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

13.3.2 Implementing AAS serialization formats and validation

2025-08-11

Introduction

Implementing AAS serialization formats and validation represents a critical technical capability for ensuring interoperability, data integrity, and system reliability within MODAPTO implementations. Serialization formats provide the standardized mechanisms for storing, transmitting, and exchanging AAS information between different systems, platforms, and organizations, while validation ensures that AAS instances comply with metamodel specifications and operational requirements.

JSON serialization represents the primary format for web-based applications and cloud-native implementations due to its lightweight structure, broad platform support, and human readability. The AAS JSON serialization format defines precise mappings between metamodel elements and JSON structures, ensuring consistent representation across different implementations. JSON serialization supports both complete AAS representations and partial serialization for specific submodels or component updates, enabling efficient data exchange for different operational scenarios.

XML serialization provides comprehensive support for enterprise system integration, document management, and long-term archival requirements. The XML format offers stronger validation capabilities through XML Schema Definition (XSD) files that define structural constraints and data type requirements. XML serialization particularly supports complex hierarchical relationships and extensive metadata requirements that are common in industrial environments with strict documentation and compliance requirements.

Binary serialization formats optimize performance for high-frequency data exchange and resource-constrained environments such as embedded systems or real-time control applications. These formats minimize bandwidth requirements and processing overhead while maintaining complete semantic information. Binary formats are particularly important for MODAPTO implementations that require frequent reconfiguration or real-time monitoring of large numbers of components.

RDF (Resource Description Framework) serialization enables semantic web integration and advanced reasoning capabilities by representing AAS information as linked data. RDF serialization supports complex queries, inference operations, and integration with external knowledge bases that can enhance reconfiguration decision-making. This format is particularly valuable for research applications and advanced analytics that require sophisticated data relationship analysis.

The serialization implementation process begins with metamodel mapping that defines precise correspondence between AAS metamodel elements and serialization format structures. These mappings must preserve all semantic information while accommodating format-specific constraints and optimization opportunities. Bidirectional mapping ensures that AAS instances can be converted between different formats without information loss or semantic inconsistency.

Namespace management addresses the challenge of ensuring unique identification and avoiding naming conflicts when AAS instances are shared across organizational boundaries.

Serialization formats must support namespace declarations that enable precise identification of concept descriptions, property definitions, and other semantic references even when similar names are used by different organizations or standards bodies.

Version management within serialization formats supports evolution of AAS implementations while maintaining backward compatibility and migration capabilities. Version information enables systems to determine compatibility levels, trigger appropriate migration procedures, and maintain historical information when AAS structures change over time. Effective version management is essential for long-term asset management and system maintenance.

Validation frameworks provide systematic approaches to verifying that AAS instances comply with metamodel specifications, semantic requirements, and operational constraints. Structural validation checks verify that AAS instances conform to metamodel class structures, attribute requirements, and relationship cardinalities. This level of validation catches basic structural errors and ensures that AAS instances can be processed by compliant applications.

Semantic validation extends beyond structural compliance to verify that concept descriptions are properly referenced, property values conform to semantic definitions, and relationships between components are logically consistent. Semantic validation requires access to concept description repositories and may involve complex reasoning processes that check for inconsistencies or missing information.

Business rule validation addresses domain-specific requirements that go beyond general AAS compliance to ensure that AAS instances meet specific operational, regulatory, or organizational requirements. These rules might specify required submodels for specific asset types, validate that certain properties fall within acceptable ranges, or verify that appropriate approvals and certifications are documented.

Performance validation ensures that AAS instances can be efficiently processed under realistic operational conditions. This includes testing serialization and deserialization performance, validating query response times, and verifying that AAS implementations can handle expected transaction volumes without degradation. Performance validation is particularly important for real-time applications and large-scale deployments.

Error handling and recovery mechanisms address the challenges of processing AAS instances that contain errors, incomplete information, or incompatible format versions. Robust implementations provide graceful degradation capabilities that enable partial processing of valid information while reporting specific errors and providing guidance for corrective actions.

Automated validation tools streamline the validation process by providing immediate feedback during AAS development and deployment activities. These tools integrate with development environments, configuration management systems, and operational monitoring platforms to provide continuous validation capabilities that catch issues early in the development lifecycle and during operational use.

Compliance reporting capabilities generate detailed reports that document validation results, identify specific compliance issues, and provide recommendations for corrective actions. These reports support both technical troubleshooting activities and regulatory compliance documentation requirements that are common in industrial environments.

Integration testing validates that AAS serialization and validation implementations work correctly with external systems, including enterprise applications, partner organization systems, and third-party tools. Integration testing ensures that theoretical interoperability translates into practical operational capability under realistic conditions with actual system interfaces and data volumes.

References

1. World Wide Web Consortium. (2017). “JSON-LD 1.1 – A JSON-based Serialization for Linked Data”
2. World Wide Web Consortium. (2012). “OWL 2 Web Ontology Language Document Overview”
3. Internet Engineering Task Force. (2014). “RFC 7159 – The JavaScript Object Notation (JSON) Data Interchange Format”
4. Organization for the Advancement of Structured Information Standards. (2012). “OASIS Universal Business Language (UBL) 2.1”