Abstract

This proposal defines a cross-domain “Rules” Submodel that turns the Asset Administration Shell into a transparent, active system with declarative, executable behavior. The proposal is co-initiated by the Open Industry 4.0 Alliance (WG “Industrial Testing of pro-active AAS”) and Manufacturing-X (TG “Modulare Produktion”) and supported by Semiconductor-X. It is especially underlined by Semiconductor-X where it will be used in the supply-chain use cases demand-and-capacity-management. The idea is to standardize machine- and human-readable rules for three core needs: (1) validation of property changes (beyond simple enums, including cross-property/submodel checks), (2) rule-based publication/subscription of events and operation handling (including dynamic payload/argument composition), and (3) derived and inner update logic. A transparent, stateless Rules-Proxy enables pre-change blocking and post-change reactions without altering existing repositories. The submodel complements (not replaces) Qualifiers by providing explicit, executable logic and invites IDTA community collaboration.

Title of the submodel

Rules for AAS-Validation, AAS-Events and AAS-Operations

Submitting body (association, consortium, company, ...)

- Open Industry 4.0 Alliance – Working Group “Industrial Testing of pro-active AAS”
- Manufacturing-X – Topic Group “Modulare Produktion”
- Semiconductor-X

Short textual description of the planned content of the submodel.

Overall, the intended cross-domain Submodel “Rules” shall be designed to further evolve the Asset Administration Shell (AAS) into a self-contained, transparent, and active entity.

We envision the future AAS as a well-defined system, capable of interacting with its environment throughout the asset's lifecycle.

Today, the AAS already serves as both a data source and a data sink. However, key components that enable interaction—such as events and operations—are already embedded in the meta-model.
However, we identify a gap in the current available means: there is a lack of standardized mechanisms to
a) model the behaviour of an AAS, and
b) enable software solutions to execute this behaviour based on a machine- and human-readable model.

Motivation 1:
Validation

As of today, properties values of a “Type 2” AAS can be created or changed by means of the AAS REST-API. In addition to this classic Client-Server-Pattern, the AAS Metamodel foresees the BasicEventElement which allows a property-value change by means of a Publish and Subscribe pattern.

In both cases, it is desirable to have an option, to validate the value change. The general IT-principle “never trust the caller” means a module, function, or service must never blindly trust that the calling entity provides correct, complete, or secure data. In our case, such a module is the AAS itself.

To enable a validation of incoming property values beyond basic and simple checks along ENUMS (“Allowed Value List”), we see it as a necessity to allow the modelling and application of rules which apply in case of a property value change. We pay particular attention to ensuring that validation rules also consider the values of other properties beyond the one currently being modified. A trivial example for such rules is a dependency like: *'If the target country is Germany, then the postal code must be five digits long*.

While rules or constraint like these could certainly be covered simply by means of program code, we suggest a more model driven approach which makes the behaviour of an AAS transparent and easy to adjust.

In other words, the envisioned submodel shall be the container for all rules which are evaluated if a client is requesting an AAS property value creation or change.

Motivation 2:
Rule based publication/ subscription of events AND rule-based handling of operations

A property value change can result in issuing outbound events. A property value change can also result in calling operations. While to some degree the Metamodel entity “Qualifiers” might allow a certain modelling of conditions if and when an Event is published (or subscribed) resp. if and when an operation is called, we see nevertheless a need for a more explicit and transparent assignment of rules/condition/constraints which shell be applied in context of AAS Events and Operations.

As like the aforementioned validation-scenarios, also a cross property or also a cross submodel constellation might be the trigger to evaluate (and feed) rules which not only might decide if an event or an operation is triggered, but also how the payload or operation arguments are (dynamically) composed.

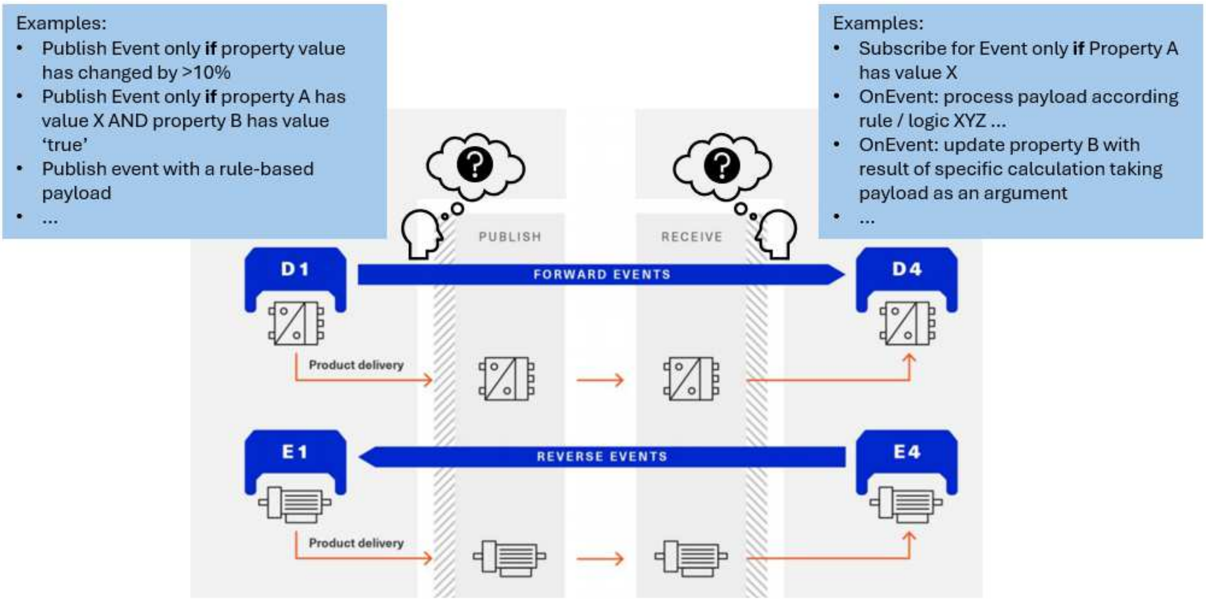

Figure 1 – Example for Pub Sub with AAS

Motivation 3:
"Inner update logic" of an AAS

A third (more might evolve) motivation for a dedicated submodel as a wrapper for all rules that are applicable for an AAS is the option to derive new values/results based on incoming property changes.
Means, incoming property X with value 1 results in an update of property Y based on a rule which takes value of X into account but also a value of property Z which adds up to the value of Y.
This particular cross property / cross submodel calculation relates also to the Motivation (or call is use case) 1 – Validation and Motivation 2 – Trigger of Events and Operations.

All three motivations/use cases have in common, that an explicit and transparent approach would be beneficial.

A first sketch of essential building blocks:

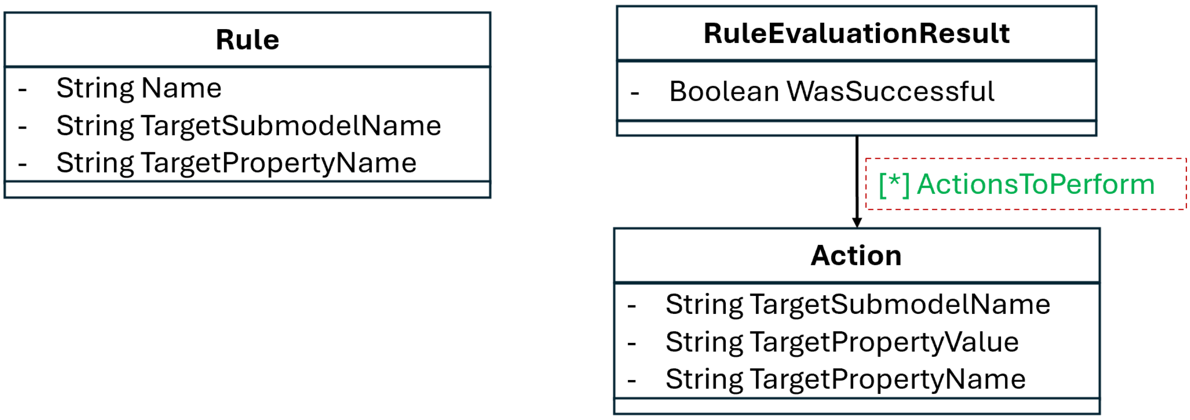

Figure 2 – Building Blocks for ruleset

As far as known, reference to other submodels, standards or specifications

- Submodel Process Parameters: IDTA-02031-1
- Submodel Process Parameters: IDTA-02031-2
  - What about state-dependent parameters? Rules-SMT can help here.
- Related are:
  - DMN (Decision Model and Notation) – is used in this proposal to define the rules
  - BPMN (Business Process Model and Notation) - the decision of whether BPMN is executed or not is defined externally – Rules-SMT can help here.
  - ECA (Event Condition Action)

Which other circles (association, society, company, ...) might be interested in participating

We see an opportunity to align with the Object Management Group. (OMG) The OMG is owner of the System Modeling Language (SysML v2) which relates closely to our vision.

Companies: Semiconductor-X, TwinTech, Aktorion Gemini Solutions, MetaLevel, Uhlmann-Group, XITASO, conplement, MMC, Twindustrial, KIWAI.EXPO21XX

A proposal for a working group consisting of chair, deputy chair and members who want to participate in the elaboration already exists? If yes, please indicate the name of the working group.

Working group does not yet exist – preliminary work was done in an Oi4 working group, but this does not provide a basis for an IDTA working group

Therefore, we are calling on the IDTA community to get involved.

Provisional contact persons:

- Christian Heinrich (christian.heinrich@openindustry4.com)
- Rüdiger Fritz (ruediger.fritz@aktorion.com)
- Daniel Lehner (daniel.lehner@twin-tech.at)
- Haris Avgoustinos (harris@expo21xx.com)
- Christina Bornträger (christina.borntraeger@meta-level.de)
- Patrick Sapel (sapel.p@uhlmann-group.com)
- Christian Günther (christian.guenther@conplement.de)
- Christian Methe (christian.methe@twindustrial.de)

Appendix

Appendix A – Why a dedicated "Rules" Submodel and not using Qualifiers?

During the conceptual phase of this proposal, we evaluated whether the existing **Qualifier** mechanism within the AAS Metamodel could be used to describe the conditions, rules, and logic outlined in the three motivations of the planned submodel.

Consideration of Qualifiers

Qualifiers are part of the AAS Metamodel (Part 1) and serve as **metadata elements** that restrict or contextualize the meaning or validity of a SubmodelElement or Property.
In principle, qualifiers could be used to **describe simple conditions**, such as threshold values, validity periods, or contextual restrictions (e.g. "valid when mode = cooling").

However, qualifiers are **purely descriptive** by design.
They provide static semantic information but **do not support executable or evaluable logic**, nor do they define relationships between multiple properties or submodels.
This makes them suitable for attaching auxiliary context to individual elements but not for representing **composite, conditional, or rule-based behaviors** that involve several elements or require runtime evaluation.

Rationale for a Dedicated "Rules" Submodel

After analysis, it became clear that the complexity and scope of the intended functionality exceed the expressive capabilities of qualifiers.
The following benefits justify the introduction of a dedicated submodel:

1. **Explicit and executable rule definitions**
   The proposed "Rules" submodel can hold rule structures (conditions, operators, affected properties, actions) in a machine-readable and interpretable form, enabling direct runtime evaluation without relying solely on external interpretation of static metadata.
2. **Cross-property and cross-submodel logic**
   Unlike qualifiers—which attach only to a single element—the "Rules" submodel supports dependencies between multiple properties and even across submodels. This is crucial for use cases such as validation (Motivation 1) or derived property updates (Motivation 3).
3. **Integrated event and operation handling**
   The submodel allows an **explicit mapping between conditions and resulting events or operations**, thus providing a transparent and maintainable way to define rule-based publication/subscription and operational triggers (Motivation 2).

4. **Transparency and maintainability**
   A model-driven rule container ensures that the AAS's behavior regarding validation, event triggering, and derived logic is **visible, configurable, and standardized**—instead of being hidden inside program code or dispersed across qualifiers.

5. **Extensibility and standardization potential**
   The "Rules" submodel offers a consistent framework that can evolve into a **cross-domain AAS template** for declarative rule management, aligning with IDTA's standardization goals and supporting future interoperability.

Summary

While **Qualifiers** remain useful for simple contextual restrictions, the **"Rules" submodel** introduces a structured, transparent, and extensible mechanism to define and execute complex validation, event, and update logic within the AAS.
This approach ensures that the AAS can evolve into a **self-contained, active digital entity**, capable of autonomous and rule-driven interaction within its lifecycle and operational context.

Appendix 2 – Architecture Proposal

This proposal defines a **transparent Rules Proxy architecture** for validation and monitoring of Asset Administration Shell (AAS) modifications. The architecture enables declarative business rule enforcement without modifying existing AAS repository implementations.

Architecture Principles

Transparent Proxy Pattern

A stateless Rules Processor Proxy intercepts all HTTP requests (GET, POST, PUT, DELETE) to AAS Submodel Elements, evaluates applicable rules, and forwards valid requests to the underlying SM Repository.

Two-Phase Rule Evaluation

Phase 1: Pre-Request Validation (Blocking)

- Rules are evaluated **before** any modification occurs
- Uses **state simulation** to evaluate both current and projected future state
- Can **block requests** that would violate business rules (HTTP 409 Conflict)
- Ensures data integrity constraints are enforced preventively

Phase 2: Post-Request Processing

- Rules are evaluated **after** successful modifications
- Enables observability and reactive behavior
- Cannot block or rollback the already-completed change

DMN Integration

Rules are defined using Decision Model and Notation (DMN) standard, enabling:

- Vendor-neutral rule definitions
- Visual rule modeling in decision tables
- FEEL (Friendly Enough Expression Language) for complex conditions
- Portability across different AAS implementations

Sequence Diagram

Exemplary sequence of application for submodel at runtime. (DMN is also just an example)

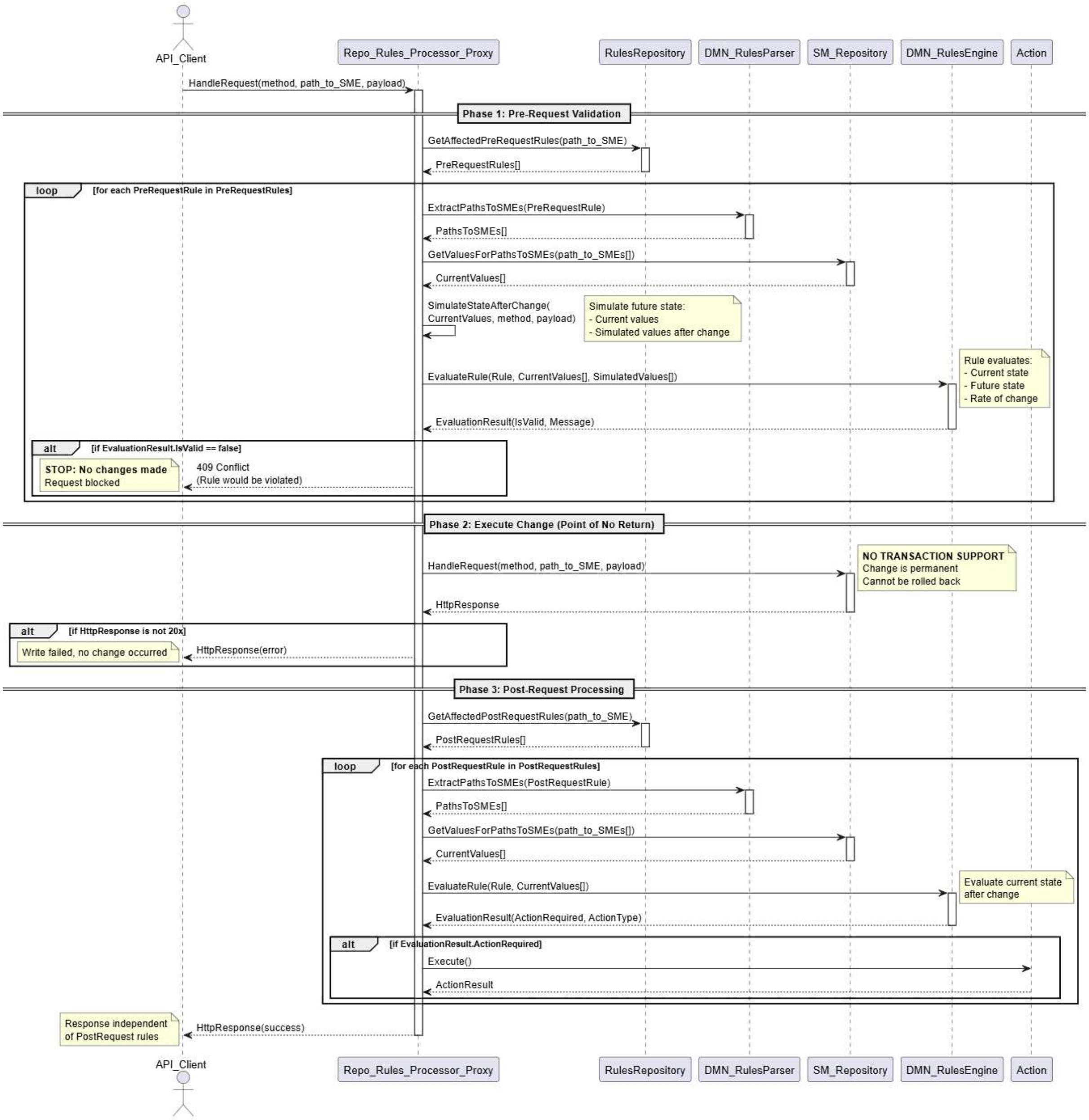

Figure 3 – Sequence Diagram