Intent Semantics: Explicit Verification Status demonstrated on the Asset Administration Shell

Moritz Dorn
Institute of Control Systems (IRS)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
ORCID: 0009-0004-8335-2147
moritz.dorn@kit.edu

Marcel Auer
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
ORCID: 0009-0008-9629-9807

Mike Barth
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
ORCID: 0000-0003-2337-063X

***Abstract*—Industrial information models like the Asset Administration Shell (AAS) form the operational foundation for production, maintenance, and digital twins in automation systems. When these models cross system boundaries between contributors, tools, or organizations, the context in which attributes were set is lost: the receiving system cannot distinguish verified values from template defaults, structural placeholders, or unverified candidates. A semantic ambiguity like this leads to inadequate interoperability, which carries significant costs in engineering and operations. This contribution proposes *intent semantics*, a formal annotation scheme that makes the verification status of every attribute value explicit and integrates into existing industrial metamodels without breaking changes. The scheme is built on a compact vocabulary of eight intent states and a deterministic process automaton that governs valid transitions, prevents semantically invalid regressions, and enables attribute-level quality gates ensuring that only verified values pass from engineering into operations. Integration is demonstrated for the Asset Administration Shell and AutomationML, and the approach is operationally deployed in a quality-assessment tool for an IDTA Submodel Template. Three lifecycle scenarios further demonstrate vocabulary coverage and regression prevention.**

***Index Terms*—intent semantics, information models, verification status, asset administration shell, cyber-physical systems, data quality**

I. Introduction

Current industrial policy actively promotes the exchange of information models across organizational boundaries through interoperable data spaces. The German ZVEI agenda for the industry of tomorrow [1] projects up to 420 billion EUR in value creation from digitalization by 2035, predicated on the reliable exchange of such models. The European Ecodesign for Sustainable Products Regulation (ESPR) [2] mandates a Digital Product Passport that is, at its core, an information model maintained across the entire product lifecycle. Yet inadequate interoperability already carries significant cost: Gallaher et al. [3] estimated $15.8 billion per year in the U.S. capital facilities industry alone. These initiatives rest on a common premise: the contents carried by information models can be trusted. This premise, in turn, depends on standards that define how physical assets are represented digitally.

Standards such as the Asset Administration Shell (AAS) [4], OPC UA, and AutomationML [5] form the backbone of modern automation systems, enabling interoperability across the asset lifecycle. As Industry 4.0 matures, these models increasingly serve not only as engineering documentation but as digital twins and therefore the operational basis for production, maintenance, and simulation. This evolving role raises the bar for fidelity: the data carried by a digital twin must accurately reflect the physical asset it represents. Grieves' original term for the digital twin concept, "Information Mirroring Model" [6], reveals this role: the information model is the conceptual identity of the digital twin. Stachowiak's general model theory [7] adds that any model is inherently a reduced representation, capturing only attributes relevant to its intended purpose. In ISO 23247, these roles are made explicit by defining a digital twin as a fit-for-purpose (Stachowiak) digital representation that is kept synchronized (Grieves) with a physical original [8].

Taken together, these definitions expose a fundamental requirement: the values of the attributes carried by an information model must faithfully mirror the reality the attributes claim to represent. In practice faithfulness means that a competent authority has verified each value to lie within an accepted tolerance. This leads to the following assumption: *an information model may begin representing reality only if all its attributes' values faithfully mirror reality.*

Engineering of industrial systems is an inherently distributed and concurrent process in which multiple contributors collaboratively specify a reality that does not yet exist. Information models, e.g., AAS Submodels, are created, refined, and maintained by different persons, in different tools, across different roles, organizations, and lifecycle phases. Template-based workflows are standard practice [9], especially within the AAS [10]: models are derived from templates, inheriting attribute values that may be domain-standard defaults, structural placeholders, or automatically generated candidates. Each of these values' relationship to reality differs, but this distinction is implicit. It was made during a design decision based on the discrete knowledge of the engineer who set the value [11].

When the model crosses a system boundary, whether between individual contributors, between engineering tools, between organizations in a supply chain, or between engineering and operations, this tacit knowledge does not transfer [12]. Because the context is lost, the receiving system, tool, or person sees only values, with no indication of whether a value's relationship with reality has been verified, inherited from a template, set to just some value, or deliberately left empty.

Yet no standardized mechanism exists to annotate which attribute values have been verified against reality and which have not. This loss of verification context manifests in two concrete problems.

Context loss at system boundaries. Generally, when an information model is exchanged across organizational boundaries, the recipient cannot distinguish which attribute values the sender has verified against reality and which merely persist from a template or were never actively set. Consider an equipment specification shared between an OEM and a supplier. A material code may reflect a verified, intended value. A protection rating may carry a domain-standard value never checked for this particular device. An operating temperature field may have been left for the recipient to provide. To the receiving system, all three are indistinguishable. The recipient must resort to manual inquiry or implicit assumptions to determine the status of each value, contributing to the interoperability costs noted above.

Absence of attribute-level quality gates. Quality gates exist at the process level, for example in the V-model according to the German guideline VDI 2206 [13] or through Verification, Validation, and Uncertainty Quantification (VVUQ) procedures [14]. However, even though quality can be evaluated for an individual attribute, no mechanism can enforce it at the level of individual attribute values. At the latest when an information model transitions from engineering into operations, a gate is needed to verify that all values have been confirmed against reality. A tolerance grade of IT7 (ISO 286) may be an appropriate domain-standard default for general machining but insufficient for a specific precision assembly. An OPC UA endpoint address such as `192.168.0.1` may be a verified configuration or a generic default carried over from a template. This way, a value that has been completely sensible for engineering purposes but has to be re-verified for operations cannot be differentiated from values that have been verified already. Without explicit annotation, neither case is detectable by automated systems because both values are syntactically valid.

To address these problems, this paper proposes *intent semantics*: a formal annotation scheme that makes the verification status of every attribute value in an information model explicit. The approach comprises a compact vocabulary of eight intent states and a deterministic process automaton that governs transitions between them through three operation classes (*set*, *confirm*, *refute*). Three claims are made:

- **C1** Intent semantics can be integrated into existing industrial metamodels without breaking changes.
- **C2** The eight intent states are sufficient to capture the verification statuses encountered in practical engineering workflows.
- **C3** The process automaton prevents semantically invalid state regressions that current practice cannot detect.

The paper proceeds from problem formulation (Section II) through related work (Section III) to the approach (Section IV), followed by evaluation (Section V), discussion (Section VI), and conclusion (Section VII).

II. Problem Formulation

The two manifestations described in Section I share a common root cause: the absence of an explicit, machine-readable annotation that captures how a given attribute value relates to reality. This section formalizes that observation.

Let an information model comprise a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$, where each attribute a_i carries a value v_i. In addition to its value, each attribute carries an implicit *verification status* that describes whether v_i has been confirmed against reality, inherited from a template, deliberately left empty, or set under any other circumstance. Conceptually, this status can be modeled as a function $intent\colon A \to S$ that maps each attribute to exactly one element of a finite set of verification statuses S. In current practice, however, neither S nor the value of $intent$ is ever made explicit. Current practice operates under the implicit assumption that every attribute value faithfully represents reality:

$$\forall\, a_i \in A\colon \quad intent(a_i) = actual \tag{1}$$

In practice, however, attribute values carry fundamentally different verification statuses that are not systematically made explicit:

$$\exists\, a_i \in A\colon \quad intent(a_i) \neq actual \tag{2}$$

The discrepancy between assumption (1) and reality (2) produces the two manifestations introduced in Section I: context loss at system boundaries, where a recipient cannot distinguish values with different verification statuses, and the absence of attribute-level quality gates, where a default value might transition into operations because it cannot be flagged. A solution requires two components: (i) a vocabulary that explicitly captures the verification status of individual attribute values, replacing the implicit assumption of Eq. (1), and (ii) a formal process model governing valid transitions between verification statuses, preventing semantically invalid regressions such as overwriting a verified value with a template default. The following section examines how existing approaches address parts of this problem and where they fall short.

III. Related Work

The problem of trusting attribute values touches several research areas. This section reviews data quality dimensions, property value statements, and adjacent verification and annotation approaches, identifying what each contributes and where it falls short.

A. Data Quality Dimensions

In operational technology (OT), every attribute value carries an implicit claim about a physical counterpart, unlike information technology, where a database record may constitute its own ground truth. Wand and Wang [15] formalize this in their work on data quality anchoring: deficiencies arise as *inconformities* between the real-world system inferred from an information system and the view obtained by direct observation. Teh et al. [16] confirm the resulting gap in a systematic review of sensor data quality, finding that only two established dimensions, incompleteness and inaccuracy, apply directly to physical sensor data.

Building on this anchoring, data quality research provides established frameworks for assessing whether data is suitable for its intended purpose, for example whether records are accurate, complete, timely, and consistent. Wang and Strong [17] proposed a hierarchical taxonomy of such dimensions that has become a widely adopted reference. Batini et al. [18] surveyed methodologies for data quality assessment and improvement, finding that most approaches share a common strategy: they measure quality dimensions as ratios over populations of data items, for example the percentage of records in a dataset that satisfy a completeness criterion [18]. ISO/IEC 25012 [19] codifies 15 data quality characteristics along these lines, but its scope is explicitly limited to structured data in computer systems and excludes real-time sensor data and embedded devices.

These aggregate assessments do not address the verification status of individual attribute values: a dataset may score highly on aggregate accuracy while individual values remain unverified defaults or placeholders, because the assessment operates on statistical populations rather than single attributes. Data quality dimensions describe *what to measure* across a dataset, but provide no mechanism to annotate whether a specific value has been confirmed against reality, inherited from a template, or deliberately left as a placeholder.

B. Property Value Statements

Epple [20] introduced property value statements (*Ausprägungsaussagen*) to formalize how properties of technical systems are described. The central insight is that the same property of a technical system can be described by multiple, coexisting statements that differ in use: a Requirement states what a value should be, a Measurement records what has been observed, an Assurance declares what a supplier guarantees, and a Setting specifies what has been configured. For example, the operating pressure of a pump may simultaneously carry a Requirement of 6 bar (from the specification), a Measurement of 5.9 bar (from commissioning), and an Assurance of 6 bar (from the manufacturer). Epple et al. [21] extended this into a Property Value Statement Model that enables automated reasoning about consistency and completeness of property descriptions across engineering tool chains.

These approaches formalize *who asserts what, with what authority, and under which conditions*, but they do not capture whether the underlying attribute value itself has been verified against reality. A property value statement of type Assurance may reference an attribute whose value is still an unverified default inherited from a template. In this case, the assurance is semantically hollow. Even though the value is set, the actor has not verified the value they will guarantee. Detecting this inconsistency is not the scope of the property value statement framework, because it does not operate on the conceptual level of the value but of the attribute.

Property value statements and the verification status of attribute values are therefore orthogonal concerns. The statements capture claims *about* a value; what is missing is an annotation of *the value itself*: whether it has been confirmed against reality, inherited from a template, or deliberately left as a placeholder. Without such an annotation, the validity of a property value statement cannot be fully assessed.

C. Adjacent Approaches

1) Annotation Mechanisms.: The AAS metamodel [4] provides Qualifiers as a generic annotation mechanism for Submodel Elements. Qualifiers attach typed key-value metadata to any element, making them a natural candidate for carrying verification status information. The IDTA Submodel Template guideline [10] standardizes the qualifier types `SMT/ExampleValue` and `SMT/DefaultValue` to give hints for template conformance. However, these qualifier types operate at template design time and are intended to be removed when instantiating a submodel [10]. In practice, instantiation is often incremental: a submodel may be partially populated while some attributes still carry template qualifiers, and when such a model is exchanged across system boundaries, the receiving party cannot distinguish which qualifiers have already been resolved and which have not. Moreover, what it means when these qualifiers are absent is not defined.

Provenance models such as W3C PROV [22] track the origin and transformation history of data artifacts. While provenance answers *where a value came from*, it does not answer *whether the value has been verified against reality*. A value with complete provenance that has crossed an organizational boundary may still be an unverified template residue, because provenance preserves the chain of custody but not the verification context that was lost at the boundary.

2) Process-Level Quality Assurance.: Quality assurance in systems engineering operates primarily at the process level. The V-model as described in VDI 2206 [13] defines quality gate checkpoints at which design artifacts are reviewed before progressing to the next development phase. Verification, validation, and uncertainty quantification (VVUQ) frameworks mandate credibility assessments before digital twins are deployed operationally [14]. These approaches ensure that entire models or subsystems pass defined criteria at defined milestones, but they do not extend to individual attribute values, because the gate assesses the artifact as a whole rather than the state of each value it contains. Similarly, IEC 62264 [23] defines quality operations management as one of four manufacturing operations categories, providing quality assurance for products and processes through statistical pro-

cess control, but its information models contain no mechanism for annotating the verification status of individual attribute values. Consequently, a model may pass a quality gate while individual attributes still carry placeholder or default values.

Research gap. The two problems identified in Section I remain only partially addressed by existing work. Rudimentary annotation mechanisms exist, such as AAS Submodel Template qualifiers (`SMT/ExampleValue`, `SMT/DefaultValue`), but these are informal, limited to template conformance, and intended to be removed upon instantiation. Across the remaining areas the same gap recurs: provenance, data quality frameworks, property value statements, and process-level quality assurance each address part of the problem, but none annotate the verification status of an individual attribute value.

What is missing is the *combination* of two capabilities that no existing approach provides together: a formal vocabulary that captures the verification status of individual attribute values, and a deterministic process model that governs valid transitions between these states. Such a combination would make the verification context of each value explicit, transferable across system boundaries, and amenable to automated reasoning. The following section introduces *intent semantics* to close this gap.

IV. Approach: Intent Semantics

As outlined in Section I, intent semantics comprises two components: a vocabulary (Section IV-A) and a process automaton (Section IV-B). The vocabulary provides a minimal yet sufficient set of states that cover the verification statuses encountered in practical engineering workflows. The process automaton formalizes valid transitions and prevents semantically invalid regressions, for example when a template is re-applied to an existing model and overwrites a previously verified value with a template default. The concept integrates with established metamodels via IEC 61360-conformant Concept Descriptions [24] without requiring changes to the metamodel itself. This is demonstrated through the AAS metamodel, where intent semantics are implemented via Qualifiers.

A. Vocabulary

Intent semantics is defined as the explicit annotation of the verification status of an attribute value in an information model. Which attributes are included in a model is a design decision (cf. Stachowiak's reduction feature [7]); whether the included attributes' values have been verified against reality is the concern of intent semantics. Intent Semantics is built on one foundational rule that follows the assumption in Section I:

Rule 1 (Rule of intent semantics): An information model may begin representing reality only if all its attributes' values carry the intent actual.

To formalize which verification statuses an attribute value can carry, the following set of intent states is defined:

$$S = \{\, actual,\ hypothetical,\ default,\ generated,\ empty,\ placeholder,\ outdated,\ invalid \,\} \tag{3}$$

TABLE I: Intent semantics vocabulary. Each state describes a specific verification status of an attribute value.

Intent	Description
actual	*Verified*. Represents the mapped reality. Target state.
hypothetical	*Verified* as describing a state that does not (yet) exist. May be verified to *actual*.
default	*Verified* as domain-standard value. Still requires instance-specific confirmation.
generated	A *set* value as candidate for verification. This can be set by a contributor or automatically produced (calculated, imported, estimated ...). Not yet verified.
empty	Deliberately *set* or left *empty*. Distinguishes intentional absence from errors.
placeholder	A *set* value without intent to be verified. Must be replaced before operational use.
outdated	Was *actual* but the *re-verification failed*. Must be replaced because reality has changed.
invalid	A value whose *verification failed*. Must be replaced because the value is incorrect.

The function $intent\colon A \to S$ assigns each attribute its current verification status, formalizing the observation from Section II that not all attributes carry the status *actual*. Every state other than *actual* marks a specific deviation where the foundational rule (Rule 1) prevents representation of reality.

Table I provides descriptions for the complete vocabulary. The relation and boundaries of three particular pairings are clarified below.

Generated vs. hypothetical. A *generated* value was produced by a contributor or a process (calculation, import, estimation) and awaits verification, implying that reality exists and verification is pending. A *hypothetical* value was classified through verification as describing a state that does not currently exist at this instance, such as a planned operating point or simulation target.

Invalid vs. outdated. Both indicate that a value does not match reality, but their histories differ. *Invalid* means verification has failed and the value was never confirmed as correct, while *outdated* means the value was once *actual* but reality has since changed, detected through failed re-verification. The distinction matters for downstream handling: an *outdated* value was the last confirmed representation of a changed reality and may serve as a provisional approximation until it is replaced, whereas an *invalid* value offers no such relation to reality.

Default vs. generated. Both represent values not yet confirmed for a specific instance. However, *default* has been explicitly classified as domain knowledge through verification (e.g., tolerance grade IT7 as a standard machining specification), while *generated* was written without any verification.

1) Design Decision: Compactness.: The vocabulary deliberately comprises only eight states. Subtypes of *generated*, such as calculated, imported, estimated, or initialized, are subsumed under a single state and finer differentiation is delegated to optional provenance attributes. This subsumption is justified because these subtypes would share the same transitions in the process automaton (Section IV-B). Their provenance differs, but not their verification path. Note that measurement is not a subtype of *generated*: measuring an attribute value against

reality is an act of verification, not of setting, and may therefore confirm the value as *actual*. This complexity reduction is motivated by intent semantics as a deep-rooted addition to all attributes of an information model, where a large state set might hinder adoption and tooling support. While the state set is kept small, the vocabulary is designed to be extensible.

2) Design Decision: Extensibility.: The design follows a principle established by HTTP status codes [25]: applications must not understand specific codes but every *class* of any status code. So an HTTP client that does not recognize status code 431 can fall back to the semantics of 400. Intent semantics is designed with this principle in mind. The eight intent states form a mandatory base layer. Beyond it, the annotation may be optionally extended with provenance metadata (who, when, how). Systems that do not understand these extensions can safely ignore them and still reason correctly about the intent state itself. This ensures that the core vocabulary remains compact and universally interpretable, while domain-specific extensions can add richness without breaking interoperability.

With the vocabulary in place, any contributor (human or system) can annotate the intent behind an attribute's value at the point of setting or verifying it. Conversely, a receiving system can inspect the intent state and act accordingly by adopting an *actual* value as trusted, verifying non-*actual* values, or initiating feedback that a sufficient value needs to be supplied. In a template-based workflow, the same information may serve as a work list: a tool can highlight attributes that have not yet reached the desired state.

B. Process Automaton

The process automaton governs which transitions between intent states are valid and through which operations they are triggered. Three operation types act on attribute values. ***Set*** writes a new value to the attribute, targeting a specific unverified state. ***Confirm*** checks the existing value against reality or domain rules and classifies it into a confirmed state. For instance, $\mathit{confirm}(\mathit{default})$ classifies the value as a domain standard, whereas $\mathit{confirm}(\mathit{actual})$ confirms it against reality. ***Refute*** determines that the value's state cannot be confirmed. A failure from a non-actual state yields $\mathit{refute}(\mathit{invalid})$, while re-verification failure of a previously *actual* value yields $\mathit{refute}(\mathit{outdated})$. Both confirm and refute abstract the concept from the concrete verification procedure: the automaton consumes only its outcome, which depending on the attribute ranges from human judgment to physical measurement against reality. Each operation type targets a distinct subset of S. Together these subsets partition the state set:

$$S_{set} = \{\mathit{empty},\ \mathit{placeholder},\ \mathit{generated}\} \quad (4)$$

$$S_{confirm} = \{\mathit{actual},\ \mathit{default},\ \mathit{hypothetical}\} \quad (5)$$

$$S_{refute} = \{\mathit{invalid},\ \mathit{outdated}\} \quad (6)$$

with $S_{set} \cup S_{confirm} \cup S_{refute} = S$ and all pairwise intersections empty. From this partition, an input alphabet is constructed in which each symbol encodes both the operation type and its target state:

$$\Sigma = \{\, \sigma(s') \mid \sigma \in \{\mathit{set},\ \mathit{confirm},\ \mathit{refute}\},\ \ s' \in S_\sigma \,\} \quad (7)$$

This yields $|\Sigma| = |S| = 8$ concrete input symbols (e.g., $\mathit{confirm}(\mathit{actual})$, $\mathit{confirm}(\mathit{default})$, $\mathit{set}(\mathit{generated})$). Since each input symbol carries its target state, the classification outcome is determined externally by the verification procedure and encoded in the symbol itself.

These components are combined into a labeled transition system:

$$\mathcal{A} = (S,\ \Sigma,\ \delta,\ s_0,\ S^*) \quad (8)$$

where S and Σ are defined in Eqs. (3) and (7), $\delta \subseteq S \times \Sigma$ is the transition relation whose valid (source state, input) pairs are enumerated in Table II (the target state is the parameter of the input symbol), $s_0 = \mathit{empty}$ is the initial state, and $S^* = \{\mathit{actual}\}$ the acceptance set. The automaton is deterministic: although Table II lists multiple reachable states per source state, each concrete input symbol $\sigma(s')$ uniquely identifies both operation and target, so the target state is fixed by the symbol's parameter. Figure 1 shows the corresponding state diagram.

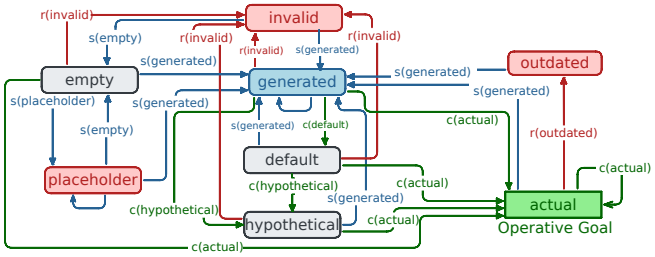

Fig. 1: Process automaton for intent semantics. Blue arrows denote *set* transitions, green arrows *confirm*, red arrows *refute*. Red borders mark dead-end states (no outgoing verify transitions). The acceptance state *actual* is highlighted green.

1) Dead-End States.: Three states have no outgoing verify transitions and can only be exited through a *set* operation.

TABLE II: Transition relation $\delta \subseteq S \times \Sigma$. Each cell lists valid target states reachable from the source state via the respective transition class.

Source s	$\mathit{set}(\cdot)$	$\mathit{confirm}(\cdot)$	$\mathit{refute}(\cdot)$
empty	gen, plh	act	inv
placeholder	empty, gen, plh	—	—
generated	gen	act, def, hyp	inv
default	gen	act, hyp	inv
hypothetical	gen	act	inv
invalid	empty, gen	—	—
actual	gen	act	out
outdated	gen	—	—

act = *actual*, def = *default*, gen = *generated*,
hyp = *hypothetical*, inv = *invalid*, empty = *empty*, out = *outdated*, plh = *placeholder*.

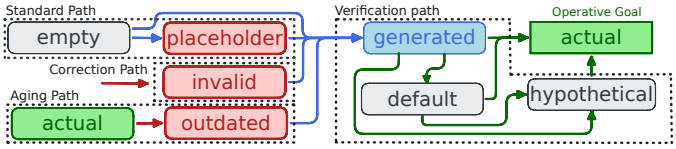

Fig. 2: Representative process paths through the intent semantics automaton.

Formally:

$$S_{\text{dead}} = \left\{ s \in S \;\middle|\; \nexists\, s' \in S_{confirm} \cup S_{refute} \colon (s, \sigma(s')) \in \delta \right\} \tag{9}$$

which yields $S_{\text{dead}} = \{placeholder, invalid, outdated\}$. This set is a derived property of δ, not an arbitrary design choice: a value identified or set as incorrect cannot become correct through verification alone, because it must first be replaced. Notably, $S_{\text{dead}} \cap S_{confirm} = \emptyset$: this means confirmation will never lead into a dead-end state.

2) Typical Process Paths.: Figure 2 illustrates how the acceptance state *actual* may be reached. The *verification path* is the fundamental building block because the remaining three paths lead to it: starting from *generated*, a single *confirm* reaches *actual*, passing through *default* when a value is first classified as a domain standard or through *hypothetical* when it refers to a not yet existent condition. The *standard path* starts with an *empty* value and ends by setting a generated value. The *correction path* recovers from a refute after a failed verification (*invalid*) through a replacing *set*, after which the verification path is attempted again. The *aging path* handles a previously *actual* value that fails re-verification (*outdated*), which likewise requires a *set* before re-verification.

The automaton turns the vocabulary into a process that can be implemented by systems that take intent semantics into account. When a system is *intending* to set or verify a value it can use the process to check if this *intent* is valid. Additionally, a quality gate may assert a verification state like $\forall a \colon intent(a) = actual$.

Together, the vocabulary (Section IV-A) and process automaton (Section IV-B) provide a complete formal framework for intent semantics. The vocabulary captures the verification status of individual attribute values; the automaton constrains which state transitions are permissible and ensures that the path toward the acceptance state *actual* is governed by explicit operations. The following section evaluates whether this framework is sufficient in practice.

V. Evaluation

This section evaluates intent semantics along three dimensions: integration into existing industrial metamodels (Section V-A), vocabulary coverage through representative lifecycle scenarios (Section V-B), and regression prevention by construction of the transition relation (Section V-C).

A. Information Model Integration

The vocabulary and process automaton are defined independently of any particular metamodel. Integration requires only a carrier mechanism that can attach an intent state to an attribute value. Two general strategies can be applied depending on the capabilities of the target metamodel:

I. **Native modeling.** If the metamodel supports user-defined type libraries or annotation mechanisms, the intent vocabulary can be modeled natively within the metamodel's own constructs.

II. **IEC 61360 Concept Description reference.** Alternatively, each intent state can be defined to be referenced as an IEC 61360 Concept Description [24]. This strategy is carrier-agnostic: the same Concept Descriptions can be referenced with any metamodel that supports external semantic identifiers.

Both strategies are purely additive because they require no modification to the target metamodel. This additive nature also covers brownfield environments: legacy systems that do not interpret intent semantics still receive a standard-conformant model and simply ignore the annotations. To validate claim C1 both strategies have been implemented. Section V-A1 demonstrates Strategy I (native modeling) for AutomationML, and Section V-A2 demonstrates Strategy II (Concept Description reference) for the AAS.

1) AutomationML Integration: AutomationML's [5] CAEX data model provides Attribute Type Libraries (ATL), in which a single `AttributeType` named `intent_semantics` is modeled to carry the eight-valued vocabulary as a `NominalScaledType` constraint. A concrete attribute is annotated by adding `intent_semantics` as a nested sub-attribute that references the ATL entry and carries the intent state as its value. The integration adds exactly one sub-attribute per annotated attribute and requires no changes to the CAEX schema. This integration is operationally deployed in a generator for Q-Metric [26], a role-based quality assessment tool for the IDTA Submodel Template 02005-1-1 "Provision of Simulation Models" [27]. The SMT foresees an AML `QualityMetricFile` containing the quality assessment. Using the generator, this AML file can be incrementally created from a template. It leverages intent semantics to preserve context between edits. The reference implementation including the ATL definition is publicly available[1].

2) AAS Integration: The AAS metamodel [4] does not provide a native enumeration mechanism. Integration therefore follows Strategy II: the intent vocabulary is defined externally as IEC 61360 Concept Descriptions, and the AAS references these definitions through its Qualifier mechanism.

Intent semantics uses ConceptQualifiers, which apply standalone semantic meaning to an element. Each intent state maps to a Qualifier instance attached to the target SubmodelElement with type `IntentSemantics` and the intent state as its string value. Listing 1 shows an annotation with the intent `generated`.

Listing 1: AAS Qualifier carrying intent annotation (simplified for readability).

[1] https://github.com/KIT-IRS/Q-Metric-Generator

```
"qualifiers": [{
  "type": "IntentSemantics",
  "valueType": "xs:string",
  "value": "generated",
  "semanticId": {"type": "ExternalReference", "keys": [{
    "type": "GlobalReference",
    "value": "irs.kit.edu/.../IntentSemantics" }]},
  "valueId": {"type": "ExternalReference", "keys": [{
    "type": "GlobalReference",
    "value": "irs.kit.edu/.../IntentSemantics/generated" }]}
}]
```

In it the `semanticId` references the Concept Description defining the intent vocabulary and the `valueId` the specific intent state. Since Qualifiers attach to any SubmodelElement, intent annotations apply at all structural levels.

Semantic Definition. Each intent state is formally defined as an IEC 61360 Concept Description following the `DataSpecificationIEC61360` template. The complete Concept Description defining the vocabulary and all eight state-specific value references is publicly available[2]. Practitioners can include Concept Descriptions directly in their AAS Environment or reference a service that provides an IEC 61360 interface. The authors implemented a prototypical service for this purpose and hosted the Concept Descriptions via concept-store[3], a lightweight service that resolves IRDIs to their corresponding IEC 61360 Concept Descriptions and additionally can serve as a proxy and cache for the IEC CDD.

B. Vocabulary Coverage

The vocabulary is evaluated by tracing three attribute lifecycles from different engineering domains through the process automaton, each exercising a distinct process path (standard, correction, aging) from Section IV-B. Table III summarizes the scenarios.

Scenario 1 exercises the standard path through the tolerance grade example from Section I: a template value is classified as a domain-standard default, then narrowed to a tighter grade for a planned precision assembly that has not yet been manufactured, and finally confirmed through post-manufacturing quality control. Scenario 2 demonstrates the correction path using the OPC UA endpoint address from Section I: the attribute begins empty (*empty*), receives a structural placeholder, is overwritten by an auto-generated address, fails verification (*invalid*), and reaches *actual* only after manual correction and commissioning. Scenario 3 traces a simulation parameter through the standard and the aging path: a friction coefficient is inherited from a template (*generated*), classified as a domain standard (*default*), replaced by a measurement, validated to *actual*, and eventually invalidated through wear (*outdated*).

Together, the three scenarios exercise all eight intent states and show why they must remain distinguishable: collapsing any two would discard information a downstream system needs. This supports Claim C2 through representative lifecycles from three engineering domains rather than an exhaustive enumeration of workflows.

[2]https://github.com/KIT-IRS/Intent-Semantics

[3]https://github.com/KIT-IRS/concept-store

TABLE III: Three attribute lifecycles from different engineering domains. Together, the scenarios cover all eight intent states.

Phase	Intent	Transition
Scenario 1: Tolerance grade of a bearing bore (standard path)		
Template value IT7	gen	$set(gen)$: from template
Classified as std.	def	$confirm(def)$: general machining standard
Set to IT5	gen	$set(gen)$: planned precision assembly
Verify design specs	hyp	$confirm(hyp)$: not yet manufactured
Final inspection	act	$confirm(act)$: measured & in tolerance
Scenario 2: OPC UA endpoint address (correction path)		
Attribute created	empty	initial state
Template assigns some IP	plh	$set(plh)$: structural placeholder
Commissioning assigns addr.	gen	$set(gen)$: auto-generated
Connectivity fails	inv	$refute(inv)$: unreachable
Manual correction	gen	$set(gen)$: corrected address
Commissioning	act	$confirm(act)$: connection verified
Scenario 3: Friction coefficient μ (aging path)		
Template created	gen	$set(gen)$: value 0.3 from template
Domain Expert reviews	def	$confirm(def)$: domain standard
Measurement	gen	$set(gen)$: yields 0.28
Validation	act	$confirm(act)$: confirmed
After 2 years	out	$refute(out)$: wear detected
New measurement	gen	$set(gen)$: yields 0.35
Re-validation	act	$confirm(act)$: confirmed

Abbreviations as in Table II.

C. Regression Prevention

Regression prevention is a direct corollary of the transition relation δ defined in Section IV-B. A regression is a transition that would move an attribute value to a state of lower verification confidence without an appropriate operation. Since δ explicitly enumerates all valid $(s, \sigma(s'))$ pairs (Table II), any pair absent from δ is forbidden by construction.

For example, a verified value (*actual*) cannot be degraded to a structural placeholder via *set*, and the three dead-end states *placeholder*, *invalid*, and *outdated* (Eq. (9)) cannot reach *actual* through *confirm* because no such transition exists in δ.

More generally, the automaton guarantees that any path from an unverified state to the acceptance state *actual* must pass through at least one *confirm* transition. No sequence of *set* operations alone can reach *actual*, because $actual \notin S_{set}$ (Eq. (4)). This confirms Claim C3: the process automaton prevents semantically invalid regressions by construction of δ.

VI. Discussion

1) Strengths.: Intent semantics provides a compact annotation scheme that makes the verification status of individual attribute values explicit. With eight states, the vocabulary is small enough for consistent adoption yet expressive enough to cover practical engineering scenarios. The separation of setting a value from verifying it enables composing instead of conflating data provenance with verification status. The layered design (Section IV-A) implements an extensibility model, where the eight mandatory states form a stable core, while optional metadata such as provenance, timestamps, or domain-specific subtypes can be attached without affecting the automaton.

The approach complements existing work. Epple's property value statements [21] formalize assertions *about* values while intent semantics captures the state *of* the value itself. A

Property Value Statement of type *Assurance* attached to a value with intent *placeholder* constitutes a contradiction that is invisible without intent annotation. Intent semantics makes such inconsistencies detectable, providing a layer of information that property value statements can build upon.

The concept enables *gating*: the formal verification that all attribute values carry the intent *actual* before an information model transitions from engineering into operations.

2) Limitations.: For now, intent annotations are not set automatically. Their maintenance requires tool integration or manual discipline. Each attribute value requires an additional annotation, introducing a linear overhead. The vocabulary is deliberately fixed at eight states, which may prove too restrictive for specialized domains requiring finer distinctions. Finally, an intent annotation is a claim by its contributor, like any other model content: intent semantics assumes cooperative contributors, while protection against deliberately false annotations remains the concern of orthogonal security mechanisms.

3) Generalizability.: The generalizability of the approach follows from the deliberate separation of the vocabulary and process automaton $(S, \mathcal{A})$ from the carrier mechanism. The automaton is carrier-agnostic in that only the mapping $S \rightarrow$ Carrier varies across metamodels. The vocabulary and the transition relation δ remain identical regardless of how intent states are represented in the model. Section V-A demonstrates this through a native modeling and a reference modeling strategy.

VII. Conclusion and Future Work

This paper introduced *intent semantics*, a formal annotation scheme that makes the verification status of attribute values in industrial information models explicit. It addresses the two manifestations identified in Section I: values get an intent annotation, so recipients can distinguish verified values from others, and automated gating is enabled before a model crosses a system boundary.

The eight-state vocabulary and the deterministic process automaton with its three transition classes (*set*, *confirm*, *refute*) formalize valid transitions while preventing semantically invalid regressions. The concept integrates with AAS and AutomationML without breaking changes and is operationally deployed for the IDTA Submodel Template 02005-1-1 [27].

Future work proceeds in three directions. First, concurrent engineering may greatly benefit from a merge policy leveraging intent semantics. This necessitates a precedence ordering over intent states, propagation of intent through hierarchical structures, and cross-level conflict resolution. Second, tool-level integration for automatic intent annotation by contributing the concept to open-source modeling libraries and SDKs. Third, the extensibility layer introduced in Section IV-A would enable richer reasoning based on more elaborate provenance concepts.

Finally, intent semantics provides a formal foundation for establishing trust in information models exchanged through industrial data spaces, as envisioned by the ZVEI agenda [1] and the European Digital Product Passport [2].

References

[1] ZVEI – Verband der Elektro- und Digitalindustrie, "Agenda für die Industrie von morgen," ZVEI, Tech. Rep., 2025.

[2] European Parliament and Council of the European Union, "Regulation (EU) 2024/1781 establishing a framework for setting ecodesign requirements for sustainable products (ESPR)," 2024. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2024/1781/oj

[3] M. P. Gallaher, A. C. O'Connor, J. L. Dettbarn, Jr., and L. T. Gilday, "Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry," National Institute of Standards and Technology (NIST), Tech. Rep. GCR 04-867, 2004.

[4] IDTA, "Specification of the Asset Administration Shell – Part 1: Metamodel," IDTA, Frankfurt am Main, Germany, Tech. Rep., 2023. [Online]. Available: https://industrialdigitaltwin.org/content-hub/aasspecifications

[5] R. Drath, A. Lüder, J. Peschke, and L. Hundt, "AutomationML – The Glue for Seamless Automation Engineering," in *Proceedings of the 13th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Hamburg, Germany, 2008, pp. 616–623.

[6] M. Grieves, "Digital Twin: Manufacturing Excellence through Virtual Factory Replication," in *White Paper*, 2014.

[7] H. Stachowiak, *Allgemeine Modelltheorie*. Wien/New York: Springer, 1973, iSBN 3-211-81106-0.

[8] International Organization for Standardization (ISO), "ISO 23247-1:2021 – Automation Systems and Integration – Digital Twin Framework for Manufacturing – Part 1: Overview and General Principles," ISO, Tech. Rep., 2021.

[9] T. Hedberg, Jr., A. B. Feeney, M. Helu, and J. A. Camelio, "Toward a Lifecycle Information Framework and Technology in Manufacturing," *Journal of Computing and Information Science in Engineering*, vol. 17, no. 2, p. 021010, 2017.

[10] IDTA, "Guideline: How to Create a Submodel Template Specification," IDTA, Frankfurt am Main, Germany, Tech. Rep., 2025. [Online]. Available: https://industrialdigitaltwin.org/content-hub/downloads

[11] C. Eckert and R. Hillerbrand, "Models in Engineering Design as Decision-Making Aids," *Engineering Studies*, vol. 14, no. 2, pp. 134–157, 2022.

[12] B. C. Kim, M. J. Pratt, R. G. Iyer, and R. D. Sriram, "Standardized Data Exchange of CAD Models with Design Intent," *Computer-Aided Design*, vol. 40, no. 7, pp. 760–777, 2008.

[13] Verein Deutscher Ingenieure (VDI), "VDI 2206 – Development of Cyber-Physical Mechatronic Systems," VDI, Tech. Rep., 2021.

[14] G. Shao, J. Hightower, and W. Schindel, "Credibility Consideration for Digital Twins in Manufacturing," *Manufacturing Letters*, vol. 35, pp. 24–28, 2023.

[15] Y. Wand and R. Y. Wang, "Anchoring Data Quality Dimensions in Ontological Foundations," *Communications of the ACM*, vol. 39, no. 11, pp. 86–95, 1996.

[16] H. Y. Teh, A. W. Kempa-Liehr, and K. I.-K. Wang, "Sensor Data Quality: A Systematic Review," *Journal of Big Data*, vol. 7, no. 1, p. 11, 2020.

[17] R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5–33, 1996.

[18] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for Data Quality Assessment and Improvement," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–52, 2009.

[19] International Organization for Standardization, "ISO/IEC 25012:2008 – Software Engineering – SQuaRE – Data Quality Model," ISO/IEC, Tech. Rep., 2008.

[20] U. Epple, "Merkmale als Grundlage der Interoperabilität technischer Systeme," *at – Automatisierungstechnik*, vol. 59, no. 7, pp. 440–450, 2011.

[21] U. Epple, M. Mertens, and C. Diedrich, "Using Properties as a Semantic Base for Interoperability," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3411–3419, 2017.

[22] W3C, "PROV-O: The PROV Ontology," World Wide Web Consortium, Tech. Rep., 2013.

[23] International Electrotechnical Commission, "IEC 62264-1:2013 – Enterprise-Control System Integration – Part 1: Models and Terminology," IEC, Tech. Rep., 2013.

[24] IDTA, "Specification of the asset administration shell: Part 3a – data specification iec 61360," IDTA, Frankfurt am Main, Germany, Tech. Rep., 2024. [Online]. Available: https://industrialdigitaltwin.org/content-hub/aasspecifications

[25] R. T. Fielding, M. Nottingham, and J. Reschke, “HTTP semantics,” Internet Engineering Task Force, RFC 9110, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[26] Institute for Control Systems (IRS), KIT, “Q-Metric: Role-Based Quality Assessment for Simulation Models,” https://www.irs.kit.edu/simulation-quality.php, 2024.

[27] IDTA, “Submodel Template Provision of Simulation Models (IDTA 02005-1-1),” IDTA, Frankfurt am Main, Germany, Tech. Rep., 3 2026. [Online]. Available: https://industrialdigitaltwin.org/content-hub/downloads