MODAPTO [101091996]: Modular Manufacturing and Distributed Control via Interoperable Digital Twins

12.2.2 Managing service lifecycle and version control through registration process

2025-08-11

Service lifecycle management and version control are critical aspects of maintaining a healthy and reliable service ecosystem within MODAPTO. The registration process incorporates sophisticated mechanisms to track service evolution, manage dependencies, and ensure smooth transitions as services evolve over their operational lifetime.

Understanding Service Lifecycle States within the MODAPTO Service Catalogue encompasses multiple phases that reflect the operational status and availability of services. The Service Lifecycle Management (SLM) Framework is a way of managing the entire service from concept to retirement. It is a structured approach that ensures that all aspects concerning service delivery, performance, and improvement are dealt with in the best way possible. It outlines the service lifecycle management process and leads to improved customer satisfaction and business outcomes.

In MODAPTO, services progress through distinct states: Draft (under development, not publicly available), Published (active and available for consumption), Deprecated (marked for future removal but still operational), and Archived (removed from active use but retained for historical reference).

Version Control Implementation follows semantic versioning principles adapted for service-oriented architectures. Once you identify your public API, you communicate changes to it with specific increments to your version number. Consider a version format of X.Y.Z (Major.Minor.Patch). Bug fixes not affecting the API increment the patch version, backward compatible API additions/changes increment the minor version, and backward incompatible API changes increment the major version. We call this system “Semantic Versioning.” Under this scheme, version numbers and the way they change convey meaning about the underlying code and what has been modified from one version to the next.

Each service manifest includes version metadata that tracks the evolution of service capabilities, interfaces, and requirements over time.

Managing Breaking Changes represents one of the most critical aspects of service lifecycle management. Breaking changes include modifications to input/output structures, removal of endpoints, changes in authentication mechanisms, or alterations to expected behavior. Deprecating existing functionality is a normal part of software development and is often required to make forward progress. When you deprecate part of your public API, you should do two things: (1) update your documentation to let users know about the change, (2) issue a new minor release with the deprecation in place. Before you completely remove the functionality in a new major release there should be at least one minor release that contains the deprecation so that users can smoothly transition to the new API.

The MODAPTO registration process enforces validation checks to identify potential breaking changes and requires explicit acknowledgment before allowing publication.

Version Transition Strategies enable smooth migration paths for service consumers. When registering a new version, the system supports multiple approaches: Parallel deployment allows both old and new versions to run simultaneously during transition periods. Blue-green

deployment strategies enable instant switching between versions with rollback capabilities. Canary releases gradually shift traffic from old to new versions while monitoring for issues. The registration interface provides configuration options for selecting appropriate transition strategies based on service criticality and consumer impact.

Dependency Management Across Versions becomes increasingly important as services evolve. The Service Catalogue tracks inter-service dependencies, alerting administrators when a service update might impact dependent services. Microservices versioning is critical to managing different renditions of the same service and improving backward compatibility. However, there are other best practices to ensure backward compatibility. But why bother so much about backward compatibility? Backward compatibility for different services ensures that the system does not break if there are changes. In other words, you need to plan to evolve all the services together without breaking the system.

For Internal and Orchestrated services, the system can automatically verify compatibility between versions before allowing updates to proceed.

Lifecycle Event Notifications keep stakeholders informed about service changes throughout the lifecycle. The registration system integrates with the MODAPTO Message Bus to broadcast notifications about version releases, deprecation warnings, and service retirements. Subscribers can configure notification preferences based on service categories, severity levels, or specific services of interest. This proactive communication ensures that service consumers have adequate time to adapt to changes.

Version History and Rollback Capabilities provide safety nets for service evolution. Every service registration creates an immutable version record in the Catalogue database, preserving the complete manifest specification at that point in time. This historical record enables forensic analysis of service evolution, comparison between versions, and rapid rollback when issues arise. Version control is critical in microservices environments because some changes might require coordination among multiple microservices where API structures or content definitions change.

The registration interface includes tools for viewing version history, comparing manifests, and initiating rollback procedures.

Automated Lifecycle Policies streamline service management by implementing configurable rules for common lifecycle transitions. Administrators can define policies such as automatic deprecation after specified periods of inactivity, mandatory security updates for services using outdated dependencies, or automated archival of services that haven't been accessed within defined timeframes. These policies reduce manual overhead while ensuring consistent lifecycle management across the service Catalogue.

Performance Impact Tracking monitors how different service versions affect system performance and resource utilization. During registration of new versions, administrators can specify expected performance characteristics and resource requirements. The system tracks actual performance metrics post-deployment, enabling data-driven decisions about

version adoption and retirement. This information appears in service detail views, helping consumers make informed choices about which versions to use.

Service Evolution Best Practices guide developers through successful lifecycle management. Design your services to be loosely coupled, have high cohesion, and cover a single bounded context. Design your services to be loosely coupled, have high cohesion, and cover a single bounded context.

The registration process encourages incremental evolution over radical redesigns, comprehensive testing before version releases, clear documentation of changes and migration paths, and maintaining backward compatibility whenever possible. Built-in validation rules enforce these practices, rejecting registrations that violate established patterns.

References

1. Semantic Versioning 2.0.0. (2023). “Software using Semantic Versioning MUST declare a public API. This API could be declared in the code itself or exist strictly in documentation. However it is done, it SHOULD be precise and comprehensive.” Available at: https://semver.org/
2. Cortex. (2024). “The 5 Stages of the Microservice Life Cycle and the Best Tools to Optimize Them.” Available at: https://www.cortex.io/
3. LeadSquared. (2024). “Service Lifecycle Management 101: Framework, Benefits, And Software.” Available at: https://www.leadsquared.com/
4. Simform. (2025). “14 Microservice Best Practices For Your Projects: The 80/20 Way.” Available at: https://www.simform.com/

5. MODAPTO Consortium. (2024). “D4.2 MODAPTO Pilot Implementation Report v1.” MODAPTO Project Deliverable.
6. ISO/IEC 19510:2013. “Information technology — Object Management Group Business Process Model and Notation.” International Organization for Standardization.