RIFT-Bridge Governance Relay Interface

OBINexus Computing - AEGIS Project Implementation

Version: 1.0.0-dev
Stage: Implementation Gate
Classification: Git-RAF Enforced
Author: AEGIS Core Engineering Team
Date: June 22, 2025

Executive Summary

The RIFT-Bridge executable serves as the central governance relay interface, orchestrating policy enforcement across all compilation stages (.riftrc.0 through .rift.7) while maintaining cryptographic attestation chains and enabling secure inter-stage communication. This component implements the zero-trust governance architecture that validates every transformation within the RIFT ecosystem.

Architectural Overview

Core Responsibilities

- **Cross-Stage Policy Coordination:** Validates policy inheritance from .riftrc.0 → .rift.7
- **Git-RAF Attestation Management:** Maintains cryptographic integrity across all stages
- **Event-Driven Governance Hooks:** Implements real-time policy enforcement listeners
- **URI-Safe DSL Channeling:** Secures communication between compilation stages
- **Anti-Ghosting Protocol Enforcement:** Ensures no stage bypasses governance validation

Component Architecture

```
rift-bridge.exe
├── Policy Coordination Engine (PCE)
├── Attestation Management Layer (AML)
├── Event Listener Framework (ELF)
├── DSL Communication Channel (DCC)
└── Governance Validation Core (GVC)
```

Policy Coordination Engine (PCE)

Stage Inheritance Protocol

c

```c
typedef struct {
    uint8_t  stage_id;                  // Current stage identifier [0..7]
    uint32_t policy_version;            // Inherited policy version
    uint8_t  governance_hash[32];       // BLAKE3 hash of inherited policies
    uint8_t  validation_signature[64];  // Ed25519 stage validation signature
    uint64_t timestamp_utc;             // UTC timestamp for audit trail
} rift_stage_context_t;

// Policy inheritance validation chain
typedef struct {
    rift_stage_context_t stages[8];     // Complete stage context chain
    uint8_t chain_integrity_hash[32];   // Overall chain validation hash
    uint8_t final_attestation[64];      // Final Ed25519 attestation signature
} rift_policy_inheritance_chain_t;
```

Policy Validation Interface

c

```c
// Cross-stage policy validation
int rift_bridge_validate_policy_inheritance(
    const rift_policy_inheritance_chain_t* chain,
    uint8_t target_stage,
    const char* policy_requirement
);

// Policy enforcement checkpoint
typedef enum {
    RIFT_POLICY_PASS,           // Policy validation successful
    RIFT_POLICY_FAIL,           // Policy validation failed - halt compilation
    RIFT_POLICY_WARN,           // Policy warning - log and continue
    RIFT_POLICY_ESCALATE        // Policy escalation required - governance review
} rift_policy_result_t;
```

Attestation Management Layer (AML)

Git-RAF Integration Architecture

c

```c
typedef struct {
    char commit_hash[65];                // SHA-256 commit identifier
    uint8_t policy_tag_hash[32];         // BLAKE3 policy chain hash
    uint8_t entropy_checksum[16];        // ChaCha20-Poly1305 entropy validation
    uint8_t aura_seal[64];               // Ed25519 AuraSeal signature
    uint64_t hardware_binding;           // TPM-bound platform identifier
} git_raf_attestation_t;

// Git-RAF validation chain
typedef struct {
    git_raf_attestation_t attestations[8]; // One per compilation stage
    uint8_t chain_merkle_root[32];         // Merkle tree root of attestation chain
    uint8_t master_signature[64];          // Final attestation signature
} git_raf_chain_t;
```

Attestation Validation Functions

c

```c
// Validate Git-RAF attestation for specific stage
int rift_bridge_validate_raf_attestation(
    const git_raf_attestation_t* attestation,
    uint8_t stage_id,
    const uint8_t* stage_output_hash
);

// Generate new attestation for stage completion
int rift_bridge_generate_stage_attestation(
    uint8_t stage_id,
    const uint8_t* stage_input_hash,
    const uint8_t* stage_output_hash,
    git_raf_attestation_t* output_attestation
);
```

Event Listener Framework (ELF)

Governance Event Types

```c
typedef enum {
    RIFT_EVENT_STAGE_ENTRY,      // Stage beginning validation
    RIFT_EVENT_STAGE_EXIT,       // Stage completion validation
    RIFT_EVENT_POLICY_VIOLATION, // Policy enforcement failure
    RIFT_EVENT_ATTESTATION_FAIL, // Cryptographic validation failure
    RIFT_EVENT_GOVERNANCE_ESCALATION, // Manual governance review required
    RIFT_EVENT_EMERGENCY_HALT    // Emergency compilation termination
} rift_governance_event_t;

// Event listener callback interface
typedef struct {
    rift_governance_event_t event_type;
    void (*callback)(const void* event_data, size_t data_length);
    uint32_t priority_level;      // Event handling priority
    bool blocking;                // Whether event blocks compilation
} rift_event_listener_t;
```

Event Hook Registration

```c
// Register governance event listener
int rift_bridge_register_event_listener(
    rift_governance_event_t event_type,
    rift_event_listener_t* listener
);

// Trigger governance event
int rift_bridge_trigger_governance_event(
    rift_governance_event_t event_type,
    const void* event_data,
    size_t data_length
);
```

DSL Communication Channel (DCC)

URI-Safe Communication Protocol

c

```c
// URI-safe DSL message format
typedef struct {
    char protocol_version[8];     // Protocol version identifier
    char source_stage[4];         // Source stage identifier [0..7]
    char target_stage[4];         // Target stage identifier [0..7]
    uint32_t message_length;      // Message payload length
    uint8_t message_hash[32];     // BLAKE3 hash of message content
    uint8_t signature[64];        // Ed25519 message signature
} rift_dsl_header_t;

// DSL message payload structure
typedef struct {
    rift_dsl_header_t header;
    uint8_t* payload_data;        // Variable-length message payload
    uint8_t integrity_check[16];  // ChaCha20-Poly1305 integrity verification
} rift_dsl_message_t;
```

Secure Inter-Stage Communication

c

```c
// Send secure message between stages
int rift_bridge_send_dsl_message(
    uint8_t source_stage,
    uint8_t target_stage,
    const void* message_data,
    size_t message_length
);

// Receive and validate DSL message
int rift_bridge_receive_dsl_message(
    uint8_t expected_source_stage,
    rift_dsl_message_t* output_message
);
```

Governance Validation Core (GVC)

Zero-Trust Validation Engine

c

```c
// Zero-trust validation context
typedef struct {
    bool stage_validated[8];           // Per-stage validation status
    uint32_t validation_failures;     // Count of validation failures
    uint64_t last_validation_time;    // Timestamp of last validation
    uint8_t trust_level;               // Current trust assessment [0-100]
} rift_trust_context_t;

// Comprehensive governance validation
int rift_bridge_validate_governance_state(
    const rift_policy_inheritance_chain_t* policy_chain,
    const git_raf_chain_t* attestation_chain,
    rift_trust_context_t* trust_context
);
```

Anti-Ghosting Protocol Implementation

c

```c
// Anti-ghosting validation checkpoint
typedef struct {
    uint64_t stage_entry_time;         // Stage entry timestamp
    uint64_t max_stage_duration;       // Maximum allowed stage duration
    uint32_t heartbeat_interval;       // Required heartbeat frequency
    uint32_t missed_heartbeats;        // Count of missed heartbeats
} rift_anti_ghosting_context_t;

// Verify stage is actively processing (not ghosted)
int rift_bridge_verify_stage_liveness(
    uint8_t stage_id,
    rift_anti_ghosting_context_t* ghosting_context
);
```

Command Line Interface

Governance Operations

```bash
# Stage validation commands
rift-bridge validate-stage --stage=N --input-hash=<hash> --output-hash=<hash>
rift-bridge verify-policy-chain --chain-file=policy_chain.rift
rift-bridge check-attestation --raf-chain=attestation_chain.raf

# Event monitoring commands
rift-bridge monitor-events --stage=all --output=governance_log.json
rift-bridge trigger-event --type=POLICY_VIOLATION --data=violation_details.json

# DSL communication commands
rift-bridge send-message --from=2 --to=3 --message-file=stage_data.dsl
rift-bridge receive-message --from=2 --timeout=30s --output=received_data.dsl

# Anti-ghosting monitoring
rift-bridge monitor-liveness --stage=all --heartbeat-interval=5s
rift-bridge check-stage-health --stage=N --max-duration=300s
```

Configuration Management

```bash
# Governance configuration
rift-bridge configure --policy-enforcement=strict
rift-bridge configure --attestation-requirement=all-stages
rift-bridge configure --anti-ghosting-timeout=60s
rift-bridge configure --dsl-encryption=chacha20-poly1305

# Status and diagnostic commands
rift-bridge status --verbose
rift-bridge diagnose --stage=N --output=diagnostic_report.json
rift-bridge audit-trail --from-timestamp=<utc> --to-timestamp=<utc>
```

Integration with RIFT Ecosystem

Stage Integration Points

```c
// Integration with .riftrc.0 (Token Sanitization)
int rift_bridge_stage0_integration(
    const char* source_file,
    const char* sanitized_output
);

// Integration with .rift.1 (Namespace Validation)
int rift_bridge_stage1_integration(
    const char* namespace_config,
    const char* validation_output
);

// Integration with .rift.7 (Hardware Deployment)
int rift_bridge_stage7_integration(
    const char* deployment_target,
    const char* hardware_attestation
);
```

Build System Integration

```makefile
# RIFT-Bridge build integration
RIFT_BRIDGE = rift-bridge.exe

# Stage validation integration
%.stage-validated: %.stage-output
  $(RIFT_BRIDGE) validate-stage --stage=$* --input-hash=$< --output-hash=$@
  $(RIFT_BRIDGE) generate-attestation --stage=$* --output=$@.raf

# Policy enforcement integration
%.policy-validated: %.stage-validated
  $(RIFT_BRIDGE) verify-policy-chain --stage=$* --input=$<
  $(RIFT_BRIDGE) check-anti-ghosting --stage=$* --max-duration=300s
```

Error Handling and Recovery

Governance Failure Modes

c

```c
#define RIFT_BRIDGE_ERR_POLICY_VIOLATION    -1001
#define RIFT_BRIDGE_ERR_ATTESTATION_FAIL    -1002
#define RIFT_BRIDGE_ERR_DSL_CORRUPTION      -1003
#define RIFT_BRIDGE_ERR_STAGE_GHOSTED       -1004
#define RIFT_BRIDGE_ERR_TRUST_EXHAUSTED     -1005
#define RIFT_BRIDGE_ERR_GOVERNANCE_ESCALATION -1006
```

Recovery Protocols

c

```c
// Automatic recovery procedures
int rift_bridge_recover_from_policy_violation(
    uint8_t failed_stage,
    const char* recovery_procedure
);

// Manual governance escalation
int rift_bridge_escalate_to_governance(
    const char* escalation_reason,
    const void* context_data,
    size_t context_length
);
```

Security and Compliance

Cryptographic Requirements

- **Hash Function:** BLAKE3 for all content hashing
- **Symmetric Encryption:** ChaCha20-Poly1305 for message integrity
- **Asymmetric Signatures:** Ed25519 for all attestation signatures
- **Random Number Generation:** Hardware-backed entropy sources

Audit Trail Requirements

```c
// Comprehensive audit trail structure
typedef struct {
    uint64_t timestamp_utc;           // Event timestamp
    char event_type[32];              // Human-readable event type
    uint8_t stage_id;                 // Associated compilation stage
    char description[256];            // Event description
    uint8_t context_hash[32];         // Hash of associated context data
    uint8_t audit_signature[64];      // Ed25519 audit trail signature
} rift_audit_entry_t;
```

Testing and Validation

Unit Test Requirements

```bash
# Governance validation tests
test_policy_inheritance_validation
test_attestation_chain_verification
test_event_listener_framework
test_dsl_communication_security
test_anti_ghosting_detection

# Integration tests
test_full_stage_coordination
test_git_raf_integration
test_hardware_attestation_binding
test_emergency_halt_procedures
```

Performance Requirements

- **Stage Validation Latency:** < 100ms per stage
- **Attestation Generation:** < 500ms per attestation
- **DSL Message Processing:** < 50ms per message
- **Event Processing:** < 10ms per event

Implementation Status: ✅ RIFT-Bridge Architecture Documented
Next Target: Import Enforcement Guide and Gossip Language Governance Layer
AEGIS Gate Progress: Implementation Gate - Component Development Phase