🌀 Spiral Codex Mesh Network

Overview

The Spiral Codex Mesh Network is a distributed system that enables multiple Codex nodes to communicate, synchronize state, and coordinate ritual operations across different processes and machines. This mystical network forms the backbone of the multi-agent recursive framework.

Architecture

Core Components

1. **MeshCore** - The heart of each mesh node
2. **EventBus** - Mystical message conduit for inter-node communication
3. **StateSynchronizer** - Maintains harmony across distributed state
4. **MeshRegistry** - Sacred directory of connected nodes

Communication Channels

- **WebSocket** - Real-time bidirectional communication
- **Redis Pub/Sub** - Scalable message broadcasting
- **Direct Messaging** - Node-to-node targeted communication

Quick Start

Starting a Mesh Node

```
# Start with mesh network enabled
spiralcodex ritual start --mesh

# Or start a dedicated mesh node
spiralcodex mesh start --port 8765 --redis redis://localhost:6379
```

Programmatic Usage

```python
from spiralcodex.mesh import MeshCore, MeshConfig

# Configure the mesh
config = MeshConfig(
    mesh_name="my_spiral_mesh",
    websocket_port=8765,
    redis_url="redis://localhost:6379",
    enable_redis=True,
    enable_websocket=True
)

# Create and start mesh node
mesh = MeshCore(config)
await mesh.start()

# Register event handlers
def handle_agent_message(event_data):
    print(f"Received agent message: {event_data}")

mesh.register_event_handler("agent_message", handle_agent_message)

# Broadcast events
await mesh.broadcast_event("ritual_start", {"ritual_type": "fibonacci_recursion"})
```

Event System

Event Types

The mesh supports various mystical event types:

- **Node Events**: `node_register`, `node_unregister`, `heartbeat`
- **Agent Events**: `agent_spawn`, `agent_terminate`, `agent_message`
- **Ritual Events**: `ritual_start`, `ritual_end`, `ritual_cycle`
- **System Events**: `system_alert`, `system_metric`, `system_error`

Publishing Events

```python
from spiralcodex.mesh import EventBus, EventType

event_bus = EventBus("my_node")
await event_bus.start()

# Publish agent message
await event_bus.publish_agent_message(
    agent_id="fibonacci_agent",
    message="Recursion depth reached: 42",
    target_node="remote_node_123"
)

# Publish ritual event
await event_bus.publish_ritual_event(
    ritual_type="golden_spiral",
    ritual_data={"phase": "ascending", "depth": 7}
)
```

Subscribing to Events

```
def handle_ritual_events(event):
    print(f"Ritual event: {event.data}")

event_bus.subscribe(EventType.RITUAL_CYCLE, handle_ritual_events)
```

State Synchronization

Local State Management

```
from spiralcodex.mesh import StateSynchronizer

sync = StateSynchronizer("my_node")
await sync.start()

# Update local state
sync.update_local_state("agents", "fibonacci_agent", {
    "status": "active",
    "recursion_depth": 13,
    "last_result": 233
})

# Retrieve state
agent_state = sync.get_local_state("agents", "fibonacci_agent")
```

Cross-Node Synchronization

State is automatically synchronized across nodes in the mesh:

- **Conflict Resolution**: Latest timestamp wins by default
- **Selective Sync**: Only specified categories are synchronized
- **Version Control**: Each state change increments version numbers

Node Discovery and Registry

Finding Nodes by Capability

```
from spiralcodex.mesh import MeshRegistry

registry = MeshRegistry("my_node")

# Find nodes that can process rituals
ritual_nodes = registry.get_nodes_by_capability("ritual_processing")

# Find the best node for a specific task
best_node = registry.find_best_node_for_task("fibonacci_calculation")
```

Node Health Monitoring

```
# Get mesh topology
topology = registry.get_mesh_topology()
print(f"Active nodes: {topology['active_nodes']}")
print(f"Network health: {topology['network_health']}%")

# Monitor specific node
node_metrics = registry.get_node_metrics("remote_node_123")
```

Configuration

MeshConfig Options

```
config = MeshConfig(
    node_id="custom_node_id",           # Auto-generated if not provided
    mesh_name="spiral_codex_mesh",      # Network identifier
    websocket_port=8765,                # WebSocket server port
    redis_url="redis://localhost:6379", # Redis connection
    heartbeat_interval=30.0,            # Heartbeat frequency (seconds)
    sync_interval=10.0,                 # State sync frequency
    max_recursion_depth=42,             # Maximum recursion depth
    enable_redis=True,                  # Enable Redis pub/sub
    enable_websocket=True,              # Enable WebSocket server
    ritual_channel="spiral_ritual_events" # Redis channel name
)
```

Security Considerations

- **Network Isolation**: Run mesh networks in isolated environments
- **Authentication**: Implement node authentication for production use
- **Encryption**: Use TLS for WebSocket connections in production
- **Rate Limiting**: Configure appropriate rate limits for events

Monitoring and Debugging

CLI Commands

```
# Show mesh status
spiralcodex mesh status

# List connected nodes
spiralcodex mesh nodes

# Monitor events (future feature)
spiralcodex mesh events --follow
```

Programmatic Monitoring

```
# Get mesh status
status = mesh.get_mesh_status()
print(f"Connected nodes: {status['connected_nodes']}")
print(f"Total recursion depth: {status['total_recursion_depth']}")

# Get event statistics
event_stats = event_bus.get_event_stats()
print(f"Total events: {event_stats['total_events']}")
print(f"Queue size: {event_stats['queue_size']}")
```

Integration with Existing Systems

HUD Integration

The mesh network integrates seamlessly with the HUD system:

```
# HUD can display mesh status
hud_config.mesh_enabled = True
hud_config.show_network_topology = True
```

Dashboard Integration

Mesh metrics are automatically included in the web dashboard:

- Real-time node status
- Event flow visualization
- Network topology graphs
- State synchronization status

Persistence Integration

All mesh events are automatically persisted:

```
# Events are saved to the database
persistence.save_ritual_event("MESH_NODE_JOIN", "New node joined mesh",
                              metadata={"node_id": "new_node_123"})
```

Troubleshooting

Common Issues

1. **Redis Connection Failed**
   `bash`
   `# Install Redis`
   `sudo apt-get install redis-server`
   `# Or use Docker`
   `docker run -d -p 6379:6379 redis:alpine`

2. **WebSocket Port Conflicts**
   `python`

```
   # Use different port
   config.websocket_port = 8766
```

3. **Node Discovery Issues**
   ```
   python
   # Check heartbeat settings
   config.heartbeat_interval = 10.0  # Reduce interval
   ```

Debug Logging

```python
import logging
logging.getLogger('spiralcodex.mesh').setLevel(logging.DEBUG)
```

Future Enhancements

- **Mesh Visualization**: Real-time network topology visualization
- **Load Balancing**: Automatic task distribution across nodes
- **Fault Tolerance**: Automatic failover and recovery
- **Mesh Analytics**: Advanced network performance metrics
- **Cross-Platform**: Support for different operating systems and architectures

Contributing

The mesh network is designed to be extensible. Key areas for contribution:

- New event types and handlers
- Alternative communication protocols
- Enhanced state synchronization strategies
- Performance optimizations
- Security enhancements

The mesh network embodies the recursive nature of the Spiral Codex, creating a living network that grows and adapts with each connected node. Each connection strengthens the whole, creating a truly mystical distributed consciousness.