ORSA Architecture Summary

Oxy Reservoir Simulation Agent

Comprehensive System Architecture Document

Document Information	
Version	1.0
Date	December 28, 2025
Author	Software Architecture Team
Status	Draft - For Review

Table of Contents

1. Executive Summary

The Oxy Reservoir Simulation Agent (ORSA) represents a transformational approach to reservoir modeling at Oxy. By leveraging advanced AI and natural language processing, ORSA will evolve from an expert ECLIPSE coding assistant to a comprehensive reservoir engineering intelligence platform, democratizing simulation capabilities while preserving and amplifying institutional expertise.

Key Architectural Highlights

Aspect	Approach	Benefit
Configuration	Every component configurable	Deployment flexibility across environments
Data Layer	Domain-driven data mesh	Optimized storage for each data type
AI Integration	Unified model abstraction	Swap models without code changes
Communication	Hybrid sync/async patterns	Responsive UI with long-running jobs
NLP Pipeline	Multi-stage translation	Speech directly to ECLIPSE deck syntax

2. System Overview

Vision and Objectives

ORSA enables both technical SMEs and business users to interact with complex simulation capabilities through conversational interfaces. The system translates natural language requests directly into ECLIPSE deck syntax, bridging the gap between user intent and technical execution.

Target User Groups

User Type	Needs	ORSA Solution
SMEs / Reservoir Engineers	Fast debugging, code generation, validation	Technical mode with full ECLIPSE control
Business Teams	Scenario analysis without technical expertise	Natural language queries, spreadsheet uploads
Asset Engineers	History matching, optimization guidance	Physics-informed recommendations
International Teams	Air-gapped, compliant solutions	Local deployment with full capabilities

High-Level Architecture Layers

Layer	Purpose	Key Technologies
User Interface	Voice/text natural language input	Speech-to-text, Web interfaces, API
NLP Translation	Convert speech to ECLIPSE deck	Transformer models, Entity extraction
API Gateway	Unified access, security, routing	Kong/Traefik, Authentication
Orchestration	Async job management	Workflow engines, Message brokers
AI Inference	Configurable model serving	Multiple LLM backends, Model registry
Data Mesh	Domain-specific storage	Cassandra, MongoDB, PostgreSQL, Neo4j, Redis

3. Core Architectural Principles

3.1 Configuration Over Convention

Every component in the ORSA architecture is designed to be configurable rather than hard-coded. This is the primary architectural philosophy that enables deployment across diverse environments.

Component	Configurable Aspects	Example Configurations
Database Layer	Backend selection, connection pools	Cassandra vs TimescaleDB, PostgreSQL vs MySQL
Message Broker	Broker type, topics, partitions	Kafka for production, Redis Streams for air-gapped
AI Models	Provider, model version, endpoints	Local CodeLlama, Cloud API, Custom fine-tuned
Model Serving	Infrastructure, scaling parameters	Ray Serve, Triton, lightweight custom
Security	Auth providers, encryption settings	LDAP, OAuth, certificate management

3.2 Microservices Architecture

The system employs a microservices approach with domain-driven design:

- Independent scaling of different system components based on load
- Technology diversity allowing optimal tools for each service
- Fault isolation preventing cascade failures across the system
- Simplified testing and deployment with smaller, focused services
- Team autonomy enabling parallel development streams

3.3 Event-Driven Communication

Asynchronous messaging ensures system responsiveness and scalability:

- Non-blocking user interactions for responsive frontend experience
- Long-running simulation job management without timeout issues
- Real-time status updates through WebSocket connections
- Cross-service data consistency via event sourcing patterns
- Decoupled services enabling independent evolution

4. Data Architecture

4.1 Data Mesh Approach

The ORSA data architecture implements a domain-driven data mesh with specialized storage optimized for different data types and access patterns. Each domain owns its data, schema, and access patterns.

4.2 Data Domains

Domain	Technology	Data Types	Access Pattern	Rationale
Simulation	Cassandra	Time-series, ECLIPSE decks	Write-heavy, sequential reads	Horizontal scaling, high write throughput
Training	MongoDB	ML datasets, model artifacts	Document-oriented, flexible	Schema flexibility for varied ML data
User	PostgreSQL	Auth, permissions, audit	Transactional, relational	ACID compliance, data integrity
Connectivity	Neo4j	Well relationships, flows	Graph traversal, patterns	Natural fit for relationship analysis
Cache	Redis	Sessions, frequent queries	Key-value, high-speed	Sub-millisecond response times

4.3 Data Flow and Integration

Event-driven data synchronization ensures consistency across domains:

- Domain events published to Kafka topics for cross-domain communication
- ETL pipelines for batch data translation between storage systems
- Data contracts defining strict interface specifications between domains
- Eventual consistency with well-defined conflict resolution protocols

4.4 Configurable Backend Strategy

The data layer supports multiple backend configurations through abstraction:

Deployment Scenario	Recommended Stack	Alternatives
Cloud Production	Managed Cassandra, Atlas MongoDB, RDS PostgreSQL	Self-hosted equivalents
Air-gapped International	Self-hosted Cassandra, MongoDB, PostgreSQL	SQLite for smaller deployments
Development/Testing	Docker Compose with all databases	In-memory alternatives
Edge/Lightweight	Redis + SQLite	Embedded databases

5. AI Layer Design

5.1 Configurable Model Architecture

The AI layer implements a flexible approach supporting multiple model providers and deployment scenarios through a unified abstraction layer that normalizes different providers into a consistent interface.

5.2 Model Abstraction Layer

Model Type	Examples	Use Case	Deployment
Local Fine-tuned	CodeLlama 34B, DeepSeek-Coder 33B	ECLIPSE debugging, code generation	Air-gapped, on-premise GPU
Cloud APIs	OpenAI GPT-4, Anthropic Claude	General queries, fallback	Internet-connected environments
Custom Domain	ORSA-trained models	Reservoir engineering specific	Any environment
Lightweight	Distilled 7B-13B models	Fast inference, limited resources	Edge, resource-constrained

5.3 AI Integration Points

Integration Point	AI Application	Technology	Purpose
NLP Translation	Speech-to-ECLIPSE	Transformer models	Convert natural language to simulation syntax
AI Inference	RE Intelligence	Domain-specific LLMs	Technical guidance and optimization
Model Router	Request classification	Lightweight classifier	Route requests to appropriate models
Job Scheduling	Resource optimization	ML optimization	Intelligent workload management

5.4 Model Serving Infrastructure

Option	Best For	Pros	Cons
Ray Serve	High-performance, multi-model	Flexible, Python-native	Complex setup
Triton	Production GPU inference	Optimized, multi-framework	Steep learning curve
vLLM	LLM-specific serving	Fast LLM inference	LLM-only
Custom Lightweight	Air-gapped, simple	Full control, minimal deps	More development effort

6. NLP Translation Pipeline

6.1 Core Innovation

The core innovation of ORSA is its ability to translate natural language directly into ECLIPSE deck syntax through a sophisticated multi-stage NLP pipeline. This enables users to speak naturally and have the system generate valid simulation input files.

6.2 Pipeline Stages

Stage	Technology	Input	Output	Validation
1. Speech-to-Text	Configurable ASR	Audio stream	Raw transcription	Confidence score
2. Intent Recognition	Transformer model	Transcription	Structured intent	Intent confidence
3. Entity Extraction	NER models	Intent + text	Named entities	Entity validation
4. Asset Validation	Database lookup	Entities	Validated entities	Asset existence
5. Syntax Generation	Code gen model	Validated entities	ECLIPSE syntax	Syntax check
6. Deck Validation	ECLIPSE parser	Generated deck	Final deck	Physics validation

6.3 Error Handling and Rollback

Each stage includes comprehensive error handling:

- Validation checkpoints at every stage transition
- Rollback capabilities to previous valid state
- User feedback loops for clarification when confidence is low
- Logging of all transformations for debugging and improvement

7. Communication Architecture

7.1 Hybrid Synchronous/Asynchronous Design

The communication layer bridges the gap between synchronous frontend expectations and asynchronous backend processing requirements.

7.2 API Gateway Layer

Endpoint Type	Method	Purpose	Response
Authentication	POST /auth/login	User authentication	Sync - JWT token
Quick Query	GET /query	Simple lookups	Sync - immediate response
Job Submit	POST /jobs	Submit simulation	Sync - job ID returned
Job Status	GET /jobs/{id}/status	Check progress	Sync - current status
WebSocket	WS /realtime	Live updates	Async - push notifications

7.3 Orchestration Layer

A workflow orchestration engine manages complex async operations:

- Job lifecycle management: queued → running → completed/failed
- Dependency handling for multi-step workflows
- Resource allocation and optimization across jobs
- Result aggregation and notification delivery
- Retry logic with exponential backoff for transient failures

7.4 Message Broker Options

Scenario	Recommended Broker	Configuration	Rationale
High-throughput production	Apache Kafka	Multi-broker cluster	Durability, scalability, event streaming
Standard deployments	RabbitMQ	HA cluster	Ease of setup, reliable delivery
Lightweight/air-gapped	Redis Streams	Single instance + replica	Minimal overhead, in-memory speed
Development/testing	In-memory queues	Embedded	Simplicity, no external deps

8. Risk Assessment and Mitigation

8.1 Identified Architecture Risks

Risk	Description	Impact	Probability	Mitigation
NLP Complexity	Speech-to-ECLIPSE accuracy	High	Medium	Multi-stage validation pipeline
Model Sync	AI model consistency	Medium	Medium	Centralized registry + health checks
Distributed Debug	Async workflow issues	High	Medium	Comprehensive tracing + logging
Data Fragmentation	Multi-DB inconsistency	Medium	Low	Event-driven sync + contracts
Performance	System scalability	High	Low	Horizontal scaling + caching

8.2 Detailed Mitigation Strategies

NLP Translation Pipeline Robustness:

- Multi-stage validation with rollback capabilities at each step
- Entity extraction validation against asset databases
- ECLIPSE syntax validation before simulation submission
- Human-in-the-loop validation for complex or ambiguous scenarios
- Continuous model improvement from user feedback

Model Synchronization Strategy:

- Service mesh with health checks and circuit breakers
- Automated failover between model providers
- Model registry tracking versions and performance metrics
- Graceful degradation when primary models unavailable

Distributed System Observability:

- Distributed tracing across all service boundaries (Jaeger/Zipkin)
- Centralized logging with correlation IDs (ELK Stack)
- Performance monitoring and alerting (Prometheus/Grafana)
- Circuit breakers to prevent cascade failures (Istio)

9. Technology Stack

9.1 Core Infrastructure

Layer	Primary Choice	Alternatives	Selection Rationale
Container Orchestration	Kubernetes	Docker Swarm, Nomad	Industry standard, ecosystem
API Gateway	Kong / Traefik	Istio Gateway, AWS ALB	Open source, feature-rich
Service Mesh	Istio	Linkerd, Consul Connect	Traffic management, security
Secrets Management	HashiCorp Vault	K8s Secrets, AWS SM	Centralized, auditable

9.2 Data and Messaging

Purpose	Primary Choice	Alternatives	Selection Rationale
Time-series DB	Apache Cassandra	InfluxDB, TimescaleDB	Write scalability, distributed
Document Store	MongoDB	CouchDB, DocumentDB	Schema flexibility, queries
Relational DB	PostgreSQL	MySQL, Oracle	ACID, feature richness
Graph Database	Neo4j	ArangoDB, Neptune	Cypher, performance
Cache	Redis	Memcached, Hazelcast	Data structures, persistence
Message Broker	Apache Kafka	RabbitMQ, Redis Streams	Scalability, event streaming
Workflow Engine	Temporal	Argo, Airflow	Reliability, failure handling

9.3 Observability Stack

Function	Tool	Purpose
Metrics	Prometheus	Time-series metrics collection and storage
Visualization	Grafana	Dashboards and alerting
Logging	ELK Stack	Log aggregation, search, and analysis
Tracing	Jaeger	Distributed request tracing
APM	Custom / DataDog	Application performance monitoring

10. Deployment Architecture

10.1 Multi-Environment Support

Environment	Characteristics	Technology Adaptations
Air-gapped International	No external connectivity, compliance	Local models, lightweight messaging, offline updates
Cloud Production	High availability, auto-scaling	Managed services, cloud-native databases
Hybrid Development	Mixed connectivity, testing	Configurable backends, staging pipelines
Edge Deployment	Limited resources, field ops	Optimized models, minimal infrastructure

10.2 Security and Compliance

Enterprise-grade security across all layers:

- Zero-trust network architecture with mutual TLS
- End-to-end encryption for data in transit and at rest
- Role-based access control (RBAC) with fine-grained permissions
- Comprehensive audit logging for compliance requirements
- Regular security scanning and vulnerability assessments

11. Implementation Roadmap

Phase	Duration	Focus Areas	Key Deliverables	Success Criteria
Phase 0: Foundation	3-4 months	Core infrastructure, basic microservices	K8s cluster, CI/CD, base services	Services communicating
Phase 1: Basic NLP	4-6 months	Speech-to-text, simple ECLIPSE gen	Working NLP pipeline, basic UI	Simple queries working
Phase 2: AI Integration	6-8 months	Model serving, intelligent routing	Multi-model support, router	Model switching works
Phase 3: Production	4-6 months	Observability, optimization, scaling	Full monitoring, HA deployment	Production-ready
Phase 4: Advanced	6+ months	Graph analytics, MORSA/FORSA	Full field integration	Complete ORSA vision

12. Conclusions

Strategic Value

This architecture represents a strategic transformation of reservoir engineering capabilities. By democratizing access to simulation tools while maintaining technical rigor, ORSA positions Oxy at the forefront of AI-driven energy operations.

The configurable, microservices-based approach ensures that this investment will adapt and grow with advancing AI capabilities, providing sustained competitive advantage through technological leadership.

Critical Success Factors

- NLP Translation Accuracy: The core value proposition depends on reliable speech-to-ECLIPSE conversion
- Comprehensive Observability: Distributed system complexity requires thorough monitoring
- Gradual Rollout: Phased deployment allows validation and refinement at each layer
- Change Management: User adoption requires attention to workflow integration

Architecture Diagram