Orleans.StateMachineES

Production-Ready Distributed State Machines for Microsoft Orleans

Event Sourcing • Compile-Time Safety • Enterprise Resilience

.NET 9 • Orleans 9.1.2 • Stateless 5.17.0 • OpenTelemetry

Executive Overview

Version 1.1.0

A comprehensive framework for building distributed state machines
with full event sourcing, hierarchical states, and enterprise-grade reliability.

Built with .NET & Orleans • 3 NuGet Packages • 10 Roslyn Analyzers

Open-source (MIT License) • Published on NuGet.org

Contact: michael.ivertowski@ch.ey.com

Contents

1 Executive Summary

Orleans.StateMachineES is a production-ready .NET library that integrates the Stateless state machine library with Microsoft Orleans' actor framework. It enables distributed state machine patterns in Orleans applications with full event sourcing, hierarchical states, and enterprise-grade reliability features. The library provides compile-time safety through 10 Roslyn analyzers and achieves significant performance optimizations through intelligent caching.

Why Orleans.StateMachineES?

- **Event Sourcing** — Complete state history with 30%+ performance improvement through auto-confirmed events and snapshot support.
- **Compile-Time Safety** — 10 Roslyn analyzers detect async anti-patterns, missing states, and design issues before runtime.
- **Enterprise Resilience** — Circuit breakers, rate limiting, retry components, and distributed saga support with automatic compensation.
- **Production-Ready** — OpenTelemetry tracing, health checks, state visualization, and comprehensive monitoring integration.
- **Performance Optimized** — TriggerParameterCache delivers ∼100× improvement for parameterized triggers with zero-allocation paths.

1.1 At a Glance

5,923 transitions/sec (event sourced)	**100×** trigger cache speedup	**30%+** event sourcing gain
10 Roslyn analyzers	**3** NuGet packages	**4** example applications

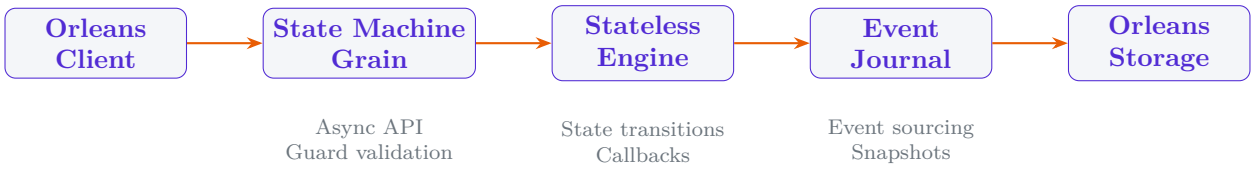
2 Core Abstractions

2.1 State Machine Grain Hierarchy

Orleans.StateMachineES provides a layered grain hierarchy for different use cases, from simple state machines to complex event-sourced workflows with hierarchical states and distributed sagas.

Abstraction	Description
StateMachineGrain	Base grain wrapping Stateless with Orleans-compatible async methods, trigger parameter caching, and comprehensive state inspection APIs.
EventSourcedStateMachineGrain	Extends StateMachineGrain with JournaledGrain-based event sourcing, automatic snapshots, idempotency, and Orleans Streams integration.
TimerEnabledStateMachineGrain	Adds state-driven timeouts using Orleans Timers (short) and Reminders (durable), with repeating heartbeat patterns.
HierarchicalStateMachineGrain	Supports nested states with parent-child relationships, state path queries, and inheritance-based behavior.
VersionedStateMachineGrain	Semantic versioning, migration strategies (automatic, blue-green), shadow evaluation, and rollback support.
OrthogonalStateMachineGrain	Parallel independent state machines (regions) with cross-region synchronization and trigger mapping.

2.2 State Machine Flow



3 Compile-Time Safety

3.1 Roslyn Analyzer Suite

Orleans.StateMachineES includes 10 comprehensive Roslyn analyzers that detect anti-patterns at compile time, preventing runtime issues before deployment.

ID	Severity	Description
OSMES001	Warning	Async lambda in state callback (OnEntry, OnExit)
OSMES002	Error	FireAsync called within state callback
OSMES003	Warning	Missing initial state configuration
OSMES004	Warning	Unreachable state detected
OSMES005	Info	Multiple entry callbacks on same state
OSMES006	Warning	State has no trigger handlers
OSMES007	Warning	Circular state transitions with no exit path
OSMES008	Warning	Guard condition complexity too high (> 10)
OSMES009	Error	State machine missing initial state assignment
OSMES010	Warning	Invalid enum value cast (unsafe numeric cast)

3.2 Async Safety Protection

The underlying Stateless library does **not** support async operations in state callbacks. Orleans.StateMachineES provides both compile-time and runtime protection:

Protection Layers

- **OSMES001** — Warns when async lambdas are used in OnEntry/OnExit
- **OSMES002** — Errors when FireAsync is called within callbacks
- **Runtime Validation** — Thread-local tracking prevents nested FireAsync
- **Clear Error Messages** — Actionable guidance for correct patterns

4 Key Features

4.1 Event Sourcing

Event sourcing provides complete audit trails with automatic state replay:

Feature	Details
Auto-Confirmed Events	30%+ performance improvement with `AutoConfirmEvents = true`, achieving 5,923 transitions/sec (0.17ms latency).
Snapshots	Configurable snapshot intervals for large event histories, with automatic state reconstruction on grain activation.
Idempotency	Built-in deduplication with LRU cache prevents duplicate command processing in distributed scenarios.
Stream Integration	Publish state transitions to Orleans Streams for real-time event processing and external system notifications.
Correlation Tracking	Track related events across distributed systems with correlation IDs propagated through sagas and workflows.

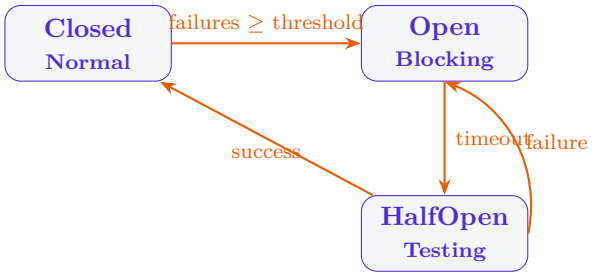
4.2 Production Enhancements (v1.1.0)

Component	Capabilities
Rate Limiting	Token bucket algorithm with configurable burst capacity, sliding/fixed windows, and real-time utilization statistics.
Batch Operations	Parallel execution API with configurable parallelism, exponential backoff retry, and progress tracking.
Event Schema Evolution	Automatic event upcasting with BFS-based version chain discovery for multi-step event upgrades.
Persistence Abstraction	Provider-agnostic IEventStore, ISnapshotStore, IStateMachinePersistence with in-memory and provider options (CosmosDB, PostgreSQL, MongoDB).
State Machine Templates	Reusable patterns: ApprovalWorkflow, OrderProcessing, RetryableOperation with configurable parameters.
State History Queries	Fluent LINQ-style API for temporal queries with GroupByState, GroupByTrigger, GroupByTime analytics.

5 Enterprise Resilience

5.1 Circuit Breaker Pattern

Production-ready resilience component with three-state management:



Configuration	Purpose
FailureThreshold	Number of consecutive failures before opening (default: 5)
SuccessThreshold	Successes needed in HalfOpen to close (default: 2)
OpenDuration	Time before testing recovery (default: 30s)
MonitoredTriggers	Specific triggers to protect with circuit breaker
ThrowWhenOpen	Throw exception or return false when circuit is open

5.2 Distributed Sagas

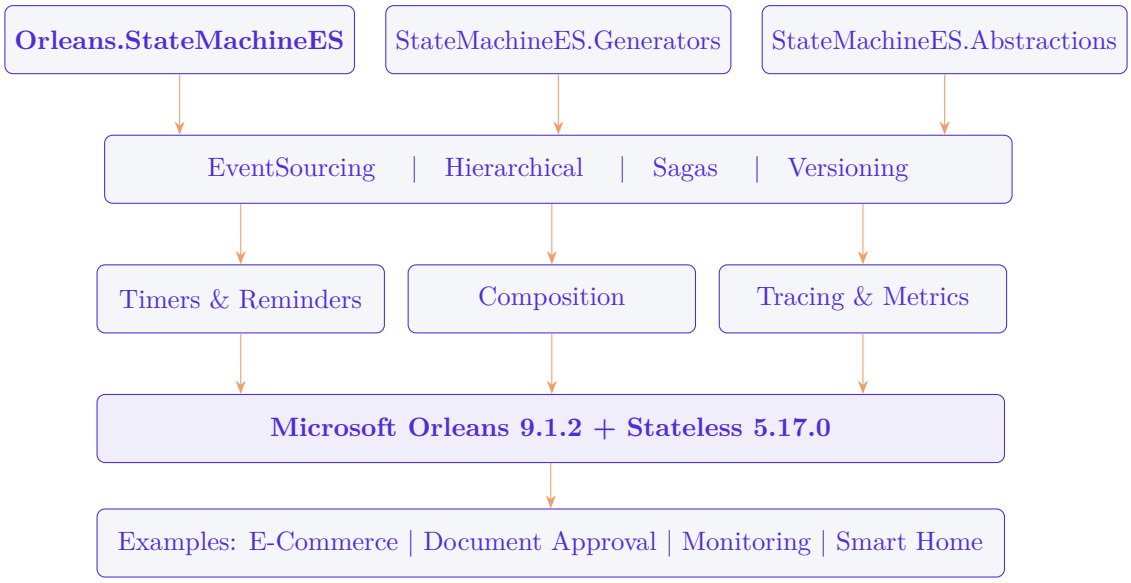
Orchestration-based multi-grain workflows with automatic compensation:

Saga Features

- **Step Orchestration** — Central saga coordinator manages business process
- **Automatic Compensation** — Failed steps trigger rollback in reverse order
- **Retry Logic** — Exponential backoff for technical failures
- **Business vs Technical Errors** — Different handling strategies
- **Timeout Management** — Per-step timeouts with graceful degradation
- **Full Audit Trail** — Event-sourced saga execution history

6 Architecture

6.1 Project Structure



6.2 NuGet Packages

Package	Version	Contents
Orleans.StateMachineES	1.1.0	Main library with all grain implementations, event sourcing, sagas, versioning, visualization, and enterprise components.
Orleans.StateMachineES.Abstractions	1.1.0	Core interfaces (IStateMachineGrain), models, and shared abstractions for extending the library.
Orleans.StateMachineES.Generators	1.1.0	Roslyn source generator for YAML/JSON specifications and 10 compile-time analyzers with full XML documentation.

7 Observability & Monitoring

7.1 OpenTelemetry Integration

Built-in distributed tracing and metrics for production visibility:

Capability	Details
Activity Tracing	Automatic spans for state transitions with grain type, ID, states, and trigger information. Supports Jaeger, Zipkin, OTLP exporters.
Metrics Collection	Counters and histograms: transitions_total, transition_duration, active_grains, trigger_errors_total, saga_executions_total.
TracingHelper	Utility class for custom operation tracing with correlation context and saga step tracking.
Health Checks	ASP.NET Core health check integration with custom providers for state machine grain health monitoring.

7.2 State Machine Visualization

Feature	Capabilities
Export Formats	DOT (Graphviz), Mermaid (GitHub markdown), PlantUML, JSON, XML, and interactive HTML with real-time updates.
Structural Analysis	Cyclomatic complexity, state count, max depth, connectivity index, unreachable state detection, and design recommendations.
Batch Visualization	Analyze multiple grain types simultaneously with comparative statistics and generated reports.
Web Dashboard	Interactive HTML visualization with Vis.js, live state tracking, transition history, and performance metrics.

8 Use Cases

Primary Use Cases

- **E-Commerce Workflows** — Order processing with payment timeouts, shipping state tracking, and complete audit trails through event sourcing.
- **Document Approval** — Multi-level approval workflows with hierarchical states, parallel review processes, and automatic escalation.
- **IoT Device Control** — Smart home automation with orthogonal regions for independent subsystems (security, climate, energy) and cross-region sync.
- **Financial Processing** — Invoice processing sagas with automatic compensation, audit compliance, and distributed transaction coordination.
- **Microservice Orchestration** — Long-running business processes with durable reminders, version migration, and graceful degradation.
- **Compliance & Audit** — Complete state history with temporal queries, event replay for debugging, and regulatory reporting.

9 Getting Started

9.1 Quick Start (NuGet)

```
# Add packages via CLI
dotnet add package Orleans.StateMachineES
dotnet add package Orleans.StateMachineES.Generators

# Or in .csproj
<PackageReference Include="Orleans.StateMachineES" Version="1.1.0" />
<PackageReference Include="Orleans.StateMachineES.Generators" Version="1.1.0" />
```

9.2 Basic State Machine Grain

```
public class DoorGrain :  StateMachineGrain<DoorState, DoorTrigger>, IDoorGrain
{
protected override StateMachine<DoorState, DoorTrigger> BuildStateMachine()
{
var machine = new StateMachine<DoorState, DoorTrigger>(DoorState.Closed);

machine.Configure(DoorState.Closed)
.Permit(DoorTrigger.Open, DoorState.Open)
.Permit(DoorTrigger.Lock, DoorState.Locked);

machine.Configure(DoorState.Open)
.Permit(DoorTrigger.Close, DoorState.Closed);

return machine;
}
}
```

9.3 Event-Sourced State Machine

```
public class OrderGrain :  EventSourcedStateMachineGrain<OrderState, OrderTrigger,
OrderState>
{
protected override void ConfigureEventSourcing(EventSourcingOptions options)
{
options.AutoConfirmEvents = true; // Critical for performance!
options.EnableSnapshots = true;
options.SnapshotInterval = 100;
}
}
```

Links & Resources

Repository	https://github.com/mivertowski/Orleans.StateMachineES
Documentation	https://mivertowski.github.io/Orleans.StateMachineES/
NuGet	https://www.nuget.org/packages/Orleans.StateMachineES/
API Reference	https://mivertowski.github.io/Orleans.StateMachineES/api/
Examples	See `examples/` directory for E-Commerce, Document Approval, Monitoring Dashboard, and Smart Home applications

Author: Michael Ivertowski • **Contact:** michael.ivertowski@ch.ey.com