (>30 Hz): Active engagement/excitement
3. **Puppet Feedback Loop**
   - Neural pattern triggers puppet action:
     - **Excitement (gamma):** Puppet rhythmic movement

    - **Calm (delta):** Puppet gentle breathing simulation
    - **Need state (theta spike):** Puppet audio cue (specific tone)
  - Reinforcement learning: brain-puppet association strengthening

4. **Caregiver Interface**
   - Dashboard displays real-time neural rhythm visualization
   - Pattern library: "Hunger signature," "Comfort request," "Joy expression"
   - Encrypted logging (GDPR-compliant)
   - Zero technical expertise required

3.3 Puppet Protocol Technical Architecture

```python
class PuppetProtocolRelay:
    """
    Physical identity anchor for neural state translation
    """
    def __init__(self):
        self.identity_anchor = SovereignIdentityModule()
        self.boundary_enforcer = OwnershipProtocolSystem()
        self.multimodal_translator = HapticFeedbackEngine()
        self.validation_network = ConsensusValidator(validators=21)

    def relay_neural_state(self, eeg_pattern):
        """
        Translate neural pattern to puppet action
        """
        # Classify intent from neural signature
        intent = self.classify_intent(eeg_pattern)

        # Map to puppet action space
        puppet_action = self.identity_anchor.map_to_expression(intent)

        # Enforce ownership boundaries (no external override)
        if self.boundary_enforcer.validate_sovereignty(puppet_action):
            self.execute_puppet_feedback(puppet_action)
            self.log_state_transition(eeg_pattern, puppet_action)

        return puppet_action
```

3.4 Ethical Design Principles

1. **Identity Sovereignty:** Puppet represents inviolable personal identity

2. **Boundary Integrity:** No external correction or override of user's puppet
3. **Autonomous Design:** User-driven puppet customization
4. **Anti-Coercion:** System never "corrects" behavior, only amplifies expression
5. **Accessibility:** Hardware cost target <$3-10 for puppet components

3.5 Data Privacy Framework

- **GDPR-Compliant:** Full user data sovereignty
- **Family-Centered Consent:** Parents control all child neural data
- **Local Processing:** EEG analysis on-device where possible
- **Encrypted Transmission:** End-to-end encryption for cloud sync
- **Open-Source Auditability:** Code publicly verifiable for trust

---

4. Actor System: Polyglot Gossip Programming

4.1 Semantic Decoherence Problem

Challenge: In distributed actor systems communicating across language boundaries (polyglot), semantic meaning drifts due to:

- Context loss in message serialization
- Asynchronous state divergence
- Lack of shared memory between actors

Solution: Use human-interpretable puppet relay as a **coherence checkpoint** where semantic drift is corrected through human-in-the-loop validation.

4.2 Error Bubbling Model (Gosilang)

Core Principle: Errors bubble UP to witnesses, not propagate DOWN to children.

```
c
```

```
// Example: Gosilang Error Bubbling in C-style pseudocode

// Define max function
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Define min function
int min(int a, int b) {
    return (a < b) ? a : b;
}

// Clamp function that uses both (bubbles errors upward)
int clamp(int value, int min_val, int max_val) {
    // Compute maximum of value and min_val
    int lower_bound = max(value, min_val);

    // Compute minimum of result and max_val
    int clamped = min(lower_bound, max_val);

    // If errors occur in max() or min(), they bubble here
    // to the calling context (main), not propagate downward
    return clamped;
}

int main() {
    int result = clamp(150, 0, 100);  // Should return 100
    // Errors from clamp→max→min bubble here for handling
    return 0;
}
```

4.3 Active State Machine Architecture

Key Distinction: Traditional state machines are **passive** (they define states but don't act autonomously). This system implements an **active state machine** that:

1. **Remembers Context:** Preserves call stack and heap memory state
2. **Acts Autonomously:** Triggers next actions based on prior state
3. **Witnesses Events:** Observer pattern tracks all state transitions
4. **Bubbles Errors:** Failures propagate to parent context, not children

```
python
```

```
class ActiveStateMachine:
    """
    Self-aware state machine with error bubbling
    """
    def __init__(self):
        self.state_memory = []  # Preserves time and space
        self.call_stack = []
        self.observers = []  # Watcher/listener registry

    def execute_with_context(self, function, *args):
        """
        Execute function with full context preservation
        """
        # Remember current state before action
        self.state_memory.append({
            'timestamp': now(),
            'call_stack': self.call_stack.copy(),
            'function': function.__name__,
            'args': args
        })

        try:
            result = function(*args)
            self.notify_observers('success', result)
            return result

        except Exception as error:
            # Error bubbles up to calling context
            self.notify_observers('error', error)
            self.bubble_error_to_parent(error)

            # System remembers: "I was doing X when error occurred"
            # Next call can reference this context
            raise
```

4.4 Error Classification System

Four-Level Error Model:

Level	Name	Behavior	Use Case
0	**OK**	Continue execution	Normal operation path
1	**Warning**	Log but continue	Recoverable anomaly
2	**Error**	Stop current task, bubble up	Syntax error, type mismatch
3	**Exception**	Invoke special handler	Silent error requiring algorithm adjustment
4	**Panic**	Kill process immediately	Catastrophic failure, failsafe trigger

Exception vs Error:

- **Error:** Something went wrong that stops execution
- **Exception:** Something needs handling but isn't necessarily wrong (e.g., edge case in algorithm)

Panic Threshold:

- System must exit immediately
- Human intervention required
- Cannot continue safely
- Example: BCI detects potentially harmful neural pattern

4.5 Observer-Watcher-Actor Pattern

Components:

1. **Watchers/Listeners:** Observe state changes passively
2. **Observers:** Track events and log state transitions
3. **Bubblers:** Propagate errors upward through call stack
4. **Actors:** Execute actions and generate events

Event Processing Flow:

```
Neural Event Detected (BCI Layer)
   ↓
[Watcher: Logs event timestamp and pattern]
   ↓
[Observer: Classifies event as eze/uche/motor]
   ↓
[Bubbler: If classification error, bubble to parent]
   ↓
[Actor: Execute puppet action based on classification]
   ↓
[Watcher: Logs puppet state change]
   ↓
[Observer: Monitors coherence between neural→puppet→actor]
```

---

5. Coherence Operator Engine

5.1 Coherence Threshold

Target: Maintain >95.4% semantic coherence between layers

Coherence Definition:

```
Coherence = (Successful_Neural_to_Puppet_Mappings / Total_Mappings) × 100%
```

5.2 Set-Based Coherence Operators

Union Operator (A ∪ B): Combines two semantic state spaces while preserving distinct identities.

Example:

- Set A: Neural patterns detected by BCI
- Set B: Puppet actions executed
- A ∪ B: Complete state space of system behavior

Intersection Operator (A ∩ B): Identifies shared semantic meaning between layers.

Example:

- A ∩ B: Moments where neural intent perfectly matches puppet expression
- High intersection = high coherence

5.3 Clamp Function for Coherence Boundaries

```c
// Coherence clamping to maintain operational range
float clamp_coherence(float measured_coherence,
                     float min_threshold,
                     float max_acceptable) {
   // Ensure coherence stays within 95.4% - 100% range
   float lower = max(measured_coherence, min_threshold);  // At least 95.4%
   float clamped = min(lower, max_acceptable);           // At most 100%
   return clamped;
}

// Usage
float current_coherence = calculate_coherence();
float safe_coherence = clamp_coherence(current_coherence, 95.4, 100.0);

if (safe_coherence < 95.4) {
   trigger_human_in_loop_validation();  // Puppet relay intervention
}
```

5.4 Active Coherence Maintenance

Passive Systems: Preserve memory but not time and space context
Active Systems (OBINexus): Preserve memory, time, and space

Mechanism:

1. System detects coherence drop (neural→puppet mismatch)
2. Active state machine remembers: "I'm doing task X with context Y"
3. System pauses and requests human-in-the-loop validation via puppet
4. Human corrects semantic interpretation through puppet interaction
5. System learns corrected mapping and resumes with restored coherence

---

6. Implementation Roadmap

Phase 1: Proof-of-Concept (Months 1-6)

Objectives:

- Validate 8-channel EEG pre-conscious detection
- Build basic puppet relay prototype
- Implement simple error bubbling in C

Deliverables:

- Functional 8-channel wireless EEG headset
- Single puppet with 3 feedback modes (haptic, audio, visual)
- Gosilang error bubbling compiler prototype
- Safety validation testing complete

Milestones:

- M1.1: EEG hardware functional (Month 2)
- M1.2: Puppet relay detects 2+ neural patterns (Month 4)
- M1.3: Error bubbling operational in test environment (Month 5)
- M1.4: Safety certification initiated (Month 6)

Phase 2: Alpha System (Months 7-12)

Objectives:

- Scale to 32-channel EEG for improved spatial resolution
- Implement real-time eze/uche/motor pattern recognition
- Deploy puppet relay in clinical autism therapy pilot

Deliverables:

- 32-channel commercial-grade headset

- Multi-modal pattern classifier (>80% accuracy)
- Parent dashboard for real-time monitoring
- 10-participant autism therapy pilot study

Milestones:

- M2.1: 32-channel hardware operational (Month 8)
- M2.2: Pattern recognition >80% accuracy (Month 10)
- M2.3: Pilot study recruitment complete (Month 11)
- M2.4: Alpha system deployed in 5 homes (Month 12)

Phase 3: Production System (Months 13-18)

Objectives:

- Full 64-channel system for research-grade fidelity
- ▌ 90% pattern classification accuracy
- Regulatory approval (FDA Class II, CE marking)
- Open-source toolkit release

Deliverables:

- 64-channel production headset (<$500 BOM)
- Complete pattern recognition suite (eze/uche/motor)
- Clinical validation study (n=50 participants)
- Open-source SDK and documentation

Milestones:

- M3.1: 64-channel production design finalized (Month 14)
- M3.2: >90% accuracy achieved (Month 15)
- M3.3: Regulatory submissions complete (Month 17)
- M3.4: Public release and community launch (Month 18)

---

7. Compliance and Safety

7.1 Medical Device Standards

- **IEC 60601-1:** Electrical safety for medical devices
- **ISO 13485:** Quality management for medical devices

- **FDA Class II:** 510(k) premarket notification (US)
- **CE Marking:** European medical device regulation (MDR)

7.2 Wireless and RF Safety

- **FCC Part 15B:** Unintentional radiators (Bluetooth)
- **Bluetooth SIG Qualified:** Protocol compliance
- **SAR Limits:** Specific absorption rate <1.6 W/kg

7.3 Data Privacy

- **GDPR Compliance:** EU data protection regulation
- **HIPAA:** Health Insurance Portability and Accountability Act (US)
- **COPPA:** Children's Online Privacy Protection Act (for pediatric use)

7.4 Spark and Fire Prevention

- **Electrical Isolation:** Redundant isolation between EEG and processing
- **Thermal Management:** Active cooling below 40°C
- **Battery Safety:** UL 2054 certified lithium-ion cells
- **Charging Safety:** Spark-proof connector design

---

8. Technical Risk Analysis

8.1 High-Priority Risks

Risk	Probability	Impact	Mitigation
False positive neural detection	High	Medium	Multi-layer validation, human-in-loop verification
EEG signal quality degradation	Medium	High	Adaptive electrode contact monitoring, real-time impedance check
Wireless interference	Medium	Medium	Frequency hopping spread spectrum, error correction
Semantic drift in actor system	High	High	Coherence operator enforcement, puppet relay checkpoints
Regulatory approval delays	Medium	High	Early engagement with FDA/CE notified bodies

8.2 Safety Failure Modes

Scenario 1: Electrical Fault

- **Detection:** Voltage/current anomaly sensors
- **Response:** Immediate power disconnect, audible alert
- **Recovery:** Manual inspection required before restart

Scenario 2: Coherence Collapse (<95.4%)

- **Detection:** Real-time coherence operator monitoring
- **Response:** Pause autonomous actions, trigger puppet relay validation
- **Recovery:** Human corrects interpretation, system resumes

Scenario 3: Battery Overheat

- **Detection:** Thermal sensors at 38°C threshold
- **Response:** Throttle processing, increase cooling, alert user
- **Recovery:** Automatic shutdown at 42°C, cool-down period

---

9. Open-Source Ecosystem

9.1 Repository Structure

```
obinexus-neurospark/
├── hardware/
│   ├── eeg-headset/        # PCB designs, BOM, assembly
│   ├── puppet-relay/       # Puppet hardware schematics
│   └── testing-rigs/       # QA and calibration fixtures
├── firmware/
│   ├── eeg-acquisition/    # Low-level ADC and wireless
│   ├── signal-processing/  # Real-time filtering and feature extraction
│   └── puppet-control/     # Haptic/audio feedback drivers
├── software/
│   ├── pattern-recognition/ # ML models for eze/uche/motor
│   ├── coherence-engine/   # Coherence operator implementation
│   ├── actor-system/       # Polyglot gossip framework
│   └── parent-dashboard/   # React-based monitoring UI
├── gosilang/
│   ├── compiler/           # Error bubbling compiler
│   ├── runtime/            # Active state machine VM
│   └── stdlib/             # Standard library with coherence ops
└── docs/
    ├── clinical-protocols/  # RTP therapy guidelines
    ├── safety-testing/     # IEC 60601-1 test reports
    └── api-reference/      # Developer documentation
```

9.2 Community Contribution Model

- **License:** AGPL v3 (copyleft for medical safety)
- **Governance:** OBINexus steering committee (21 validators)
- **Contribution Process:**

1. Fork repository
2. Implement feature with tests
3. Submit pull request with safety analysis
4. Community review (minimum 3 approvals)
5. Integration testing in staging environment
6. Merge to main branch

9.3 Clinical Validation Framework

Partnership Model:

- Academic research institutions conduct independent validation
- Anonymized data shared via secure research portal
- Publications use open-access journals (preprints on arXiv)
- Replication encouraged through hardware/software reproducibility

---

10. Glossary

Active State Machine: State machine that autonomously triggers actions based on preserved context (memory + time + space).

Actor: Computational entity that processes messages and generates events in distributed system.

BCI (Brain-Computer Interface): System translating neural signals to external actions without peripheral nerves/muscles.

Bubbling (Error): Upward propagation of errors to parent calling context, opposite of downward propagation.

Coherence: Measure of semantic consistency between neural intent, puppet expression, and actor computation.

eze: Visual thought monitoring mode (visual cortex pattern detection).

Gosilang: Programming language with built-in error bubbling and active state machines for OBINexus.

Observer: Component that tracks and logs state transitions in event processing.

Panic: Catastrophic error level requiring immediate process termination.

Polyglot Gossip: Actor communication pattern across multiple programming languages with semantic preservation.

Puppet Protocol: Physical relay mechanism using tangible objects as identity anchors for neural state translation.

Semantic Drift: Gradual loss of meaning as context flows across system boundaries.

uche: Internal monologue monitoring mode (language processing area detection).

Watcher: Passive component observing events without modifying system state.

---

11. References

Academic Literature

- Wolpaw et al. (2002). "Brain-computer interfaces for communication and control."
- Nicolelis & Lebedev (2009). "Principles of neural ensemble physiology underlying the operation of brain-machine interfaces."
- Vansteensel et al. (2016). "Fully implanted brain-computer interface in a locked-in patient."

Regulatory Guidance

- FDA (2021). "Guidance for Industry: Implanted Brain-Computer Interface Devices."
- ISO 14971:2019. "Medical devices - Application of risk management."
- IEC 60601-1:2012. "Medical electrical equipment - General requirements for basic safety."

OBINexus Framework

- "Puppet Method Protocol: Sovereign Identity Communication Framework" (2025)
- "Gosilang Technical Specification: Error Bubbling and Active State" (2025)
- "Intentional Fragility as a Structural Principle" (2025)

Appendix A: Coherence Operator Mathematics

A.1 Formal Definition

Let **N** be the neural state space, **P** be the puppet action space, and **A** be the actor computation space.

Coherence Function:

```
C: (N × P × A) → [0, 1]

C(n, p, a) = w₁·sim(n, p) + w₂·sim(p, a) + w₃·sim(n, a)

where:
- sim(x, y) is semantic similarity function [0, 1]
- w₁, w₂, w₃ are weights (w₁ + w₂ + w₃ = 1)
- Default: w₁ = 0.5, w₂ = 0.3, w₃ = 0.2
```

Coherence Threshold:

```
Operational_Coherence = C(n, p, a) ≥ 0.954
```

A.2 Union Operator

```
(N ∪ P) = {x | x ∈ N OR x ∈ P}

Interpretation: Complete semantic space of neural+puppet system
```

A.3 Intersection Operator

```
(N ∩ P) = {x | x ∈ N AND x ∈ P}

Interpretation: Perfect semantic alignment moments
Coherence_Quality = |N ∩ P| / |N ∪ P|
```

Document Control

Version History:

- v1.0 (2025-10-26): Initial cleaned specification from transcript

Authors:

- OBINexus Technical Working Group
- Transcribed and formatted by Claude (Anthropic)

Review Status:

- ⚠️ Draft - Requires engineering review
- ⚠️ Pending: Coherence operator empirical validation
- ⚠️ Pending: Clinical trial protocol IRB approval

Next Steps:

1. Technical review by BCI engineering team
2. Safety analysis by medical device consultants
3. Pilot study design with autism therapy clinicians
4. Gosilang compiler prototype milestone planning

End of Technical Specification