applied sciences

MDPI

Article

Query-Defined Submodels: Publishing Topological Graph Analyses to the Asset Administration Shell

Patrick Suwinski [1,*], Luca A. Linhsen [1], Hesham A. S. Salem [1], Annerose Braune [1] and Martin Wollschlaeger [2]

[1] Institute of Automation, Faculty of Electrical and Computer Engineering, TUD Dresden University of Technology, 01062 Dresden, Germany; luca-lin@outlook.de (L.A.L.); hesham_ahmed_saleh.salem@mailbox.tu-dresden.de (H.A.S.S.); annerose.braune@tu-dresden.de (A.B.)
[2] Chair of Industrial Communications, Institute of Applied Computer Science, Faculty of Computer Science, TUD Dresden University of Technology, 01187 Dresden, Germany; martin.wollschlaeger@tu-dresden.de
* Correspondence: patrick.suwinski@tu-dresden.de

Abstract

The Asset Administration Shell (AAS) standardises how industrial assets are described, but its query language does not support traversal of references between assets, so a client must evaluate patterns spanning several assets on data retrieved in bulk. Graph databases evaluate such patterns natively, yet their results reach only consumers with graph tooling. We publish graph analysis results as AAS submodels, composing an analysis query with an export function. A template records the query, the source revision and the validity, references the affected assets and marks derived values as computed. The submodel's extent follows from the query, not from a source document. We prove conformance, determinism and traceability, and that for a broad class of local queries the export cannot be inverted. An end-to-end implementation with Neo4j and Eclipse BaSyx measures the export side of a cost model. For 200 drives, the pipeline cuts latency 1.95-fold and volume 5.92-fold against a client filtering by value. A client that also trims, multiplexes and compresses reaches parity. Counting processing and transfer, the graph replica pays off above 0.32 reads per plant change if every tenth change triggers an export, and above 0.60 if every change does. The approach suits selective analyses read by several consumers.

Keywords: Asset Administration Shell; fault detection; labelled property graph; graph database; digital twin; submodel template; AAS Query Language; Industry 4.0; semantic interoperability

Check for updates

Academic Editor: Dan Sui

Received: 3 October 2026
Revised: 23 October 2026
Accepted: 26 October 2026
Published: 29 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Rising demands on the flexibility of industrial plants translate into a steadily growing number of technical components and denser interconnection between them, which surfaces as data flows and deep dependencies across system boundaries. That is the setting in which cyber-physical production systems [1] and the digital twin have become the organising concepts [2,3], the latter covering realisations [4] from passive digital models to cognitive twins that reason over their own representation [5].

For the interoperable representation of assets, the Industrial Digital Twin Association (IDTA) has published the specification of the Asset Administration Shell (AAS) [6], embedded in the Reference Architecture Model Industrie 4.0 (RAMI 4.0) [7]. The specification provides a metamodel, a set of Representational State Transfer (REST) Application

Programming Interfaces (APIs), and, since version 3.1 of Part 2, the AAS Query Language (AAS-QL) [8]. The AAS, the target format of this work, is the basis on which cross-organisational data exchange in Industry 4.0 is expected to take place.

1.1. Motivating Case Study

The case that runs through this paper is taken from industrial practice and serves as the evaluation frame. Continuous presses for the manufacture of wood-based panels reach lengths beyond 60 m [9]. The control architecture assumed here is a redundant programmable logic controller (PLC) system in tandem operation driving a large number of actuators, among them motors and hydraulic cylinders; the evaluation assumes 200 such drives per press, a count these installations reach and exceed. The drives are arranged in two rows to the left and right of a conveyor belt, each drive having an immediate neighbour before and behind it save at the ends. The drive count sets the scale of the evaluation rather than entering the cost model as a measured value, and Section 5.7 varies it by a factor of sixty-four for that reason.

Availability is secured by a topological redundancy scheme. The failure of an individual drive is offset by its neighbours, and the plant continues to operate. A functionally critical condition arises only under a topological pattern, namely when two adjacent drives fail simultaneously. Detecting such a condition therefore requires evaluating neighbourhood relations across asset boundaries, and this is where the standardised access path becomes expensive.

Two access patterns follow from this architecture, the first evaluated in Section 5 and the second in Section 6.5. The *horizontal* problem is the one just described, retrieving the adjacency relation and the fault state across the 200 drive shells, which are peers whose adjacency two controllers hold between them, and it is dominated by the volume of the topology. The *vertical* problem is retrieval along the containment hierarchy, from the plant through five production lines, two presses each, down to 200 drives per press carrying five sensors apiece, hence 10,000 sensor submodels at the leaves, and it is dominated by the number of leaves. The deployment assumed throughout stores the topology in Neo4j, a labelled property graph (LPG) database, and queries it in Cypher, the query language of that store.

Adjacency between two drives is expressed in the AAS by a `RelationshipElement` whose two `Reference` attributes address the participating shells. AAS-QL provides no mechanism for dereferencing such a reference, which Section 3.3 derives from the normative text rather than asserts. The evaluation therefore cannot be delegated to the server through standard AAS-QL: an assistance system has to retrieve the relationship structure, resolve the references itself and evaluate the adjacency predicate locally on every cycle, transferring a volume of data that grows with the size of the installation rather than with the size of the answer.

1.2. Research Question and Contribution

Graph data models are designed for precisely this class of problem. Their query languages provide variable-length path patterns, and with ISO/IEC 39075:2024 an LPG query language is internationally standardised [10]. Previous work has used graph models for digital twin construction [11,12] and for knowledge-based data spaces in adjacent domains [13]. What such approaches do not provide is a route by which their results reach systems that possess no graph tooling. A Cypher result is not an AAS artefact, and consuming it presupposes the very infrastructure that the standard was introduced to make unnecessary.

The purpose of moving such results into the AAS is easy to mistake. The architecture proposed here duplicates data twice over, maintaining a replicated graph and deriving a submodel from it, at a cost Section 5.6 quantifies; the duplication is instrumental, since what is offered for exchange is the exported submodel and not the replica, and the evaluation must answer whether the shared artefact repays the copy behind it. The aim is to express what a database holds in a format on which other parties have already agreed. Analysis results living in a graph database, or in a relational schema whose junction tables encode the same relations, are reachable only through an interface specific to that store, and the difficulty is not expressiveness but reach: a recursive common table expression answers the adjacency query perfectly well, yet the answer stays available only to a client holding credentials for that database, knowing its schema and agreeing with its owner on what the columns mean. Every further consumer needs an agreement of its own, and expressing the same content as an AAS submodel replaces that set of bilateral agreements with one shared reference.

This gains weight from the regulatory side, and two propositions must be kept apart. The first is settled law: Regulation (EU) 2024/1781 establishes the Digital Product Passport as a framework requirement, Article 9 providing that the information requirements adopted for a product group shall make market access conditional on a passport being available [14], and Regulation (EU) 2023/1542 fixes a date, requiring an electronic record for light means of transport batteries, industrial batteries above 2 kW h and electric-vehicle batteries from 18 February 2027 [15]. The second is not law. Both regulations reference the ISO/IEC 15459 [16] family for the data carrier and the identifier while expressly admitting alternatives: the Ecodesign Regulation until harmonised standards are published and the battery regulation through the phrase "or their equivalent", and both leave the data model of the passport content to delegated acts and harmonised standards; neither prescribes a metamodel. That the passport should be carried by the AAS is an industry proposal, worked out in detail in an IDTA submodel template for the passport metadata [17] and in a joint guideline of the IDTA, Catena-X and the BatteryPass consortium realising the battery passport as seven templates [18].

To the extent that this proposal prevails, the AAS becomes the format in which a large share of industrial product data arrives in the first place, and ingesting submodels conforming to such templates into a company's own database and publishing derived results back in the same format become two directions of one exchange rather than two unrelated conversions. The direction from the AAS into a graph is established practice, reviewed in Section 2. The opposite direction has been treated for the Resource Description Framework (RDF), where it re-expresses what the graph already holds for LPG sources only as the inverse of a prior ingestion or as a reference to a known fault (Sections 2.1 and 2.2). What is missing is the enriching form, in which the exported artefact carries the outcome of an analysis and not only a re-expression of the source.

This yields the question addressed here:

> Under which conditions can the result of a topological graph analysis be published as a standards-conformant AAS artefact such that a consumer requires no graph tooling, and under which operating conditions does this repay the cost of maintaining the replicated graph?

The paper makes four contributions.

1. **A proof that reference traversal is not expressible as an AAS-QL query condition.** That the language is not feature-comparable to the SPARQL Protocol and RDF Query Language (SPARQL) or the Graph Query Language (GQL) is stated in the specification

itself [8] [Part 2, Query Language, Limitations]; what is added here is a derivation. Section 3.3 states five premises drawn from the normative text of IDTA 01002-3-2 [8] and one locality premise about the evaluation semantics and derives that no single query can make reachability over the graph the RelationshipElements induce a condition on its result, together with the cost of the client-side evaluation that replaces it. What is inexpressible in one query is distinguished from what is infeasible in principle; the latter is not claimed.

2. **A formal account of what publishing a computed result entails.** The pipeline is characterised as the composition of an analysis query with an export function (Section 4.2). Two properties of the result graph carry the argument: the extent of the export is determined by the query rather than by a source document, and the exported model may carry derived properties not present in the source. Section 4.3 establishes conformance, determinism, traceability, and, for non-exhaustive bound-determined local queries, non-injectivity, so the pipeline is unidirectional; the mutable source is why Section 4.4 makes provenance mandatory.
3. **A submodel template with an explicit semantic strategy.** Sections 4.4 and 4.5 propose `TopologyAnalysisResult`, which separates analysis semantics from domain semantics, distinguishes observed from derived status along `semanticId`, so that the distinction is available as a query condition where a `Qualifier` is not (Section 3.3), marks every computed property in addition with the dictionary qualifier for calculated values, and, where no dictionary reference is available, mints a deterministic identifier from the source key and marks it as minted. Neither of the last two is new to the AAS. The qualifier is the one the IDTA template for time series data lists, and Sections 4.5 and 6.5 name the established concept for an element without resolved semantics, a reserved concept description. What the cascade adds is that the substitute keeps the source key recoverable rather than replacing it with a constant.
4. **An evaluation with an explicit epistemic status.** Sections 5 and 6.3 present a parameterised cost model whose export side is measured end to end against Neo4j and Eclipse BaSyx, classify every quantity as established, published, assumed or derived, compare against an optimised client, also with transport compression and composed optimisations, a graph-backed repository, a cache, a finer partition and a plant-side encoding, express the cost of the replicated model as a break-even ratio in resource time, and demonstrate consumability by a working consumer, which the end-to-end run also applies to the artefact read from the repository and by two negative controls.

Transformations from a graph into the AAS exist, and Section 2 reviews them; submodels publishing a computed value alongside its provenance exist too: the carbon footprint and predictive-maintenance templates of the IDTA [19,20] and the Data Administration Shell [21] do, and none is claimed here. What appears not to be covered is a submodel that has its *extent* fixed by an analysis query rather than by its template and that records, beside the computed content, the identity of the query and the revision of the mutable graph it was evaluated on (Section 4.4). The discipline itself is established elsewhere: recording the query and the state of the source is what the recommendations on citing evolving data prescribe [22], and one published AAS template already carries a query, though as a pointer to an external interface rather than as what fixes its own extent [23]. What Section 2 finds unoccupied is its encoding as a submodel template. The construction itself is not the claim, since, where the analysis is expressible in SPARQL, a `CONSTRUCT` query over RDF could materialise a computed result as readily as a Cypher one; the claim is the template and the provenance discipline a computed result requires, together with the formal reason for that requirement. Section 2 substantiates this against the seven lines of work identified there.

2. Related Work

The combination of the AAS with graph-based knowledge representation is neither new nor rare. Huebner and Ihlenfeldt [24] survey the field in a systematic mapping study covering 32 publications, of which 14 address AAS and knowledge graphs specifically, and identify four contribution topics, namely semantic foundations and formal mappings, automated model construction, knowledge-graph-driven operational applications, and distributed systems with vertical integration. Their study is used here as the frame of reference. The lines below are ordered by what they transform into.

2.1. Ontology-Based Representations of the AAS

The most established line renders the AAS in RDF and the Web Ontology Language (OWL). Grangel-González et al. [25] proposed representing the Asset Administration Shell with RDF, RDF Schema and OWL vocabularies so that shell content becomes dereferenceable and can be retrieved with SPARQL; Bader and Maleshkova [26] developed this into the Semantic Asset Administration Shell, and Bader et al. [27] published a knowledge graph of the Industry 4.0 standards and reference initiatives themselves.

More recent work moves from representation to application, giving an ontology-based modelling method for AAS content [28], targeting circular-economy scenarios [29], and reaching self-configurable industrial agents [30], hybrid graph matching of industrial models [31] and behaviour models for virtual commissioning [32]; two surveys record the resulting landscape and the wider use of ontologies in digital twins [33,34]. Common to the AAS representations in this line are the direction and location of the query: AAS content is lifted into a semantic-web representation and answers are obtained there, by SPARQL or a reasoner, which presupposes semantic-web tooling.

A related tradition outside the AAS must be set apart. Ontology-based data access (OBDA) exposes the content of a store through a shared target vocabulary so that consumers need no knowledge of the store, and declarative mappings specify that exposure [35,36]; the Semantic Asset Administration Shell maps by rules [26,37]. Replacing bilateral agreements with one shared reference is the motivation of that tradition, for which this work claims no novelty. What differs is what the mapping does, and two obvious formulations of the difference are wrong. An OBDA mapping is fixed at design time, so the extent of the exposure is fixed by the mapping assertions rather than by the client's question, but not by the schema, since a mapping assertion in the sense of Poggi et al. [35] [Section 3] pairs an arbitrary SQL query with a conjunctive query over the ontology. Nor is the answer assembled only from stored data, since certain-answer semantics returns tuples entailed by the terminology and stored under no queried predicate [35] [Section 2.3]. The difference is that whatever an OBDA answer adds is a logical consequence of a declared terminology and computable by rewriting, whereas here the extent follows from an analysis query and what the result adds is computed rather than entailed. The apparatus for comparing graph metamodels under such mappings, in particular information and semantics preservation, is adapted by Angles et al. [38] from the literature on direct mappings, which translate a database automatically and without input from the user, and applied there to three mappings from RDF into property graphs; Section 6.5 returns to both notions.

Three works in this line run in the direction this paper pursues. The Tractus-X SPARQL bridge exposes a knowledge graph behind an AAS server and registry interface, translating requests to the AAS API into SPARQL through a templating approach [39]. The AAS tree it serves is virtual and mirrors the graph, and the exposure re-expresses the graph and is recomputed per request rather than carrying a computed result, which places it with OBDA rather than with the present work. A prototype by Fraunhofer IWM transforms an RDF graph into an AAS instance through a SPARQL `CONSTRUCT` query followed by JavaScript

Object Notation (JSON) for Linked Data framing and validation against the AAS JSON schema [40]. There the query does fix the extent, which makes it the closest artefact to the construction of Section 4. Rongen et al. [41] give the peer-reviewed treatment of the same direction, a set of mapping rules that generate an AAS model from an RDF-based one. What all three share, and what separates them from the present work, is that what they serve re-expresses what the graph already holds. None of them produces properties the source does not contain, and none addresses what follows from that difference, namely the direction of the transformation and the provenance the result has to carry. Section 6.2 states the distinction in terms of Section 4.2, where the apparatus for it is available. The claim is about what these artefacts do rather than about what their construction could be made to do, since a SPARQL `CONSTRUCT` can, where the analysis is expressible in it, compute as readily as a Cypher one, so the difference is one of purpose and of what the exported model then has to carry, not one of expressive power.

2.2. Property-Graph Mirrors of the AAS

A second line stores AAS content and context in an LPG, trading formal entailment for traversal performance and schema flexibility.

Luxenburger et al. [42] describe a service infrastructure in which an AAS registry emits registration events and a Neo4j service maintains a structural mirror. Sonnenberg et al. [43] extend this substantially: fine-grained create, update and delete events from BaSyx services are streamed through Apache Kafka into Neo4j [44], the graph is kept synchronised, and a large language model (LLM) queries it through the Model Context Protocol. Rübel et al. [45] build a Neo4j fault-diagnosis graph outside the AAS; a follow-up mirrors AAS content into it and closes the loop by writing references to identified faults back into a submodel of the affected module [46]. Among the works reviewed in this line, that is the only one that returns an analysis result to the standard, so the distinction belongs here: the write-back is a per-task insertion of references into a submodel fixed in advance, with no template, no query-determined extent, no provenance or validity attributes, and content that is a reference to a known fault rather than a property computed by the analysis. Sapel et al. [47] publish a decision-support approach for injection moulding with its Neo4j implementation.

A distinct variant compiles rather than mirrors. The `neo4aas` library [48] converts between AAS and Neo4j in both directions, models the AAS class hierarchy as multiple node labels, checks specification constraints and compiles AAS-QL queries into Cypher. Its export direction is, at first sight, the one this paper pursues, but it is not: the exporter reconstructs a document from a subgraph its own importer previously wrote, so what it inverts is its own ingestion and its domain is the set of images of AAS documents rather than arbitrary property graphs. The library reports the export as partial and not fully restoring AAS-specific semantics. There is no submodel template, no strategy for assigning `semanticId` values to content that did not arrive with one, and no derived content, since a re-import has nothing to derive.

The compiler appears to remove the limitation that motivates this work and does not, for an instructive reason: the expressive power of a compiled query is bounded by its source language, and the operators AAS-QL offers are Boolean combinations of comparisons, string predicates, temporal accessors and casts (Section 3.3), none of which produces a path expression. Compiling into Cypher changes where a query is evaluated, not what it can express. The implementation corroborates the reading of the normative text: its documented operator set contains no traversal operator, and it lists among its limitations that `ModelReference` and `ExternalReference` targets are stored but neither resolved nor traversed automatically [48], in an implementation that compiles into a language whose

path patterns would express such a traversal. Section 5.5 tests the consequence quantitatively. The direction here is AAS to graph: the graph is the query engine, answers leave the standard, and only a partial exporter or a write-back returns them.

2.3. Transformation of Engineering Formats into the AAS

A third line transforms existing artefacts, mostly into AAS content. The transformation library of SAP [49] converts AutomationML into AAS; file- and engineering-artefact converters generate shells from vendor models and from engineering data logistics [50,51]; Miny et al. [52] transform one submodel into another under declared rules; and Eichelberger and Weber [53] generate submodel APIs from IDTA specifications, which produces code rather than AAS content. Large language models have been applied to the same task, generating AAS instances through agent pipelines [54]. These transformations, the code generator excepted, share the target of the present work but not its input: the format converters are structure-preserving, in that the output adds nothing the source lacks, and in no case does an analysis query fix the extent. Section 4.2 makes the difference precise.

Two neighbouring lines are recorded without being pursued. Knowledge graphs have been integrated with OPC Unified Architecture rather than the AAS by a formal mapping to the semantic web [55] and in the opposite direction [56]; Huebner and Ihlenfeldt [24] find that stream of comparable maturity to the AAS branch, at 14 publications each. Its concern is machine-level information modelling, and where a twin is modelled in the OPC UA information model its semantics are fixed by that model and by a Companion Specification rather than by the IDTA metamodel, so the submodel templates and the `semanticId` mechanism this work relies on take a different form there. Independently of either standard, graph representations are established for analysing production systems by pattern mining for twin creation [12], weighted dependency graphs [57] and twins derived from knowledge graphs [11]. That line supplies the analytical capability this work presupposes and does not address how a result reaches other systems in a standardised form, which is the question taken up here.

2.4. Positioning

Table 1 compares seven lines of work with this one by transformation direction, by what determines the extent of the export, by what the result carries beyond the source, and by what a consumer must be equipped with.

Table 1. Lines of work bordering on the production of AAS content. *Direction* names the transformation, *Extent set by* what fixes how much is transformed, *Derived content* what the result carries beyond the source, in the grades the text distinguishes: none, a reference only, entailed under a declared theory, generated by a language model, or computed by a declared analysis.

Line	Representative Work	Direction	Extent Set by	Derived Content	Consumer Needs
1	ontology-based AAS representations [26,28,29]	AAS → RDF/OWL	source document	entailed	triple store, SPARQL
2a	event-driven graph mirrors [42,43]	AAS → LPG	source document	none	graph tooling
2b	the `neo4aas` compiler [48]	AAS ↔ LPG	prior ingestion	none	AAS client or graph tooling
2c	diagnosis written back to the AAS [46]	LPG → AAS	one diagnosis task	reference only	AAS client
3a	engineering formats into the AAS [49,51]	format → AAS	mapping rules	none	AAS client
3b	language models generating AAS content [54]	text → AAS	prompt and source document	generated	AAS client
3c	submodel-to-submodel transformation [52]	AAS → AAS	transformation rules	none	AAS client
3d	API generation from specifications [53]	IDTA specification → code	specification document	none	programming toolchain

Table 1. *Cont.*

Line	Representative Work	Direction	Extent Set by	Derived Content	Consumer Needs
4a	OPC UA into ontologies [55]	OPC UA → OWL	source model	entailed	triple store, SPARQL
4b	OPC UA models from a graph [56]	knowledge graph → OPC UA	source graph	none	OPC UA client
5	graph analysis of production systems [57]	graph → query result	a query	computed	graph tooling
6a	RDF into the AAS by mapping [41]	RDF → AAS	mapping rules	none	AAS client
6b	RDF into the AAS by query [40]	RDF → AAS	a query	none	AAS client
7a	database-backed AAS facades [58]	database → AAS	a query	none	AAS client
7b	the Data Administration Shell [21]	dataset → AAS	pipeline metadata	computed	AAS client and the dataset
7c	the Tractus-X SPARQL bridge [39]	knowledge graph → AAS	a query	none	AAS client
7d	submodels publishing a computed value [19,20]	analysis → AAS	the template	computed	AAS client
7e	AAS submodels for condition monitoring [59]	—	device configuration	none	AAS client
8	**this work**	**LPG → AAS**	**a query**	**computed**	**AAS client**

Lines 1, 2a and 2b make the AAS queryable by graph means. The graph is the query engine and the AAS is the data source. The answer is produced outside the standard and, line 2b excepted, consumed outside it. Line 2c alone returns a diagnosis to the AAS, though as a reference to a known fault rather than as a computed value. Line 3a produces AAS content, but from static engineering artefacts, preserving structure, and line 3c transforms one submodel into another under declared rules, which is the closest published precedent for generating a submodel from existing submodel content [52]. Line 3b generates rather than computes, and what it generates arrives without an account of where it came from. Lines 6 and 7c run from a graph into the AAS as the present work does, but their output re-expresses what the graph holds rather than what an analysis over it concluded. Line 7d is the converse case and the nearest precedent for the content: the carbon footprint and predictive-maintenance templates [19,20] publish a computed value through the AAS beside an account of how it was obtained. Their extent is fixed by the template, so a consumer receives the same shape whatever the analysis found, and neither records the revision of the source. Line 7e structures condition monitoring and maintenance as submodels without carrying the prediction model or its output [59]. The *Extent set by* cell separates both from line 8; the revision of the source appears in no column and is taken up in Section 4.4.

Line 7a shares the extent criterion of this work but rejects the architecture adopted here. Van Leeuwen et al. [58] expose a relational database as an AAS by carrying out `SQL4AAS.ListQuery` and `SQL4AAS.ContentQuery` in AAS Qualifiers of a submodel template, so that a query and not a source document fixes what the exposed tree contains. Their stated principle is the opposite of the one adopted here: they argue for a proxy that holds no copy, "as this requires us to maintain and synchronize each copy of the data", and their proof of concept reports that periodic querying is infeasible at factory volumes, which is the same view-maintenance cost Section 5.6 prices rather than avoids. Two differences decide the architecture. A facade recomputes on each request, so what its proof of concept exposes is what the database holds and not what an analysis over it concluded; and its `Qualifier`-borne queries are read at request time rather than recorded with the answer, so the artefact carries no provenance. The choice between a facade and a materialised view is a classic one, and this work takes the second branch and states its price. Löcklin et al. [21] adapt the AAS to datasets, recording origin, performed analyses and their results while referencing the data themselves, which is the closest existing template

for analysis metadata, though its extent is fixed by the pipeline and not by a query; van Leeuwen et al. point from that template to a survey of database technologies [60].

The direction pursued here is the complement of lines 1, 2a and 2b and differs from line 3 in what determines the extent of the export. Graph analyses are made consumable *through* the AAS: the graph is the analysis engine, the AAS is the publication format, and the result re-enters the standard in a form requiring no graph tooling on the consumer side. In the classification of Huebner and Ihlenfeldt [24], this occupies a position at the margin of *automated model construction* that their sample does not cover. Two of their findings bear on the present work. Validation research (23 publications) outnumbers evaluation research (3), and the absence of quantifiable operational metrics is named as a central gap; this paper does not close it, since Section 5 presents an analytical cost model rather than a production measurement, but it addresses it explicitly and states the limits of what such a model supports. Their second finding, that architectures combining several standards remain scarce, is consistent with the two transformation directions being complementary, since a deployment may operate an event-driven mirror and an analysis export at the same time, and Section 6.2 explains why neither substitutes for the other.

3. Background and the Expressiveness Limit of AAS-QL

This section establishes the two formalisms the proposed pipeline connects, and derives the expressiveness limit of AAS-QL that motivates it. The limit is not stated as an opinion about the standard. It is derived from the normative text of IDTA 01002-3-2 [8].

3.1. Labeled Property Graphs

Definition 1 (Labeled Property Graph)**.** *Let $\mathcal{L}$ be a set of labels, $\mathcal{K}$ a finite set of property keys and $\mathcal{V}$ a set of values. An LPG is a five-tuple*

$$G := (N, E, \rho, \lambda, \sigma), \tag{1}$$

where N is a finite set of nodes, E a finite set of edges with $N \cap E = \emptyset$, $\rho : E \to N \times N$ assigns to each edge its ordered pair of incident nodes, $\lambda : N \cup E \to \mathcal{P}_{\text{fin}}(\mathcal{L})$ assigns a finite label set, and $\sigma : (N \cup E) \times \mathcal{K} \rightharpoonup \mathcal{V}$ is a partial function assigning property values.

Definition 1 adopts the five-tuple of Angles [61], with two deviations. First, Angles lets σ return a non-empty *set* of values, whereas Equation (1) restricts it to a single value per key. Second, Angles types λ as a partial function into non-empty label sets, whereas Equation (1) makes it total into the finite subsets $\mathcal{P}_{\text{fin}}(\mathcal{L})$. The two typings of λ are in bijection once the empty set is identified with undefinedness, so the second deviation is a reformulation rather than a generalisation, and it is narrowed again in Section 4.2, where a primary label has to be selected. Multi-valued properties play no role in the transformation developed below, and the restriction keeps the enrichment $\Delta\sigma$ of Section 4 single-valued and, under the standing assumption of Section 4.2, directly representable as an AAS `Property`. The underlying graph-theoretic foundations are treated in [62] and the wider notion of a knowledge graph in the survey of Hogan et al. [63]. Figure 1 shows such a graph for the press control of Section 1.1, namely a PLC controlling drives that are topologically adjacent to one another.

Property graphs are queried declaratively with Cypher [64], one of the languages that inspired the property-graph query language standardised as ISO/IEC 39075:2024 [10,65]. Both provide variable-length path patterns, i.e., syntax for expressing reachability over an unbounded number of edges within a single query. This capability is what the remainder of this section contrasts with AAS-QL.

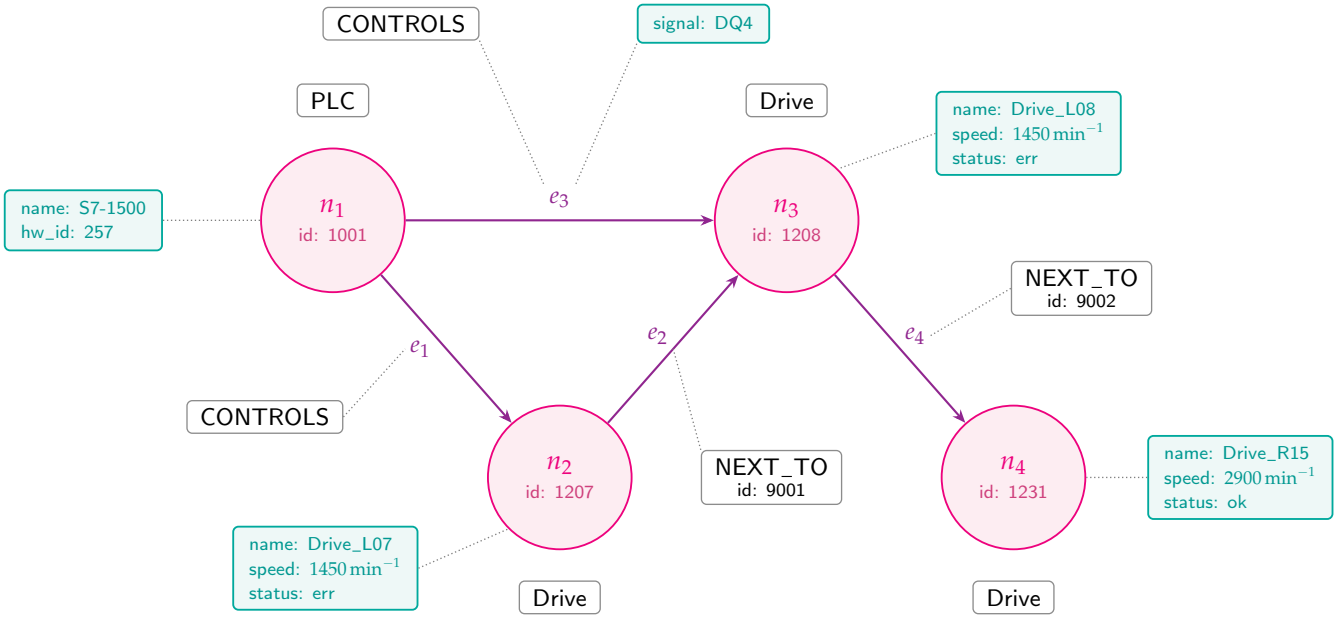


Figure 1. Labeled property graph of the press control. Nodes n_1 to n_4 (pink), edges e_1 to e_4 (violet), properties (teal) and labels (white). The adjacency relevant to the failure analysis is carried by the edges e_2 and e_4 of type `NEXT_TO`. Node n_4 is sound and enters the result graph not for its own state but as the endpoint of e_4, which is what the closure condition of Section 4.2 requires. Each drive node also carries the identifier of its shell, which rule R1 of Section 4.6 reads and the figure omits.

3.2. Asset Administration Shell

The AAS is the metamodel specified by the Industrial Digital Twin Association for the digital representation of industrial assets [6]. An `AssetAdministrationShell` is contained in an `Environment` together with the `Submodel` instances it references, each bundling one aspect of the asset. Submodels contain `SubmodelElement` instances, in particular `Property` for scalar data, `SubmodelElementCollection` (SMC) for nesting, and `RelationshipElement` for explicit links. A `RelationshipElement` carries two `Reference` attributes, `first` and `second`. A `Reference` addresses any Referable that can be identified, whether it lies in the same Submodel, in another Submodel of the same shell, in a different shell, or in a different environment altogether. A *model reference* resolves a chain of keys within an Identifiable, and an *external reference* names a target outside the metamodel, by an identifier that may itself be a concatenation. Semantic interoperability rests on the `semanticId` attribute, which points to a concept definition in an external dictionary such as the ECLASS product classification standard [66] or IEC 61360.

The set of RelationshipElements of an AAS environment therefore induces a directed graph over Referables. Whether this induced graph is accessible to the standardised query language is the question addressed next.

3.3. Expressiveness of the AAS Query Language

AAS-QL is specified in IDTA 01002-3-2 [8], which describes it as inspired by the Resource Query Language, a filter language over collections of resources rather than a language for navigating between them [8] [Part 2, Query Language]. The limit is therefore a design decision and not a deficiency. Five features of that specification are relevant here, together with one premise about how it is evaluated. All six are stated as premises: the first five so that each can be checked against the grammar and the stated limitations of the specification, and the sixth so that what the argument assumes beyond that text stands in one place and can be weighed.

Assumption 1 (Candidates)**.** *Limitation 1 of the subsection* Limitations *in [8] states that the language "focusses on Identifiables" and that "[i]t is only possible to formulate queries for Asset Administration Shells, Submodels, and their according Descriptors, as well as ConceptDescription objects . . . not every Referable can be queried" [8] [Part 2, Query Language, Limitations]. A query over an* `Environment` *therefore returns Shells, Submodels or ConceptDescriptions, though its conditions may read the SubmodelElements these contain.*

In the API a query is issued against one repository endpoint, so a single request ranges over the resources of one of these types. Taking their union is the more generous reading, and the impossibility established below holds under it.

Assumption 2 (Grammar)**.** *The production* `<query>` *of the context-free grammar given in [8] derives an optional select statement, a* `<logicalExpression>` *and an optional filter list.* `<logicalExpression>` *derives nested Boolean combinations (*`$and`*,* `$or`*,* `$not`*) of comparisons whose operands are field identifiers, literals of string, numeric, hexadecimal, Boolean, date-time or time type, casts, and the attribute forms* `CLAIM`*,* `GLOBAL` *and* `REFERENCE`*. It further derives* `$match`*, which is satisfied when one and the same entry of a list satisfies every condition under it. It also derives a Boolean literal and a cast of an operand to Boolean; that cast may itself carry a field identifier and is therefore bound by Assumption 6 below, like any other comparison. The comparison operators are* `$eq`*,* `$ne`*,* `$gt`*,* `$lt`*,* `$ge`*,* `$le`*, extended by* `$starts-with`*,* `$ends-with`*,* `$contains` *and* `$regex` *for strings, and the functions* `$dayOfWeek`*,* `$dayOfMonth`*,* `$month` *and* `$year` *yield numeric operands from temporal ones. The grammar provides* `<idShortPath>` *for navigation along the containment structure of a Submodel, but no production that traverses an edge of the graph induced by RelationshipElements, and none for variable-length patterns, transitive closure or fixpoint computation.*

Assumption 3 (Queryable fields)**.** *Limitation 4 of that subsection in [8] forbids traversing* `Entity/statements`*,* `AnnotatedRelationshipElement/annotations` *and the variables of* `Operation`*, and restricts the attributes of SubmodelElements available as query conditions to* `semanticId`*,* `supplementalSemanticIds`*,* `idShort`*,* `value`*,* `valueType` *and* `language`*. The* `first` *and* `second` *attributes of* `RelationshipElement` *are not among them, nor reachable indirectly through* `value`*. The grammar does admit comparisons over the components of a Reference, through* `<ReferenceClause>`*, which derives* `type`*,* `keys[i].type` *and* `keys[i].value`*, the index being optional and an omitted one matching some key, and is reachable from* `<FieldsAAS>`*,* `<SemanticIdClause>`*,* `<SupplementalSemanticIdClause>` *and* `<SpecificAssetIdsClause>`*, so* `$sme#semanticId.keys[0].value` *is well formed. It is not reachable from the* `value` *alternative of* `<FieldsSME>`*, a bare terminal admitting no continuation, so* `$sme#value.keys[0].value` *is not derivable. The syntactic observation is matched by a normative one: "[a]ttribute declarations point to literal values that provide the input for comparisons. It is not possible to point to objects or lists, only atomic values" [8] [Part 2, Query Language, Identification of Fields], and the value of a* `RelationshipElement` *is an object holding two References. Under the evaluation semantics a field identifier that does not resolve on a candidate "produces no comparison match" and excludes the candidate without error [8] [Part 2, Query Language, Query Evaluation Semantics]. The language therefore exposes the components of a Reference wherever a* queryable *attribute is Reference-valued, and does not for the value of a SubmodelElement.*

Assumption 4 (Scope of recursion)**.** *Field identifiers relevant to a repository query are rooted at* `$aas`*,* `$sm`*,* `$sme` *or* `$cd`*. The descriptor roots the grammar also provides do not arise over an* `Environment`*. From those four roots evaluation continues only along a closed list of navigation attributes, namely* `AssetAdministrationShell/submodels`*,* `Submodel/submodelElements` *and the* `value` *of* `SubmodelElementCollection` *and* `SubmodelElementList`*. The first of these*

is itself a `Reference`*, and the language does dereference it, which is why this premise has to be stated as a closed list rather than as a blanket denial that any Reference is followed. The recursive behaviour required for* `$sme#value` *descends into nested collections and lists along the same list. Beyond it, no production continues a field path from a* `Reference` *held in the* value *of a* `SubmodelElement`*, in particular not from* `RelationshipElement/first` *or* `/second`*. That evaluation reaches their targets by no other route is not a property of the grammar and is the content of Assumption 6.*

Assumption 5 (Granularity of the result)**.** *The production* `<selectStatement>` *derives exactly one form,* `$select id`*, declaring only "whether the response shall deviate from this default behaviour, and only return a list of identifiers instead of a list of objects" [8] [Part 2, Query Language, Select Expression]. Beyond this, the grammar offers* `$filters`*, a list of* `<QueryFilter>` *blocks each pairing a fragment with a* `<logicalExpression>`*, which "allows clients to reduce the amount of returned data" and can "never increase the amount of data or introduce information that is not already present in the underlying AAS" [8] [Part 2, Query Language, Query Filter]; a fragment ending in* `[]` *moreover "applies row-wise filtering to each array entry", so a filter can delete individual members of a collection and not only trim them. Neither half of a filter block reaches an endpoint: the condition is itself a* `<logicalExpression>` *and is bound by Assumptions 2–4, and no alternative of* `<FieldIdentifierFragment>` *admits* `first` *or* `second`*, its* `$sme` *alternative carrying the same six attributes as Assumption 3. Filtering can remove what the analysis does not read; it cannot reduce the adjacency to the part the analysis needs.*

Assumption 5 is worth separating from Assumptions 2–4 because it constrains something different. Assumptions 2–4 bound which *conditions* a query may express. Assumption 5 bounds what a query may *return*, and the two interact in a way that matters below. A condition on `$sme` narrows which Submodels match. A filter narrows what each match carries. Neither can make the number of adjacency elements transferred depend on which of them the analysis is about, because that would require a condition over an endpoint. Section 5.4 reports what maximal filtering achieves and what it leaves untouched.

The specification itself frames these restrictions explicitly. The subsection *Limitations* opens with the statement that the AAS Query Language "is not intended as feature-comparable to existing query languages like SPARQL or GQL" and refers implementers to vendor-specific extensions where such expressiveness is required [8] [Part 2, Query Language, Limitations].

The five premises above, together with the locality premise stated below, settle the question raised at the end of Section 3.2. If no field identifier reaches the endpoints of a `RelationshipElement`, and if no production of the grammar dereferences those endpoint References in order to continue evaluation at their targets, then the graph induced on the Referables lies outside what a query can traverse. The argument is made in two steps, an invariance lemma and the statement it yields.

One convention is needed first. A query is said to *decide* a predicate φ over a class $\mathfrak{E}$ of AAS environments if there is one request context in which, for every $\mathcal{E} \in \mathfrak{E}$, the result of evaluating the query on $\mathcal{E}$ is non-empty exactly when φ holds in $\mathcal{E}$. Quantifying over a class rather than over one fixed environment is what makes the statement non-trivial, since for a single environment the answer is a constant and any query returning the right emptiness decides it trivially.

Assumption 6 (Locality of evaluation)**.** *Whether a candidate satisfies a* `<logicalExpression>` *depends only on the* atomic *values to which the productions* `<FieldsAAS>`*,* `<FieldsSM>`*,* `<FieldsSME>` *and* `<FieldsCD>` *resolve, at the candidate itself, at the Referables it contains and at those containing it along the navigation attributes of Assumption 4, at the ConceptDescriptions their* `semanticId`*s name, for which the specification states no narrowing and which are admitted here*

generously, and, through the `REFERENCE` *attribute form, at Referables named by a literal identifier. A field identifier that resolves to an object rather than to an atomic value contributes no comparison match. Through* `CLAIM` *and* `GLOBAL` *the outcome may in addition depend on the clock and on the security token of the request, neither of which is part of the environment. In no case does it depend on data of the environment outside those four productions.*

This premise is semantic rather than syntactic, which is why it is stated apart from the five that record properties of the normative text. Its `REFERENCE` clause is the exception and needs no semantic warrant: `<ReferenceAttribute>` pairs the literal identifier with a field path drawn from those same four productions, none of which admits `first` or `second` as a field, so the grammar alone bounds what the form reads of the instance it names, and a literal introduces neither a variable nor a parametric number of steps. The clause is recorded because a blanket premise would be false, not because it is assumed. What the premise adds beyond the grammar is two sentences. That a field identifier resolving to an object contributes no match is assumed rather than derived: the specification states that attribute declarations "point to literal values" and not "to objects or lists" [8] [Part 2, Query Language, Identification of Fields], and its evaluation clause fixes what happens when a field fails to resolve *because it is absent*, but neither settles a field that is present and holds an object. The other is the last sentence, that no further locus of dependence exists. Both are consistent with the normative evaluation clause [8] [Part 2, Query Language, Query Evaluation Semantics], which fixes what happens when a field identifier does not resolve on a candidate but not the locus of dependence. Without a premise of this kind, nothing would follow from the grammar at all, since the absence of a production is a statement about syntax, whereas inexpressibility is one about what the evaluation can distinguish.

Lemma 1 (Endpoint invariance)**.** *Let $\mathcal{E}$ and $\mathcal{E}'$ be AAS environments that hold the same Referables with the same containment and navigation structure and that differ only in the values of* `first` *and* `second` *of one or more RelationshipElements, so that they agree, for every Referable including those elements, on every atomic value any of the productions* `<FieldsAAS>`, `<FieldsSM>`, `<FieldsSME>` *and* `<FieldsCD>` *resolves to. Then under Assumptions 1–6, every AAS-QL query evaluated in one request context has the same set of matching candidates on $\mathcal{E}$ and on $\mathcal{E}'$. In particular its result is non-empty on $\mathcal{E}$ exactly when it is non-empty on $\mathcal{E}'$.*

Naming the four productions rather than the six attributes of Assumption 3 is not cosmetic: the latter are attributes of SubmodelElements, while a query may equally compare a field of a Shell such as `assetInformation.globalAssetId`. The hypothesis that the two environments differ in `first` and `second` and the hypothesis that they agree on every atomic value are consistent, because the pair of References is the `value` of a `RelationshipElement` and not an atomic value.

Proof. By Assumption 2 a query is an optional select statement, a `<logicalExpression>` and an optional filter list, and by Assumption 1 the candidates it ranges over are the Shells, Submodels and ConceptDescriptions of the environment, which are the same in both. By Assumption 3 the attributes in which the two environments differ are not among the queryable ones, and neither is any attribute derived from them, since the `value` alternative of `<FieldsSME>` admits no `<ReferenceClause>` continuation; the one queryable field that names them at all, the bare `value` of the `RelationshipElement`, denotes an object rather than an atomic value and by Assumption 6 contributes no comparison match, alike on both environments. By Assumption 4 evaluation descends only along its navigation attributes, which also agree. By Assumption 6, the truth of the `<logicalExpression>` at a candidate is thus determined by data on which the two environments agree, so the sets of matching

candidates coincide. The select statement changes only how a match is represented. A filter block removes parts of a matching object but, by Assumption 5, admits no condition that can tell the two apart, so the number of objects returned is the same. □

The lemma is about which candidates match and deliberately not about what is transferred. The two are different, and the difference is the subject of Observation 1 below. Under Assumption 5 a matching Submodel is returned with the endpoints in it, since no filter is obliged to remove them and the analysis needs them. The endpoints therefore cross the network. What no query can do is make them a condition.

Proposition 1 (Reference traversal is not expressible as a query condition)**.** *For every $k \geq 1$ there exist a class $\mathfrak{E}$ of AAS environments and a pair u, v of Referables such that reachability of v from u within k edges of the graph $R(\mathcal{E})$ induced on the Referables by the RelationshipElements of $\mathcal{E}$ differs across $\mathfrak{E}$, yet under Assumptions 1–6, no single AAS-QL query decides it over $\mathfrak{E}$.*

Proof. For $k = 1$ take $\mathcal{E}_1$ whose only RelationshipElement is x, with `first` $= u$ and `second` $= v$, and $\mathcal{E}_2$ identical except that x has `second` $= w$ for some third Referable w. The two environments hold the same Referables and agree on every atomic value the four productions of Lemma 1 resolve to, including at x itself, and differ only in an endpoint. The predicate holds in $\mathcal{E}_1$ and fails in $\mathcal{E}_2$, whereas by Lemma 1 every query has the same set of matching candidates on both, hence the same emptiness of the result. No query therefore decides it.

For $k \geq 2$, the same construction applies to the last edge of a chain $u \to z_1 \to \cdots \to z_{k-1} \to v$, where $\mathcal{E}_1$ holds no RelationshipElement other than those of the chain, so that no alternative path of length at most k exists. Redirecting the endpoint of the final RelationshipElement again produces two environments differing only in an endpoint, and Lemma 1 applies unchanged. Nothing in a query's `<logicalExpression>` compensates: by Assumption 6, its truth depends only on the loci that assumption admits, none of which is reached by following an endpoint Reference, and by Assumption 2, the grammar has no variable-length pattern, transitive closure or fixpoint. No query therefore decides the predicate for $k \geq 2$ either. □

One qualification belongs with the proposition: it concerns the language, not every conceivable deployment. A producer that encoded the endpoints of an edge into its `idShort`, or that carried a neighbour identifier as the atomic value of a `Property` alongside the relationship, would let a client select edges by a string predicate or by `$sme#value`. Neither encoding violates the metamodel. What both require is that the consumer know which attribute holds an endpoint and how it is spelled, which the specification nowhere states. A registered submodel template would state it, and until one exists, the convention is bilateral, as the template of Section 4.4 is until it is submitted. A third encoding goes further and stores no relationship at all: a precomputed component label, one value per node, lets one condition decide undirected connectivity from a known label, though no bounded distance. It does so by publishing a derived property as a value, which is the construction of Section 4 performed by the plant, fixed to one question when the plant is modelled and carrying neither the query it answers nor the revision it was computed on. The proposition states what AAS-QL can express of a `RelationshipElement`, and an encoding convention does not extend it.

Proposition 1 must also not be read as a claim of general impossibility, and the distinction matters for the argument developed in this paper. Figure 2 shows what the interface leaves a client to do.

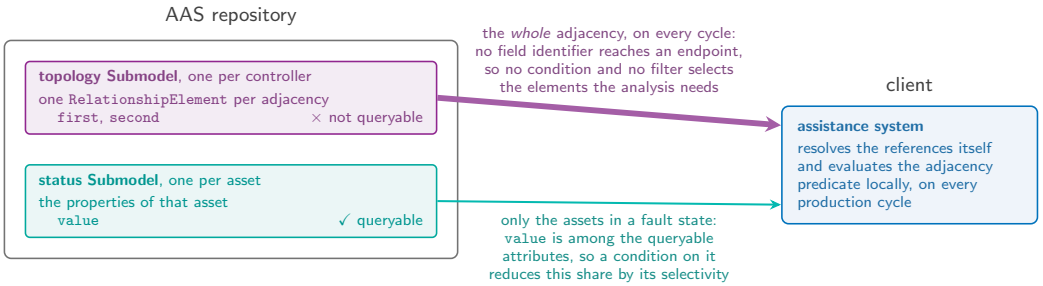


Figure 2. What the standardised interface obliges a client to transfer, and what a condition can remove. Assumption 3 admits `value` as a query condition, so the portion carrying asset state shrinks with the fault rate. It admits neither `first` nor `second`, and no fragment of the filter list of Assumption 5 reaches them either, so no condition selects the adjacency elements this class of analysis needs and that portion crosses the network in full on every cycle. Observation 1 names the three cost terms this produces and Section 5.4 reports what share of the baseline payload the untouched portion is. The width of each arrow indicates the volume of the portion it carries, not to scale.

Observation 1 (Client-side evaluation and its cost)**.** *Let n be the number of assets whose adjacency is described, b the branching factor of the hierarchy above them, d its depth and p the page size. A k-hop neighbourhood predicate over $R(\mathcal{E})$ remains* evaluable *by a client in either of two ways. It may dereference hop by hop, forming round $i+1$ from the References returned by round i, at up to k dependent rounds, transferring the Submodels the visited neighbourhood lies in; or it may retrieve the adjacency in bulk and evaluate all k hops locally, at $O(d + n/p)$ dependent rounds, transferring the whole of it. Which is cheaper is a property of the partition rather than of k: where the adjacency of many assets is held in few Submodels, one hop already retrieves a whole topology Submodel and the hop-by-hop strategy degenerates into the bulk one. A partition fine enough to make hop-by-hop cheap is a change to the plant model, costed in Section 5.8. The bulk strategy is the one costed below, and three quantities characterise bulk cost and respond differently to optimisation.*

1. ***Dependent rounds.*** *A bulk client that resolves the hierarchy level by level needs $O(n/p)$ rounds in a flat topology and $O(d + n/p)$ in a hierarchy. This is the term most easily removed. A condition on `$sm#semanticId` identifies the targets directly wherever they are semantically distinguishable, leaving the $\lceil n/p \rceil$ rounds a server-generated page cursor to keep sequential. What this removes is the hierarchical descent and not the hop iteration, since the targets of a hop are determined by endpoint values, and those are not queryable.*
2. ***Requests.*** *Retrieving one Referable per request costs n requests, plus $\sum_{i<d} b^i$ for the levels above them. Pagination reduces their sum to $\Theta(d + n/p)$. The gain is a constant factor, and it is large.*
3. ***Transferred volume.*** *This is the term that resists. A condition on a queryable attribute reduces the volume in proportion to its selectivity, so the portion carrying asset* state *shrinks. The portion carrying the* topology *does not, because by Assumption 3 no condition and no filter selects the relevant RelationshipElements. Filtering under Assumption 5 reduces what each adjacency element carries by a constant factor; row-wise filtering can also delete array members, but no condition it admits addresses an endpoint, so it cannot select the elements this class of analysis needs and cannot reduce their number. At a fixed partition and bounded degree it therefore remains $\Theta(n)$. The qualification matters: re-partitioning the plant model so that the number of Submodels grows with n does reduce the volume, and Section 5.8 shows by how much and why it does not carry over to the round structure.*

What is not expressible in a *single* query is therefore not infeasible in principle. It is relocated into the client and paid for in network traffic, and Section 5 models a client that paginates, parallelises and filters on values so that what remains is the third term rather than an artefact of a weak baseline. Two consequences shape the remainder of this

paper. The limit resides in the language and not in the persistence layer, since replacing the storage back-end leaves Assumptions 2–4 untouched, and an AAS-QL-to-Cypher compiler such as [48] inherits the expressiveness of its source language. Since an analysis that must traverse references cannot be delegated through AAS-QL, it must be performed elsewhere, and its *result* returned to the standard, which is the transformation direction developed next.

4. Materials and Methods

4.1. Overview and Design Rationale

Section 3 established that topological patterns spanning several assets held as `RelationshipElements` cannot be evaluated within a single AAS-QL query (Proposition 1) and that the resulting client-side evaluation scales with the size of the installation (Observation 1). The approach proposed here does not attempt to extend the query language. It relocates the analysis to a system whose query language does provide path patterns and returns the *result* of that analysis to the AAS as a standards-conformant artefact.

This is a deliberate architectural separation between the system that computes and the system that publishes, and it places the approach in an established class of designs, that of a derived, read-only representation maintained alongside an authoritative source, known in database theory as a materialised view [67,68] and in software architecture as a read model [69] [Chapter 12]. Naming the class matters, because it determines which claims the approach may legitimately make and which it may not. The questions that do not follow definitionally, namely what is materialised, in which form, at what cost, and from which point of operation the cost is recovered, are addressed in Sections 4.2–4.7 and 5.

4.2. Formal Framework

Let $G = (N, E, \rho, \lambda, \sigma)$ be an LPG according to Definition 1, representing the topology of the production system under consideration.

Definition 2 (Analysis query and result graph)**.** *An analysis query is a map q that assigns to G a* result graph

$$G_q := q(G) = (N_q, E_q, \rho_q, \lambda_q, \sigma_q, \Delta\sigma), \tag{2}$$

with $N_q \subseteq N$, $E_q \subseteq E$, $\rho_q = \rho|_{E_q}$ and $\lambda_q = \lambda|_{N_q \cup E_q}$, and with E_q closed under its endpoints,

$$\rho(e) \in N_q \times N_q \quad \text{for all } e \in E_q, \tag{3}$$

so that the first five components of G_q form an LPG in the sense of Definition 1. Partial functions are identified with their graphs, i.e., with the set of argument-value pairs, so that containment and difference are the ordinary set-theoretic operations. On $(N_q \cup E_q) \times \mathcal{K}$ the query may extend the property assignment but not contradict it, $\sigma_q \supseteq \sigma|_{(N_q \cup E_q) \times \mathcal{K}}$, and the set of derived properties *is*

$$\Delta\sigma := \sigma_q \setminus \sigma|_{(N_q \cup E_q) \times \mathcal{K}}. \tag{4}$$

The sixth component is not redundant. Rules R3 and R4 of Section 4.6 treat a property differently according to which of the two sets its key-value pair lies in, and σ is not recoverable from σ_q alone, so the split has to be carried rather than recomputed.

Two properties of the result graph (2) carry the argument of this paper.

Selectivity. Definition 2 permits $N_q = N$ and so does not itself bound the size of the result; selectivity is an assumption about the queries of interest and is stated as such. In the intended use $|N_q| \ll |N|$, because the analysis query returns the subgraph matching a topological pattern and not the topology. What follows from the definition is the weaker

and sufficient statement that the extent of the export is governed by the size of G_q, which the export emits as one document, rather than by the size of the source graph. The ratio $s := |N_q|/|N|$ is termed the selectivity; Section 5 holds it constant while $|N|$ varies.

Enrichment. Definition 2 admits $\Delta\sigma = \emptyset$, so that $\Delta\sigma \neq \emptyset$ in Equation (4) is an assumption about the queries of interest and not a consequence, exactly as selectivity is. Derived properties are values produced by the pattern match rather than consumed by it. In the case study an edge of type `NEXT_TO` whose two incident drives are both failed receives `assessment` = `Critical_Adjacency_Failure`, and one with a single failed endpoint receives `assessment` = `Non_Critical`; no such property exists in G. A structure-preserving conversion satisfies $\sigma_q = \sigma|_{(N_q \cup E_q) \times \mathcal{K}}$ by definition, so $\Delta\sigma$ distinguishes the present approach from that family formally rather than rhetorically.

Standing assumptions. Every node is one logical asset. A total injective id : $N \cup E \to \mathcal{I}$ maps every node and edge into the set $\mathcal{I}$ of identifiers the source system maintains, and a total $\ell : N \cup E \to \mathcal{L}$ with $\ell(x) \in \lambda(x)$ selects one label as primary, taking the least under a fixed total order on $\mathcal{L}$ where $\lambda(x)$ holds more than one, so the choice does not depend on iteration order. For ℓ to be total, $\lambda(x)$ must be non-empty and finite, which holds of any graph a production system maintains. A further standing assumption is needed by rules R1 to R4: every label is a string, and every value in the image of σ_q, and hence of $\Delta\sigma$ by Equation (4), is scalar and carries a type for which the metamodel defines a `DataTypeDefXsd`, fixed by a map ty supplied with the configuration, and serialises into the lexical space of that type using only the characters AASd-130 admits. Definition 1 leaves $\mathcal{V}$ unstructured, and this assumption keeps it so: a property holding a list or a nested record lies outside the rules below; decomposing one into a subordinate collection would need an inductive $\mathcal{V}$ and a uniqueness argument per level that are not given here. Serialisation of N_q and E_q follows a fixed total order on $\mathcal{I}$, a total on those sets because id is injective, and fixed orders on $\mathcal{K}$ and $\mathcal{L}$ govern the properties and the labels within each collection, so repeated runs are byte-identical.

Definition 3 (Export function and pipeline)**.** *Let $\mathbb{G}$ denote the set of LPGs whose nodes and edges are drawn from a fixed universe and which satisfy the standing assumptions of Section 4.2 on* id *and ℓ, write $\mathbb{G}_q$ for those result graphs that also satisfy the one on* ty*, and let $q : \mathbb{G} \to \mathbb{G}_q$ be an analysis query, meaning a map for which $q(G)$ is a result graph of G in the sense of Definition 2 for every G. Let $\mathbb{S}$ denote the set of submodels that conform to the metamodel [6]* and *whose embedding in an `Environment` validates against the file schema, a distinction Section 4.7 shows to be non-vacuous. Besides the result graph, the construction consumes three further inputs that G_q does not contain, namely the query text, the revision of the source graph and the wall clock, so the* export function *is typed*

$$\tau_\nu : \mathbb{G}_q \times \mathcal{Q} \times \mathcal{R} \times \mathcal{T} \to \mathbb{S}, \tag{5}$$

where $Q \in \mathcal{Q}$ is the text whose denotation is the analysis query, so that $q = [\![Q]\!]$, $\mathcal{R}$ is the set of source revisions and $\mathcal{T}$ is the set of timestamps. The first argument is a result graph and therefore carries $\Delta\sigma$, which rules R3 and R4 require and which σ_q alone does not determine. Further values are fixed by the pipeline configuration rather than supplied per run and are collected in the subscript ν, whose Uniform Resource Identifier (URI) namespace component is ν_0. They are the namespace ν_0, the query identifier j and the validity horizon δ of rule R0, the type map ty*, the maps* id *and ℓ, the maps of Lemma 2 and the sanitiser of rule R3, the two partial injections of rules R3 and R4, and the cascade of Section 4.5. The identifier of the generated submodel is derived from ν_0 and j and introduces no further input. Property 2 below is a statement about the arguments* and *this*

configuration, both of which must be held fixed for it to say anything. Fixing the three per-run arguments yields the export pipeline

$$P_\nu^{Q,r,t} := \tau_\nu(\cdot, Q, r, t) \circ q \;:\; \mathbb{G} \to \mathbb{S}. \tag{6}$$

Fixing them is the only reading under which the pipeline is a map from source graphs to documents at all. Where the auxiliary arguments are immaterial, the superscript is dropped.

The composition (6) is what this paper calls *query-defined reification*. The second word carries a different and fixed sense in the semantic-web literature, where it names the representation of a statement as a resource; the sense intended here is the older and more general one. The two words carry the two halves of the contribution: *reification* because a transient query result is given the standing of a first-class artefact, and *query-defined* because its extent is decided by a question rather than by the boundary of a source document. Figure 3 sets out the composition, the inputs it consumes and what it does not admit.

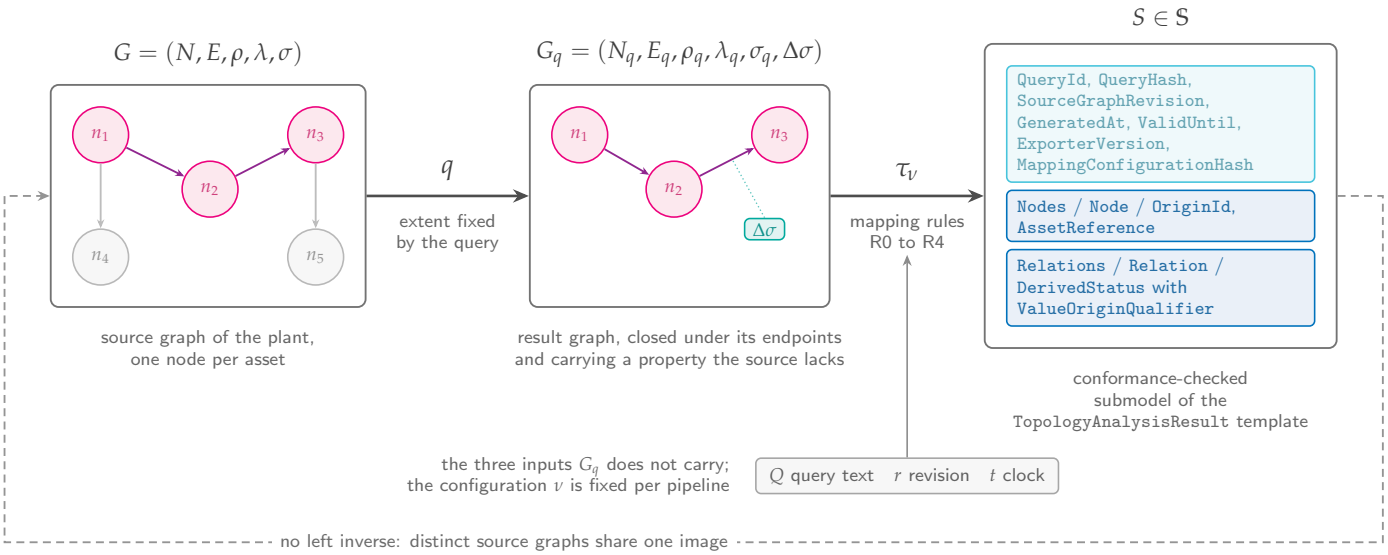


Figure 3. The pipeline of Definition 3. The analysis query q returns the subgraph a topological pattern matches, so the extent of the export follows from the query rather than from the boundary of a source document, and it may attach values the source graph does not hold, collected in $\Delta\sigma$ by Equation (4). The export function τ_ν consumes three further inputs that the result graph does not carry, which is why Equation (5) types it as it does, and realises the mapping in rules R0 to R4 of Section 4.6. The grey nodes lie outside N_q; deleting one leaves the image unchanged, which is the construction Proposition 2 turns into the absence of a left inverse.

The realisation of τ_ν is given by the mapping rules of Section 4.6, which are subordinate to Definition 2. They determine *how* the result graph is represented, whereas q determines *what* is represented at all.

Scope of applicability. The pipeline is formulated for labelled property graphs, but what it requires are properties of the result graph and of the analysis query rather than of a particular store or query language. Properties 1–3 of Section 4.3 depend only on the result graph and the configuration ν, and Proposition 2 in addition on the properties of the query on the source graph that it names. None of them depends on the query language. Table 2 lists the graph models that meet these conditions and what each needs first.

Table 2. Graph models to which the pipeline applies. The conditions are those of Definition 1 and the standing assumptions of this section: one node stands for one asset, every element carries a total injective identifier and a primary label, and properties take scalar typed values.

Graph Model	Condition	Status
Labeled property graph queried with Cypher or GQL	the standing assumptions	direct
Property graph with multi-valued or nested properties	properties decomposed into scalar values before the export	after decomposition
RDF or OWL knowledge graph	a mapping into a property graph [38], or the analysis written as a SPARQL `CONSTRUCT` query, IRIs as identifiers, edge properties through RDF-star or reification	after mapping
Relational schema with junction tables	a recursive query that returns stable keys, read as a result graph	after mapping
Hypergraph or n-ary relation	the relation reified as a node, since a `RelationshipElement` is binary	after reification
One asset spread over several nodes	outside the rules, see Section 6.5	not covered

4.3. Properties of the Pipeline

Three properties and one proposition characterise the pipeline of Definition 3.

Property 1 (Conformance)**.** *For every input that satisfies the standing assumptions of Section 4.2 that bear on the rules, namely a total injective identifier, a non-empty finite label set of strings for every element, and for every property a scalar value serialising into the lexical space of the type* ty *gives it, using only the characters AASd-130 admits, and for every configuration satisfying the hypotheses of Lemma 2 and rules R3 and R4 and under which the shell identifier or* `globalAssetId` *that rule R1 reads lies inside* `Identifier` *for every node of the result graph, and for a namespace* ν_0 *and identifier* j *whose derived identifiers stay inside* `Identifier` *and whose three sub-namespaces, those of the origin identifiers, of the generated submodel and of the fallback concepts, are pairwise disjoint, the rules of Section 4.6 construct a document that lies in* $\mathbb{S}$*. Writing* $\hat{\tau}$ *for the construction before its codomain is asserted and* D *for the set of admissible inputs, the claim is* $\mathrm{im}(\hat{\tau}|_D) \subseteq \mathbb{S}$*, where* im *denotes the image of a function, that is, the set of values it actually takes. Equation* (5) *records the conclusion, and the property is what has to be established for that typing to be legitimate.*

AASd-022, which makes `idShort` unique among non-identifiable siblings, is guaranteed by construction, and the guarantee has to be stated precisely because the obvious argument fails: rule R1 derives the identifier from the primary label *and* the source identifier, and concatenation $\|$ is not injective, since ‘‘`A_B`’’ $\|$ ‘‘`_C`’’ and ‘‘`A`’’ $\|$ ‘‘`_B_C`’’ coincide.

Lemma 2 (Uniqueness of the generated identifiers)**.** *Let* Σ *be the letters, digits and underscore, which AASd-002 admits in every published version, let* $\mathrm{enc} : \mathcal{L} \cup \mathcal{I} \to \Sigma^*$ *be injective with no underscore in its image, let* san *map labels into* Σ^* *so that* $\mathrm{san}(a)$ *is non-empty and begins with a letter, and let* $|\mathrm{san}(\ell(x))| + 1 + |\mathrm{enc}(\mathrm{id}(x))| \leq 128$*, the length* `NameType` *admits, for every element* x*. Then Equation* (7) *below is injective on any set of elements with pairwise distinct identifiers, and every value satisfies AASd-002.*

Proof. Since enc produces no underscore, the substring of $\mathrm{san}(\ell(x)) \,\|\, \text{‘‘_’’} \,\|\, \mathrm{enc}(\mathrm{id}(x))$ following its last underscore is exactly $\mathrm{enc}(\mathrm{id}(x))$. Two elements with the same generated identifier therefore agree in $\mathrm{enc}(\mathrm{id}(x))$, hence in $\mathrm{id}(x)$ by injectivity of enc, hence they coincide. Injectivity of san is not required. For AASd-002, the value begins with the first character of $\mathrm{san}(\ell(x))$, which is a letter by hypothesis, and draws every further character from Σ. It is at least two and at most 128 characters long. □

The escape is required: a Neo4j `elementId` has the form `4:uuid:12`, and AASd-002 admits no colon, so inserting the raw identifier would break conformance. The other constraints relevant to the generated structure, AASd-117, AASd-118 and AASd-121 to AASd-128, and AASd-107 to AASd-109 in the second stage, follow from the rules by inspection and are also checked (Section 4.7).

Property 2 (Determinism)**.** *Of the two claims, the second is the one usually wanted. First, $\hat{\tau}$ depends on its arguments and on ν alone. Equal arguments yield byte-identical documents, because the mapping rules depend on no iteration order and generate no random or clock-derived identifier, and because N_q and E_q, their properties and their labels are serialised in the orders Section 4.2 fixes on $\mathcal{I}$, $\mathcal{K}$ and $\mathcal{L}$. Second, τ_ν is invariant in its timestamp argument up to the two provenance attributes that record it. For $t, t' \in \mathcal{T}$ the documents $\tau_\nu(G_q, Q, r, t)$ and $\tau_\nu(G_q, Q, r, t')$ are identical except in the values of `GeneratedAt` and `ValidUntil`.*

Both claims follow from the construction of the mapping rules, and together they make the artefact reproducible: two executions over the same result graph differ only in the two timestamp attributes, so exporter output can be compared byte for byte once those are masked.

Property 3 (Traceability)**.** *The mapping rules induce an element-level map τ_ν^{el} from $N_q \cup E_q$ into the direct members of the collections `Nodes` and `Relations` of $S := \tau_\nu(G_q, Q, r, t)$, where those collections are present. It is a bijection onto those direct members, and its inverse is recoverable from S, the namespace ν_0 and the maps* enc *and* id *of ν through the `OriginId` property.*

Injectivity is Lemma 2; surjectivity holds because the rules create no member without a pre-image, and the closure (3) makes the references between members resolvable within S. Traceability is realised by the `OriginId` property of rule R1 rather than by reusing internal database identifiers as AAS identifiers, which would violate the requirement for globally unique identifiers. Rule R1 extends the trace past the source graph to the asset itself: every member of `Nodes` carries an `AssetReference` that a consumer resolves against the repository without knowing any convention of the producing system. An `OriginId` identifies its element in the revision that `SourceGraphRevision` records, since Neo4j guarantees the mapping between an `elementId` and its element only within one transaction [70].

Proposition 2 (Non-injectivity)**.** *Call q* bound-determined *on G if for every subgraph G' of G containing $N_q \cup E_q$ the query returns on G' exactly the elements it returns on G, and* local *if $\Delta\sigma$ is a function of the returned elements with their labels and properties. A positive non-projecting subgraph pattern match, one whose patterns carry no negation and no aggregation and no match that binds an element it does not return, is bound-determined if no pattern requires a witness outside $N_q \cup E_q$. Call q* non-exhaustive *on G if $N_q \subsetneq N$. If q is bound-determined, local and non-exhaustive on some G, then $P_\nu^{Q,r,t}$ is not injective for any fixed (Q, r, t).*

Proof. Since $N_q \subsetneq N$ there exists $w \in N \setminus N_q$. Let G' be obtained from G by deleting w together with its incident edges. By the closure (3), E_q contains no edge incident to w, so G' still contains $N_q \cup E_q$ and by bound-determinedness, the query returns the same elements on both; locality then leaves the values of $\Delta\sigma$ unchanged. Hence, $q(G') = q(G)$ and therefore $P_\nu^{Q,r,t}(G') = P_\nu^{Q,r,t}(G)$, while $G' \neq G$. Two distinct source graphs share one image. □

No premise is a formality. An exhaustive query leaves no node to delete, so the argument gives nothing. A query with a negated pattern, such as one selecting drives

that have *no* failed neighbour, consults elements that never enter the result, and deleting one of them can change the answer; the same applies to a derived value aggregating over the whole plant. A query that projects is excluded for the same reason: `MATCH (a:Drive)-[:NEXT_TO]->(m) RETURN a` contains no negation yet binds m without returning it. Bound-determinedness is separate because those three do not imply it: a pattern may require a witness that neither binds nor returns, as `MATCH (a)-[:NEXT_TO]->(b), (a)-[:CONTROLS]->()` does, and deleting that witness destroys the match, though every returned element survives. The analysis query of the case study returns every element it binds and requires no such witness. The auxiliary arguments must be held fixed, since G and G' would, in general, carry different revisions, and the documents would then differ in `SourceGraphRevision` alone; holding it fixed is the assumption that the two graphs are alternative contents labelled by the same revision.

Proposition 2 is why the pipeline is unidirectional for such q. It admits no left inverse, because a left inverse would make it injective. What follows is not that a reverse direction is impossible but that none is *determined* by the target alone, so a write-back channel would require an update policy resolving the ambiguity, placing the problem in the class of view updates studied in the literature on bidirectional transformations [71–73]; Katti et al. [74] illustrate the effort such a policy entails in an adjacent manufacturing setting. This is consistent with the role of a materialised view, which is read-only unless such an update policy is defined.

4.4. Design of the Target Submodel

4.4.1. Anchoring

The generated submodel is attached to the AAS of the *aggregate asset*, meaning the continuous press or the production line, and not to any of the individual drive shells. Under this anchoring, the submodel describes one aspect of exactly one asset, namely the topological state of that system, in accordance with the notion of a submodel in [6]. The affected subordinate assets appear as collections carrying a producer-namespace identifier.

This follows established IDTA practice rather than departing from it, since the submodel template *Hierarchical Structures enabling Bills of Material* [75] models relations between several assets within a single submodel, using `RelationshipElement` instances for `HasPart`, `IsPartOf` and `SameAs`.

4.4.2. Structure of the Template

The submodel, proposed here as the template `TopologyAnalysisResult`, is organised as shown in Table 3. A submodel that publishes a computed value together with its provenance is not new: the IDTA predictive-maintenance template [20] carries a predicted remaining useful life alongside a reference to the model that produced it, the model's boundary conditions and a confidence interval, and the carbon-footprint template [19] a calculated emission figure alongside its method, publication date and expiry. That triple is what `QueryHash`, `GeneratedAt` and `ValidUntil` carry here, where the predictive-maintenance template records neither a generation time nor a horizon. The wider pattern is older still [76,77], and for a source that changes under the result, it is set out in the recommendations on citing evolving data [22]. What is added follows from the pipeline. The extent of the artefact is governed by the size of the result graph, so a consumer needs the identity of the query and not merely the name of a method, which is what `QueryId` and `QueryHash` supply; the result mixes observed with computed values in one collection, which is why they are separated by `semanticId`; and the source is a mutable graph rather than a study, which is why `SourceGraphRevision` is mandatory.

Table 3. The proposed `TopologyAnalysisResult` submodel template. All `semanticId` values are relative to the namespace `https://tu-dresden.de/iat/aas/smt/TopologyAnalysisResult/1/0`, provisional and also the `semanticId` of the submodel itself. The `ValueOriginQualifier` carries the IRDI of the IEC Common Data Dictionary instead. The last block applies within each `/Node` and `/Relation`.

SemanticId	Type	Card.	Purpose
`/QueryId`	`Property` (string)	1	identifier of the analysis query
`/QueryHash`	`Property` (string)	1	SHA-256 digest of the query text
`/GeneratedAt`	`Property` (xs:dateTime)	1	time of generation
`/SourceGraphRevision`	`Property` (string)	1	revision token of the source graph
`/ExporterVersion`	`Property` (string)	1	version of the exporter
`/MappingConfigurationHash`	`Property` (string)	1	SHA-256 digest of the configuration ν
`/ValidUntil`	`Property` (xs:dateTime)	0..1	expiry of the snapshot
`/Nodes`	SMC	0..1	nodes of the result graph, omitted when $N_q = \emptyset$
`/Node`	SMC	1.. *	one element of N_q
`/OriginId`	`Property` (string)	1	URI of the preimage in the source graph
`/AssetReference`	`ReferenceElement`	1	reference to the shell of the asset, or to its `globalAssetId`
`/ObservedStatus`	`Property` (string)	0..1	observed state, from σ
`/Relations`	SMC	0..1	edges of the result graph, omitted when $E_q = \emptyset$
`/Relation`	SMC	1.. *	one element of E_q
`/Relationship`	`RelationshipElement`	1	model references to both node SMCs
`/OriginId`	`Property` (string)	1	URI of the preimage in the source graph
`/DerivedStatus`	`Property` (string)	0..1	computed state from $\Delta\sigma$, with the values `Critical_Adjacency_Failure` and `Non_Critical` the template fixes
ValueOriginQualifier	`Qualifier`	1	IEC CDD `0112/2///61360_4#AAF582` with the value calculated, on every property rule R4 writes
`/Labels`	`SubmodelElementList`	0..1	the labels of $\lambda(x)$ besides the primary one, as `Property` elements
external	`Property` or SMC	0.. *	a domain property from σ_q, carrying the `semanticId` the cascade of Section 4.5 resolves
`/SemanticResolution`	`Qualifier`	0..1	marks a generated `semanticId`

Three design decisions deserve comment. First, nodes and relations are separated into two top-level collections, which keeps the model navigable without knowledge of the source graph. Second, the properties of the template names, `ObservedStatus` and `DerivedStatus`, are distinguished by their `semanticId`. The distinction is drawn for the analysis vocabulary and not, in general. A domain property that the analysis computed rather than read carries a domain `semanticId` like any other, and rule R4 marks it, as every computed property, with the `ValueOriginQualifier` of the IEC Common Data Dictionary, whose purpose Section 6.5 sets apart from that of the `semanticId`. Third, a node is a `SubmodelElementCollection` and not an `Entity`, as the template for bills of material models its parts [75], because Assumption 3 excludes `Entity/statements` from traversal, so the observed status of a node would no longer be available as a query condition.

The two collections carry cardinality 0..1 rather than 1 because of the ordinary operating case: a monitoring query returns an empty result graph whenever the plant behaves normally. A memberless collection would be admissible, since the `value` of a `SubmodelElementCollection` has cardinality 0..* and the JSON mapping rules of [6] require only that the empty array be omitted; the template nevertheless makes the collection itself optional, so that its absence rather than its emptiness carries the meaning that the

query returned no elements of that kind, which a consumer tests once. It cannot be confused with the absence of an analysis, since six provenance attributes are mandatory: a consumer finding `QueryId` and `GeneratedAt` but no `Nodes` learns that the analysis ran at a stated time and found nothing.

The separation of `ObservedStatus` and `DerivedStatus` makes the distinction between σ and $\Delta\sigma$ visible *in the model* rather than in accompanying documentation, so that a consumer can tell which values were observed and which derived by the analysis query. Two of the rows are placed by design rather than by necessity. Definition 2 permits $\Delta\sigma$ on nodes and σ on edges, and the template fixes the state attribute of a node as observed and that of an edge as derived because that is what the analysis class considered here produces. Domain properties are not fixed by the template at all, which is what the row marked *external* records. Section 5.9 demonstrates a consumer that relies on `ObservedStatus` and `DerivedStatus` on the worked example of Figure 1.

4.4.3. Provenance and Temporal Validity

A materialised view is, by construction, a snapshot and becomes stale as the source changes. The six mandatory provenance attributes of Table 3 make this explicit and machine-readable. `GeneratedAt` and `SourceGraphRevision` allow a consumer to assess currency and to reject an outdated result, while `QueryId` and `QueryHash` identify which analysis produced the artefact and `ExporterVersion` and `MappingConfigurationHash` with which configuration, which makes the result reproducible and, together with Property 2, lets two exports be compared without normalisation. The validity horizon δ behind `ValidUntil` has to match the export policy, ending later than the next export is expected, or a consumer discards a view that is still current. Section 5 treats the staleness of the view as a quantity it bounds rather than as a defect.

4.5. Semantic Annotation Strategy

Semantic interoperability requires that a consumer interpret the received elements without prior agreement on their names, and two levels differ in how reliably that can be met. The *analysis semantics*, meaning what `TopologyAnalysisResult`, `Nodes`, `Relations`, `ObservedStatus`, `DerivedStatus` and `GeneratedAt` denote, are fixed by the template of Section 4.4 and are therefore well defined for any consumer holding it, independently of the source graph. The *domain semantics*, meaning what a property `speed` denotes, originate in the graph and can only be as good as the data there; the pipeline resolves them by the following cascade.

1. **Direct.** The key is itself an International Registration Data Identifier (IRDI) or an Internationalized Resource Identifier (IRI); for example, an ECLASS [66] identifier such as `0173-1#02-BAA120#008` imported with the engineering data. The test is the IRDI production of ISO/TS 29002-5 [78] or the IRI production of RFC 3987 [79] and belongs to the configuration ν so that Property 2 quantifies over it. Admissibility as an `Identifier` is not the test, since the metamodel types are bounded strings and every key would pass. A key that passes is adopted unchanged.
2. **Dictionary.** The key fails 1 and is resolved through a concordance table maintained with the pipeline configuration, mapping keys to concept definitions.
3. **Fallback.** Neither applies. A deterministic URI is generated from ν_0 and the injective $\mathrm{san}'(k)$ of rule R3, and the element receives a qualifier `SemanticResolution` = `generated`. The AAS has a convention for this case: the technical-data template reserves a concept description `SemanticIdNotAvailable` for elements carrying no distinctive `semanticId` [80]. It is the more standardised choice and the less informa-

tive one, one constant standing for every unresolved key, where the generated URI is derived from the key and keeps it recoverable.

The qualifier of the third case is what makes the strategy defensible: a consumer encountering a generated URI detects it by machine instead of mistaking a syntactically valid identifier for a semantically grounded one. The claim is therefore bounded and testable. The *analysis result* is machine-interpretable against the template, whereas the interpretability of the *domain attributes* is a property of the data source that the model declares rather than assumes.

4.6. Mapping Rules

The following rules realise τ_ν. Rule R0 consumes the arguments the result graph does not carry and rules R1 to R4 consume the result graph itself.

Rule R0 builds the envelope and the provenance. The generated `Submodel` takes an `Identifiable/id` derived from ν_0 and j, an `idShort`, `semanticId` and `administration` version from the template, `kind Instance` and, where the corresponding sets are non-empty, the two top-level collections `/Nodes` and `/Relations`. R0 forms the seven provenance properties of Table 3: `QueryId` from j, `QueryHash` as the SHA-256 digest of the query text Q with its line ends normalised, `ExporterVersion` as the version of the exporter, `MappingConfigurationHash` as the SHA-256 digest of the canonical serialisation of ν, `SourceGraphRevision` from r, `GeneratedAt` from t and `ValidUntil` from $t + \delta$. `QueryHash`, `SourceGraphRevision`, `GeneratedAt` and `ValidUntil` are the only elements Q, r and t determine, which is why Definition 3 carries Q, r and t as arguments at all, and the last two are what Property 2 refers to.

Rules R1 to R3 map the structure of the result graph and rule R4 the analysis outcome.

Rule R1 maps the nodes. Each $x \in N_q$ becomes an SMC carrying the `semanticId` `/Node` of Table 3 and, as the first member of a node, an `OriginId` property carrying `/OriginId`, followed by a `ReferenceElement` `AssetReference` carrying `/AssetReference`. Its value is a `ModelReference` keyed `AssetAdministrationShell` on the identifier of the shell of the asset x stands for, which ν names as a property key of x, or, where the asset has no shell, an `ExternalReference` to its `globalAssetId`, under a second key. Both keys are consumed here and excluded from rule R3, and where x carries neither, τ_ν refuses the export rather than emit an incomplete instance. Its `idShort` is formed as

$$\texttt{idShort}(x) := \mathrm{san}(\ell(x)) \parallel \text{‘‘_’’} \parallel \mathrm{enc}(\mathrm{id}(x)), \tag{7}$$

where san and enc are the maps of Lemma 2. The suffix is required: without it, two nodes carrying the same primary label would receive the same `idShort` and violate AASd-022, and rule R2 faces the same problem for edges of one type. Lemma 2 makes uniqueness a property of the construction rather than something checked afterwards. The internal identifier is additionally preserved as a URI $\nu_0/\mathrm{enc}(\ell(x))/\mathrm{enc}(\mathrm{id}(x))$ in a property `OriginId` (Property 3), globally unique under the namespace condition of Property 1, which is what keeps it apart from the identifier of the generated submodel and from a fallback concept. The escape is what makes the two components separable and is not optional, since nothing in Definition 1 prevents a label or an identifier from containing a solidus. Where $\lambda(x)$ holds labels besides the primary one, they are carried, last, as a `SubmodelElementList` with the reserved `idShort` `Labels` and the `semanticId` `/Labels` of Table 3, whose `typeValueListElement` is `Property` and whose `valueTypeListElement` is `xs:string`, as AASd-109 requires for that combination. Its members carry no individual `semanticId`, which keeps AASd-107 satisfied without a `semanticIdListElement`, and none carries an `idShort` either: AASd-117 exempts direct children of a list from requiring

one, and since the withdrawal of AASd-120, the metamodel no longer forbids one [6], so the omission is a decision of the rules, which Property 2 needs stated for the serialisation to be determined. Every `idShort` reserved inside a collection is the leaf of a `semanticId` of Table 3, and rule R3 must avoid it. The label branch is specified but neither exercised nor implemented, the example's nodes carrying one label each. The attribute `category` would be the obvious place for them and is deliberately not used, since the metamodel marks it deprecated and types it as a single `NameType`, so a set of labels could be placed there only under a private encoding convention.

Rule R2 maps the edges. The metamodel offers two constructs for an edge that carries data of its own, and the choice between them is a design decision rather than a necessity. The specification states that "if this semantic definition requires additional information not contained in the first or second object referenced via the relationship, the SubmodelElement type `AnnotatedRelationshipElement` shall be used instead" [6] [Part 1, Relationship Element]. The condition is on the semantic definition and is not met here, since the generated `RelationshipElement` carries nothing beyond its endpoints and the derived status is a sibling inside the enclosing `/Relation` collection. Two considerations decided this. Assumption 3 excludes annotations from the queryable fields, whereas atomic values nested inside a `SubmodelElementCollection` are reachable by `$sme#value` (Assumption 4); and treating nodes and edges alike keeps Equation (7) and Property 3 uniform over $N_q \cup E_q$. Since edges of an LPG may carry properties while a native `RelationshipElement` admits no sub-elements, each $e \in E_q$ becomes an SMC carrying the `semanticId` `/Relation` and encapsulating (i) a `RelationshipElement` whose `first` and `second` are model references to the SMCs of $\rho_q(e)$ in that order, carrying the `semanticId` `/Relationship` and, as AASd-117 requires an `idShort`, the reserved `Relationship`, (ii) the `OriginId` of rule R1, (iii) the properties of the edge and (iv) its `Labels`. The `relationshipType` of the source system is treated as the primary label, and the `idShort` is formed by Equation (7).

Rule R3 maps the properties. Each pair $(x,k) \in \mathrm{dom}(\sigma|_{(N_q \cup E_q)\times\mathcal{K}})$ becomes a `Property` of `value` $\sigma_q(x,k)$ and `valueType` $\mathrm{ty}(\sigma_q(x,k))$, whose `semanticId` is resolved by the cascade of Section 4.5. Its `idShort` is $\mathrm{san}'(k)$, where $\mathrm{san}' : \mathcal{K} \to \Sigma^*$, injective, is a sanitiser whose image satisfies AASd-002 within `NameType`, possible because $\mathcal{K}$ is finite, and avoids the identifiers the template reserves, the fallback URI built from it staying inside `Identifier`. One class of key is exempt: the configuration names a partial injection $\mathrm{tmpl} : \mathcal{K} \rightharpoonup \mathcal{A}$ into the analysis vocabulary $\mathcal{A}$ of Table 3, and a key in its domain at an $x \in N_q$ takes the reserved `idShort` and analysis `semanticId` it names rather than passing through san' and the domain cascade; in the case study $\mathrm{tmpl}(\texttt{status}) = \texttt{ObservedStatus}$. `OriginId`, `Labels` and `Relationship` are not reached this way, since rules R1 and R2 construct them, and they are among the identifiers both maps must avoid. As tmpl is injective with image disjoint from that of san', the union is injective, and as σ_q is a function the keys of one element are pairwise distinct, so injectivity carries AASd-022 to the members of each collection for the reason Lemma 2 carries it to the collections. A structured value has no case here, the standing assumption of Section 4.2 excluding it, so san' is needed on $\mathcal{K}$ alone and the rule admits no recursion; the collection form the *external* row of Table 3 is reserved for that case and is constructed by no rule given here. The instance released with the implementation names the properties by the resolved concepts rather than by the raw keys of Figure 1.

Rule R4 maps the derived properties. The configuration names a second partial injection $\mathrm{tmpl}_\Delta : \mathcal{K} \rightharpoonup \mathcal{A}$ into the analysis vocabulary, disjoint in its image from tmpl, san' and the identifier rules R0 to R2 reserve. A pair $(x,k) \in \mathrm{dom}(\Delta\sigma)$ with $x \in E_q$ and $k \in \mathrm{dom}(\mathrm{tmpl}_\Delta)$ becomes a `Property` of `value` $\Delta\sigma(x,k)$ and `valueType` $\mathrm{ty}(\Delta\sigma(x,k))$ carrying the reserved `idShort` and analysis `semanticId` tmpl_Δ names; in the case study

$\text{tmpl}_\Delta(\texttt{assessment}) = \texttt{DerivedStatus}$. Any other pair takes san′ and the domain cascade of Section 4.5, as rule R3 treats a source key outside dom(tmpl). A derived key lying in dom(tmpl) deliberately does not take the reserved identifier, which would publish a computed value under the concept for an observed one. R4 is therefore total on $\Delta\sigma$, and Definition 2 needs no restriction on which keys a query may compute, the standing assumption bearing on values and not on keys. Such a property reaches the consumer with a domain `semanticId`, which alone does not mark it as computed. Rule R4 carries the analysis semantics. Without $\Delta\sigma$, the pipeline would reduce to a format conversion. Every property rule R4 writes, under tmpl_Δ or through the cascade, carries the `ValueOriginQualifier` of the IEC Common Data Dictionary, `0112/2///61360_4#AAF582`, with the value calculated, `0112/2///61360_7#CTA004`, in the form the IDTA template for time series lists it [23]. A consumer can therefore recognise any computed property as computed, not only the `DerivedStatus` template names.

4.7. Implementation

4.7.1. Target Environment

The intended deployment stores the plant topology in Neo4j [81], a property-graph database with index-free adjacency [82], alongside an AAS infrastructure such as Eclipse BaSyx [83]. In that environment, a node may carry several labels, whereas a relationship carries exactly one `relationshipType`, which rule R2 treats as the primary label. The internal identifier is the `elementId`, which Equation (7) and the namespace ν_0 consume after escaping. Critical failure patterns are identified by horizontal pattern matching in Cypher and evaluated server-side. Listing 1 gives the query of the case study. Its records become the result graph by a convention that is part of ν: the columns `a`, `r` and `b` are graph elements, the start and end node of `r` fixes its direction, and every further column is a derived property of `r`, which rule R4 consumes. An element returned in several records must be returned identically, and the closure condition holds by construction, since every edge brings its two endpoints. Neo4j guarantees no order of the records, so the exporter serialises N_q and E_q in the order of their identifiers and the properties in the order of their keys, the fixed orders Property 2 relies on.

Listing 1. Analysis query q of the case study. The `CASE` expression is what rule R4 consumes: it computes a value present nowhere in the graph, which is the difference between an analysis and a structure-preserving conversion. The `OR` admits an edge with one failed endpoint, so the sound endpoint enters the result graph through the closure condition.

```
MATCH (a:Drive)-[r:NEXT_TO]->(b:Drive)
WHERE a.status = 'err' OR b.status = 'err'
RETURN a, r, b,
       CASE WHEN a.status = 'err' AND b.status = 'err'
            THEN 'Critical_Adjacency_Failure'
            ELSE 'Non_Critical' END AS assessment
```

4.7.2. Reference Implementation and Verification

The template of Table 3, the rules for arbitrary scalar properties, the conversion of query records and the constraint checker are provided as a reference implementation of the export function together with the instances it produces. Conformance (Property 1) is checked in two stages, and the distinction between them matters.

The first stage is the constraint checker accompanying the implementation. It evaluates AASd-002, AASd-022, AASd-117, AASd-118 and AASd-121 to AASd-125 together with the

structural well-formedness of `Property` and `RelationshipElement` instances, reporting each violation with its constraint number and element path. The remaining `Qualifier` and reference constraints hold trivially here and are left to the second stage, which claims completeness against the metamodel. That stage uses the reference library `aas-core3` [84] and the official AAS test engines; the example instances pass all four checks, namely deserialisation, verification of every metamodel invariant, an identical serialisation cycle and validation of the `Environment` against the official schema, and the dated report is preserved with the implementation, to be regenerated whenever an instance changes.

Each stage found a defect. The first detected a violation of AASd-022 in an earlier version of rule R2, which derived the `idShort` of an edge from its `relationshipType` alone, so two adjacency edges received the same identifier; Equation (7) is the correction and converts the constraint from something checked into something guaranteed. The second exposed a distinction the class model does not capture: an instance may satisfy every metamodel invariant and still be rejected as a file, because the JSON mapping clause forbids empty arrays, so `aas-core3` accepts a document the test engines reject. The rule applies to `Nodes` and `Relations` as well, and the empty result graph is not marginal but what the pipeline produces whenever the monitored pattern is absent, which in a functioning plant is almost always. The implementation therefore emits the submodel embedded in an `Environment` carrying only populated collections, and the first-stage checker was given a file-level rule applied recursively.

End-to-end run. The analysis query is executed, not only released. A plant graph of two rows of 100 drives, each carrying a status, four further properties and the identifier of its shell, with `NEXT_TO` edges of two properties between neighbours and two controllers, is generated into Neo4j 5.26.8. The query of Listing 1 is evaluated through the official driver, its records are converted by the convention of Section 4.7.1, and the export is validated in both stages, written to an Eclipse BaSyx `aas-environment` server [83], version 2.0.0 milestone-15 with its in-memory back-end, read back and validated again. Three fault configurations are run: none, the base case of four isolated faults with 12 nodes and 8 edges, and one with an adjacent pair, 10 nodes, 7 edges and one critical edge. Each exported result equals the one an independent oracle computes from the generator. Two exports of the same revision are equal up to their two timestamps (Property 2); deleting a sound drive far from the faults leaves the export equal up to its timestamps and revision (Proposition 2); every `OriginId` leads back to its element on the same database state and every `AssetReference` to a shell on the server (Property 3); and the document read back equals the one written, up to the default value of `Qualifier/kind` that the server writes out. The exporter, version 1.0.0, takes its configuration from a versioned file whose digest every exported submodel records. The run takes place in containers under Docker Desktop with WSL 2 on an AMD Ryzen 7 PRO 250 under Windows 11, the client on the Docker network of the two servers, and every series of measurements follows a warm-up of the servers. Section 5.9 evaluates the result read from the server, and Section 5.3 reports the costs the run measures.

5. Evaluation

5.1. Design and Epistemic Status

The evaluation applies the pipeline of Section 4 to the continuous-press scenario introduced in Section 1.1. It takes the form of a parameterised cost model rather than a measurement on a running installation. Its export side is the exception: the end-to-end run of Section 4.7 measures the query, the conversion of its records, the serialisation, and the write and the read of the published submodel, and Sections 5.3 and 5.10 set these measurements against the model. This is a deliberate choice, and its consequences are stated here.

A single measurement is bound to one network topology, one hardware configuration and one server implementation. Its numerical outcome would not generalise, and the reader could not separate the properties of the method from the properties of the testbed. A cost model inverts this trade-off. It forfeits the authority of an observation but exposes every cost contribution individually and permits the response to be examined along the parameter axes. Where a published measurement fits a parameter, it is adopted rather than assumed.

Two claims must therefore be kept apart throughout this section.

1. That a single retrieval is cheaper than n sequential retrievals follows from materialisation itself. It is not an independent finding and is not presented as one.
2. Whether the advantage survives an *optimised* baseline, whether it is caused by the query language or by the persistence layer, and from which operating point it repays the cost of the replicated model do not follow definitionally. They are the subject of Sections 5.4–5.7.

Three findings carry this section, and Section 6.1 ranks them by how well each is supported: the latency and volume ratios against the optimised baseline, the location of the difference between the two architectures, and the location of the limit, that no condition and no filter reaches the endpoints of a `RelationshipElement`, whatever share of the payload the adjacency is under a given plant model. Beside them stands the price of the replicated graph, which Section 5.6 expresses as a read-to-change ratio. Everything else quantifies how far each of these moves when one assumption is varied.

5.2. Scenarios

The evaluation is conducted on the horizontal problem. The vertical one is governed by the same mechanism and is reported in Section 6.5, where it also marks the boundary of the approach. This subsection introduces six scenarios, of which five enter the comparison. A4 is described for completeness and priced in Section 6.3. The filter bound A3f, the plant-side encoding A6 and the finer topology partition A5 are introduced in Sections 5.4 and 5.8. Section 5.4 reports the five that enter the comparison together with what each one costs.

Scenario A1 lists the shells and then issues one request per Referable; A2 paginates and merges those requests. Hypertext Transfer Protocol (HTTP) keep-alive and concurrent requests are enabled in every scenario, A1 included, so that the step from A1 to A2 isolates pagination and merging, and scenario B differs from A2 in its server-side parameters alone, being served by a graph-backed repository as realised by [48].

Scenario A3 is the baseline against which the approach is assessed. Beyond A2, it names the topology Submodels by `$sm#semanticId` and exploits `value` as a queryable field (Assumption 3): a condition on `$sme#value` restricts the status retrieval to the drives in a fault state. Its two remaining requests, one for the topology of both controllers, which A2 already merged, and one page of status Submodels, are independent and issued in a single round. A disjunction over a `$sm#semanticId` condition and the value condition is derivable from the grammar and would merge both into one, which the model does not assume; it lowers the latency from 53.22 ms to 52.36 ms here, by 3.73% at the worst seed of Section 5.10, by 8.31% once A3's round is charged the server term it pays only on average, and by 10.16% where both hold. Within that margin A3 is the best a client can do *at a fixed plant model, without a private replica, in the API's default* Normal *serialisation, on one connection per parallel request, with the payload uncompressed and without* `$filters`. Four of those six, compression among them, are priced in Section 5.4 and two in Section 6.3, each buying more than the merge does.

A4 holds a private replica of the topology and refreshes it out of band, so a query round transfers asset state only. It lies outside the comparison for the reasons Section 6.3 gives and is priced there and in Table 7. Scenario C is one unpaginated retrieval of the consolidated submodel.

Reporting a comparison against A1 or A2 alone would overstate the result. A competent client paginates, parallelises and above all filters on values wherever the language permits. Scenario A3 does all three; every ratio below refers to it unless stated otherwise, and A1 and A2 are retained only to show what the individual optimisations contribute. A3 also shows where the language limit binds. Value filtering reduces the *status* retrieval in proportion to the fault rate but cannot reduce the *topology* retrieval, since the `first` and `second` attributes of a `RelationshipElement` are not queryable, so the client transfers the topology in full and evaluates the adjacency predicate locally on every cycle. Scenario B is the discriminating case for the second claim above: it changes only the server-side processing time, so if the advantage of scenario C were an artefact of document-oriented persistence, B would close most of the gap.

5.3. Parameters

Every quantity entering the model belongs to exactly one of four classes, recorded with the quantity, and the distinction determines which sentences the evaluation may support. *Established* (E) means measured here or fixed normatively by a specification, and is verifiable by recomputation or by consulting the standard; *published* (P) a measurement reported by others and adopted with citation; *assumed* (A) set without external support and consequently varied over a range; *derived* (D) computed from the preceding three by a stated formula. Monte Carlo figures below use 1000 runs and, save a seed sweep, the seed `20260725`, so two runs on the same parameter set produce identical result files. Payload figures are in binary multiples, so KiB denotes 1024 B, and transfer rates in decimal multiples, as is usual for network bandwidth.

The byte sizes of Table 4 are established quantities. Each structure was built as a JSON object following the serialisation rules of IDTA 01001-3-2 [6], encoded as UTF-8 with the compact separators an HTTP interface uses, and its length recorded; none was estimated or transcribed by hand. The construction is calibrated against the worked example printed in the specification: the property `MaxRotationSpeed` in *Normal* format measures 198 B there and 198 B here, with byte-identical structure, which makes the serialisation logic checkable independently of every later step. What the table does *not* establish is how many properties a real node or drive submodel carries; those counts are assumptions varied in Section 5.7. The request and round counts reported below are derived: their form follows from the AAS API, and their magnitude from the assumed plant size.

The synchronisation cost of the replicated graph is a published measurement. Sonnenberg et al. [43] report under 6 ms per property-update event for a Kafka-to-Neo4j ingestion pipeline, and this evaluation adopts that value unchanged. The parameter most open to challenge is deliberately taken from a measurement published by a neighbouring approach.

The cost of an export run is decomposed, because two of its terms differ in how well they can be substantiated. The end-to-end run of Section 4.7 measures the Cypher query: evaluating Listing 1 through the driver on Neo4j 5.26.8 takes 2.98 ms (median of 500 repetitions inside one Docker network, 5th to 95th percentile 2.50 ms to 3.98 ms), of which the server reports about 1 ms. Sonnenberg et al. [43] report 3 ms for a query with simple value access and 67 ms for one parsing version strings, neither of them a topological pattern match. The measured value holds for one machine, one database size and one query, so c_{query} stays class A at 3 ms, which the measurement confirms for the case study, and the range up to 67 ms is carried through Section 5.6. The serialisation *is* measurable, since it

is the reference implementation itself. Executing it over result graphs of increasing size with 500 repetitions per point on an AMD Ryzen 7 PRO 250 under Windows 11, with the garbage collector suspended and the median taken, yields the scaling law

$$c_{\text{serialise}}(m) = 19.30\,\mu\text{s} \cdot m, \qquad m = |N_q| + |E_q|, \tag{8}$$

with a coefficient of determination of 0.9974, measured on the revised template, whose asset references and qualifiers make each element larger than it would be without them. The intercept is constrained to zero because a model with no elements costs nothing. What is reported is a scaling law rather than an absolute duration, the former being a property of the implementation and the latter of the machine; the slope remains machine dependent, so the cost model reads it from the benchmark output, and the value quoted here is the one that output records. For the case study, this gives $c_{\text{serialise}} = 0.386\,\text{ms}$, and together with the query at 3 ms and the write, $c_{\text{export}} = 6.386\,\text{ms}$. Writing the exported submodel to the BaSyx server takes 1.21 ms in the end-to-end run, so the 3 ms assumed for c_{write} is conservative for this server. The element count follows from the topology and the fault rate, and the closure condition of Definition 2 makes it larger than the number of faulted drives. Four drives are in a fault state, each incident to two `NEXT_TO` edges, and the query returns every edge with at least one faulted endpoint, so $|E_q| = 8$; that is the sparse-fault upper bound, since adjacency among the faulted set only shrinks it. Closure then adds a distinct sound endpoint per edge under the same bound, giving $|N_q| = 12$ and $m = 20$. The fault rate and the selectivity $s = |N_q|/|N|$ are therefore different quantities, the first assumed and the second derived from it. Table 5 lists the remaining retrieval quantities with their class. The effective concurrency of a round is $\min(\text{client}, \text{server})$ and enters both the latency waves and the server time.

Table 4. Payload constants, obtained by serialising AAS structures element by element (class E). Each row is measured on its own identifiers, so a composite row does not decompose into the atomic ones. *Value-only* is the `$value` serialisation, `$filters` the filtered size. The plant-side adjacency elements carry the identifiers a plant model would give them and not the longer ones of Table 3, so 1199 B rather than 1301 B per adjacency element, which lowers most ratios reported later. Element counts are assumptions (class A), the property counts being varied in Section 5.7.

Structure	Normal [B]	Value-Only [B]
`Property` (with `semanticId`), `$filters` 89 B	198	25
Node SMC (5 properties, `OriginId`, `AssetReference`)	1854	322
Plant edge SMC (rel., 2 props), `$filters` 595 B	1199	527
Result edge SMC (rel., `OriginId`, 2+1 props, qualifier)	2166	624
`RelationshipElement` of an edge, bare (bill-of-material form)	631	—
Drive status submodel (12 properties), `$filters` 407 B	2566	125
Provenance block, and in *Normal* the two collection headers	2427	371
`Submodel` envelope, `AssetAdministrationShell` envelope	353/541	—
`ModelReference`	224	—

The request counts of the last block are derived and can be checked against the table. A1's requests are one for the shell listing, one per controller and one per drive, which is $1 + 2 + 200 = 203$ in the horizontal problem; the vertical one costs 1 + 5 + 10 + 2000 + 10,000 = 12,016 over its five levels. The over-fetch factor, the ratio of the drives the value filter returns to those the analysis needs, is set to unity, the setting most favourable to the baseline.

Server-side time is decomposed so that each term can be assessed on its own, namely a fixed cost per request for dispatch and authorisation (2 ms), a per-element cost for lookup and access control (12 µs for document-oriented and 3 µs for graph-backed persistence),

and serialisation at 200 MB s^{-1} irrespective of the back-end. For a page of one hundred Submodels, this yields 17.7 ms, an effective rate of 14.5 MB s^{-1}, which is slow for pure serialisation, but an AAS repository also resolves and authorises each element. Client-side parsing is set to 200 MB s^{-1}.

Table 5. Topology and timing parameters of the cost model, with the class of each. The topology figures fix the case study of Section 1.1 and the timing figures are the base point of the sensitivity analysis. A bullet marks the quantities swept below, each over the range recorded with the corresponding experiment, and the remainder are held fixed at the value the case study gives them. A circle marks the derived quantities Section 5.7 varies one at a time.

Quantity	Value	Class	Swept
Drives per press	200	A	•
Controllers holding the topology	2	A	•
Neighbours per drive, the node degree	2	A	•
Vertical fan-out, plant to line to press to drive to sensor	$5 \times 2 \times 200 \times 5$	A	
Fault rate	0.02	A	•
Over-fetch factor	1.0	A	•
Publication cost of one export c_{write}	3 ms	A	•
Median round-trip time (RTT)	20 ms	A	•
Round-trip dispersion, geometric standard deviation	1.6	A	•
Bandwidth	100 Mbit s^{-1}	A	•
Concurrent client connections	8	A	•
Concurrent server workers	8	A	•
Page size of the API	100	A	•
Fixed server cost per request	2 ms	A	•
Server cost per element, document-oriented/graph-backed	12 µs/3 µs	A	•
Server serialisation and client parsing rate	200 MB s^{-1}	A	•
Requests, A1 horizontal/vertical, A3	203/12,016/2	D	○
Result graph $\lvert N_q\rvert/\lvert E_q\rvert/m$	12/8/20	D	○
Selectivity $s = \lvert N_q\rvert/\lvert N\rvert$	0.06	D	○

The ratio of four between the two persistence variants is an assumption, and it is worth stating why no published figure was adopted for it. Sonnenberg et al. [43] report the only published measurements comparing the two access paths, but they do not yield a single ratio. For their use case, selecting robots by parsing and comparing version strings, Cypher required 67 ms against 27 ms for the AAS REST API, whereas a simplified query performing a plain value lookup completed in 3 ms. Which access path is faster therefore depends on the query, and their own observation that a local deployment favours the REST API through negligible network latency applies as well. The ratio is consequently treated as an assumption and varied in Section 5.5 from 16:1 in favour of the graph to 1:2 against it, the latter covering the case their measurements exhibit.

The base case assumes an established connection, corresponding to the continuously operating assistance system described in Section 1.1. Connection establishment is a one-off cost that affects scenario C disproportionately, since C issues only a single request. It is therefore reported separately in Section 5.7.

Apart from the export side, simulation was chosen over a physical testbed because a testbed would have to fix a hardware configuration and a network topology that would then require justification, while the installations the case study is drawn from belong to industrial partners whose network layout cannot be disclosed. Section 6.6 states what a physical evaluation would add.

5.4. Results

Table 6 reports the outcome over 1000 runs per scenario.

Table 6. Scenario comparison for the horizontal problem. The round structure is derived from the API, a retrieval's pages being charged within one round although the page cursor keeps them sequential, which favours A2 and B; requests, payload and latency follow from the parameter set of Section 5.3 (D). *Rnd.* counts rounds, not network waves, and the four latency contributions, given where decomposed, are round-trip time, server, transfer and client, each the median of that contribution, so they need not add exactly to the total. The lower block gives four bounds on the baseline, each presupposing something the comparison should not assume. Median over 1000 runs; the brackets are the 5th and 95th percentiles, the round-trip draw being the only random term.

Scenario	Req.	Rnd.	Payload	Latency Contributions [ms]				Total [ms]
				RTT	Srv.	Trf.	Cli.	
A1 one request each	203	3	737.1 KiB					1191.8 [1095.4; 1305.2]
A2 paginated	4	3	737.1 KiB	68.34	30.11	60.38	3.77	162.6 [139.5; 206.5]
B graph-backed	4	3	737.1 KiB	68.34	13.90	60.38	3.77	146.4 [123.2; 190.3]
A3 value-filtered	2	1	244.9 KiB	25.39	6.51	20.06	1.25	53.22 [41.4; 78.7]
C consolidated submodel	1	1	41.36 KiB	20.02	3.70	3.39	0.21	27.32 [16.8; 50.6]
Bounds on the baseline								
both requests merged	1	1	244.9 KiB					52.36
plant adjacency bare	2	1	134.0 KiB					40.88
A3f, with `$filters`	2	1	118.5 KiB					39.23
A3 under `$value`	2	1	103.4 KiB					40.54

The optimisations are worth what they cost. Pagination and merging reduce the horizontal case from 1191.8 ms to 162.6 ms. Adding the value filter and collapsing the three rounds into one reduces it further to 53.22 ms. Both contribute: the filter removes the objects the analysis does not read, and the collapse removes two of the three dependent rounds. Against this baseline, scenario C reduces latency by a factor of 1.95 and the transferred volume by a factor of 5.92. The corresponding figures against A2, namely 5.95 and 17.8, are not reported as the result of this work, because a client that fails to use the value filter is not the state-of-the-art but an unoptimised implementation.

The decisive observation concerns the composition of what A3 still transfers. Of its 244.9 KiB, 95.9% is the topology. Value filtering reduces the status portion in proportion to the fault rate and leaves the topology portion untouched, because `RelationshipElement/first` and `/second` are not queryable (Assumption 3), so the baseline retransmits the complete topology on every evaluation cycle and that portion scales with the installation rather than with the answer. This is Proposition 1 expressed as a volume, and it is why the advantage of scenario C persists against any client under the qualifications of Section 5.2. A client keeping a private replica is faster, which is scenario A4 of Section 6.3.

Where Table 6 gives them, the latency columns attribute every millisecond to a named term. A fifth term, connection establishment, is absent because the base case assumes an existing connection; Section 5.7 reports the cold start separately, where it matters more to scenario C than to the baseline.

Four bounds belong with the baseline. Each presupposes something the comparison should not assume, and the lower block of Table 6 gives them; against scenario C they leave latency ratios of 1.92, 1.50, 1.44 and 1.48. The first is the merged client just described, which changes the request structure and not the payload. The second concerns how the plant models its own adjacency, a decision of the deployment: the element is priced here as a `SubmodelElementCollection` holding a `RelationshipElement` and two attributes, the form the pipeline's own output takes, whereas the bill-of-material template of the IDTA [75] uses a bare `RelationshipElement`, giving a volume ratio of 3.24. Another form, carrying

the neighbour identifier in the atomic value of a `Property`, is priced below as scenario A6. The third, A3f, is `$filters`, added one version after the language itself. It strips from each returned object everything the metamodel does not make mandatory and the analysis does not read and removes about half of the volume (2.86 to C) and the sub-elements the server would price, but none of the round structure, since the number of adjacency elements returned is unchanged. The last is the value-only (`$value`) serialisation applied to the baseline alone (volume ratio 2.50), whose compression is bought by discarding the semantic annotation, so that the two sides would no longer carry the same artefact.

The encoding that reaches the adjacency. Section 3.3 names a plant model that brings an endpoint within reach of a value condition: the neighbour identifier carried as the atomic value of a `Property` beside the relationship. Scenario A6 prices it. One condition on `$sme#value` returns every faulted drive with its own neighbour list, which settles the pattern of the case study inside a single response, since a neighbour that is itself faulted is already in the returned set. It takes 23.79 ms against 27.32 ms for scenario C, a ratio of 0.87, and transfers less than a third as much. Because that condition is exhaustive over the faulted drives, the client can certify any drive it can name as sound by complementation, and the neighbour lists of two faulted ends intersect in the sound drive between them, so a two-hop pattern is settled in the same response and at the same cost, while the result graph the pipeline publishes grows with the depth. On the analysis this paper motivates, the encoding is therefore cheaper than the pipeline, and that is stated here rather than left for a reader to discover. What it does not do is remove the structure. Deeper patterns need adjacency the condition does not return, which arrives only in the submodels of drives it does not select, and the dependent rounds of Observation 1 return with them; how many a client needs at a given depth turns on how it reuses the condition and on the topology, and is not priced here, the case study's topology fixing no round count for a k-hop match. Taking the other branch of that observation instead and retrieving the encoded plant in bulk costs 3.52 against C, or 1.51 where the server also implements `$filters`, the option bound A3f grants the baseline. The encoding therefore buys reach, and buys it when the plant is modelled, whereas what depth an analysis needs is settled when it is asked.

Compression and the composed client. Table 7 grants transport compression to every client and to scenario C alike. It is measured on whole response bodies, since compression is not additive, with gzip at level 6 (zlib-ng 1.3.1, whose ratios differ slightly from those of the reference zlib) and Brotli at quality 5, and it charges the processor time for compressing on the server and decompressing on the client (class E, on the machine of Equation (8)). The topology response of A3 compresses 61-fold with gzip, a page of status submodels 18-fold and the consolidated submodel 25-fold, because the repeated identifiers and references of a plant topology are what compress best. All of these bodies are built as the payload model prices them. The submodel the end-to-end run read back from the server compresses 33-fold, mostly because each of its unresolved properties repeats the same qualifier and is therefore not taken to stand for the modelled one. Compression narrows the gap: with gzip the latency ratio falls from 1.95 to 1.40 and the volume ratio from 5.92 to 2.67. The client optimisations compose. `$filters` lowers the ratio to 1.24 and multiplexing the two requests onto one connection to 1.02, so the fully composed client reaches parity with the pipeline at base parameters, and 1.01 at the fast corner. Merging both requests into one, without `$filters`, is an alternative to multiplexing rather than a further step and gives 1.37. Under compression the cache and the plant-side encoding remain faster than the pipeline, at 0.94 and 0.95, and the finest partition remains slower, at 1.95, while transferring less. Section 6.1 states where the pipeline nonetheless offers an advantage.

Table 7. Client optimisations composed, with transport compression granted to every client and to scenario C. Each cell gives the latency and the volume ratio against C under the same compression, at base parameters, the last column the latency ratio at the fast corner (tenfold bandwidth, round-trip dispersion 1.05). The first three rows are cumulative. Merging both requests into one, here without `$filters`, is an alternative to multiplexing.

Client	Ratio to C, Latency/Volume			Ratio to C, Latency
	Uncompressed	Gzip	Brotli	Gzip, Fast Corner
A3, value filter	1.95/5.92	1.40/2.67	1.43/1.95	1.20
and `$filters` (A3f)	1.44/2.86	1.24/1.57	1.25/1.11	1.03
and multiplexing	1.24/2.86	1.02/1.57	1.03/1.11	1.01
A3 merged, without `$filters`	1.92/5.92	1.37/2.67	1.39/1.95	1.36
A4, topology cache	0.86/0.24	0.94/0.34	0.94/0.35	0.95
A5, finest partition	1.79/0.54	1.95/0.22	1.93/0.15	1.88
A6, plant-side encoding	0.87/0.29	0.95/0.40	0.95/0.42	0.95

5.5. Contribution of the Persistence Layer

Scenario B improves upon A2 by 10.0% (162.6 ms to 146.4 ms). Rounds, requests and volume are identical; only the server term falls, from 30.11 ms to 13.90 ms. Applied to the reference baseline instead of to A2, the back-end gives 50.30 ms against 53.22 ms, a ratio to scenario C of 1.84, so it removes 2.92 ms, or 11.3%, of a gap of 25.91 ms.

The conclusion does not rest on how the server-side parameters were chosen. Setting the server term to zero in *every* scenario, an idealisation applied symmetrically that no implementation can surpass, leaves A2 at 132.5 ms, A3 at 46.71 ms and C at 23.62 ms, a ratio of 1.98 against 1.95. Nor does it rest on the assumed ratio between the persistence variants: varying the per-element server time of the graph back-end from 0.75 µs to 24 µs, from 16:1 in favour of the graph to 1:2 against it, moves the baseline on that back-end between 49.6 ms and 57.1 ms, so even at the favourable end its ratio to C stays at 1.81. What settles the matter is what the residual gap consists of: A3 and C issue the same one dependent round, so it is not a round-count effect, and roughly two thirds is volume and a fifth an order statistic over A3's two parallel requests, worth some 5 ms. Neither is touched by the store behind the interface. *The limit resides in the query language, not in the persistence layer*: a graph-backed repository, and equally an AAS-QL-to-Cypher compiler, accelerates the queries AAS-QL can express and does not enable those it cannot.

5.6. Amortisation

Scenario C presupposes a replicated topology, whose maintenance consumes resources, so the advantage is real only if reads recover that expenditure. Equating the work both architectures spend per unit of operating time yields the break-even ratio

$$r^{*} = \frac{c_{\text{maint}}}{W_{\text{A3}} - W_{\text{C}}}, \quad c_{\text{maint}} = c_{\text{sync}} + \eta \cdot c_{\text{export}}, \quad c_{\text{export}} = c_{\text{query}} + c_{\text{serialise}}(m) + c_{\text{write}}, \tag{9}$$

where W_X is the resource time a read of scenario X occupies on the server, client and link; its total of Table 6 less the time spent waiting for round trips, and $f_{\text{read}}/f_{\text{update}} > r^{*}$ favours the pipeline. Since c_{sync} is a cost per property-update event entering the replica, f_{update} is the rate of such events and not the rate at which the plant is rebuilt, and f_{read} is the rate at which the assistance system evaluates the failure condition. Which properties must be mirrored follows from the query: the analysis reads adjacency and fault state, so the replica ingests topology and status changes and no other process value, which makes the events counted by f_{update} a consequence of the analysis rather than of how much data the plant produces. Their rate is not thereby a property of the analysis at all: it is set by how often those properties change in the plant and by how the replica is fed, which is

why the operating point below is stated as an assumption. *View maintenance* is used in the sense of the literature on materialised views [67]. The cost comprises synchronisation per change event (c_{sync}, class P) and the amortised export run at a trigger rate η (A), the latter composed of the query evaluation (class A), the serialisation of Equation (8) (class E) and the write into a repository (class A). Here, η is the number of export runs per change event, and Section 5.10 bounds what it implies for the staleness of the view. The write is charged because the exported submodel exists in the exporter's memory and must reach a repository before scenario C can read it, whereas every baseline retrieval is charged a full round trip; omitting it would price the two sides in different currencies.

Resources and latency kept apart. The numerator of Equation (9) is processor work, charged at elapsed times that overstate it. The denominator is the resource time a read of C saves against A3, the processing on the server and client and the occupancy of the link, without the time spent waiting for round trips, which occupies neither machine nor link. At base parameters a read of C occupies 7.30 ms of resource time against 27.83 ms for A3, so the saving is 20.53 ms and $r^* = 0.32$. Of the savings, 16.67 ms is link occupancy and 3.86 ms processing, so counting processing alone gives 1.72. The latency gain is a benefit of its own, 25.91 ms per read, and is not traded against work. Counting it instead, 1 ms of client waiting against 1 ms of server work gives 0.26, and counting only the server term gives 2.36. The conclusion survives all four readings, since the operating point of the case study is about 3600 reads per change event. Under every reading r^* is a marginal criterion: it prices the work of one change event and one read cause, and not the fixed cost of operating a graph store, a change-data-capture path and an export service beside the existing infrastructure. That cost is a deployment question and is not modelled here.

Several consumers and several views. The maintenance is paid once, however many consumers read the view, so with k consumers reading at the same rate, the threshold per consumer falls to r^*/k. Several views need two cases, which Table 8 separates. Read by different consumers, v views share the synchronisation of the replica, and each adds only its own exports, provided the further analyses read no property the first one does not mirror, so the threshold per view falls with v, from 0.32 to 0.09 at five views. Needed by one consumer over the same topology, the views compete with a baseline that retrieves the topology once and evaluates every analysis locally, whereas the pipeline serves one view per analysis. The resource time the pipeline saves then shrinks with v, the threshold rises to 0.55 at two analyses, and the advantage vanishes at $W_{\text{A3}}/W_{\text{C}} = 3.81$ analyses.

Why a break-even ratio. Alternative criteria were not tested empirically, and the choice rests on three properties. The ratio is the point at which the cost rates of the two architectures are equal, maintenance per change event plus work per read against work per read, so any criterion that compares the two cost rates linearly yields the same threshold. It is dimensionless, and the decision it supports depends on the read and change rates only through their ratio, whereas their absolute values are assumptions of the case study. And it is the criterion by which materialised views are selected in data warehouses, where views are chosen by the query work they save [85] and, once their maintenance is counted, a view pays when the frequency of the queries it answers, weighted by the work it saves, exceeds the frequency of updates weighted by its maintenance cost [86], Equation (9) being the case of a single view. Economic criteria such as the return on investment or the total cost of ownership would need monetary values and the fixed cost of a graph store, a change-data-capture path and an export service, which this paper does not model, and criteria based on queueing or on response-time targets would need arrival processes and service levels the case study does not define. The ratio is therefore a technical comparison of two architectures and not an economic model.

Table 8. Break-even ratio in resource time for k consumers and v views, at base parameters. One view per consumer: each view is read by its own consumers, and the views share the synchronisation of the replica, which holds while they read no further property. All views per consumer: every consumer needs all v views, analyses over the same topology, against a baseline that retrieves the topology once per consumer and evaluates them locally.

Consumers k	One View per Consumer			All Views per Consumer		
	$v = 1$	$v = 2$	$v = 5$	$v = 1$	$v = 2$	$v = 5$
1	0.323	0.177	0.090	0.323	0.550	none
2	0.162	0.089	0.045	0.162	0.275	none
5	0.065	0.035	0.018	0.065	0.110	none
10	0.032	0.018	0.009	0.032	0.055	none

Two of the assumptions in Equation (9) shape it most, namely η and c_{query}, so the break-even ratio is reported over both rather than as a single number (Table 9). A third dependence is less obvious. The denominator is itself derived from the timing assumptions, so whatever narrows the gap raises the threshold. In the latency reading, on the fastest link examined, the gain falls from 25.91 ms to 10.90 ms and r^* rises from 0.256 to 0.609. At the two-factor corner of Section 5.10 it reaches 1.10, and the largest value any pair of the eleven produces is 1.20, on the fastest link at four times the fault rate. The threshold therefore does not stay below unity everywhere in the region examined. Both corners lie on the fastest link, where the retrieval cost the pipeline saves has largely disappeared on both sides, and a high fault rate narrows the gap further. In resource time the dependence is stronger, since link occupancy is most of what a read saves. The fastest link alone raises r^* to 1.20, and where it meets four times the fault rate, a read saves only 0.26 ms of resource time, and r^* reaches 25.5. Every value stays far below the operating point of about 3600 reads per change event.

Table 9. Break-even ratio over the export policy η and the query cost, in resource time ($W_{\text{A3}} - W_{\text{C}} = 20.53$ ms) and in latency ($T_{\text{A3}} - T_{\text{C}} = 25.91$ ms), with $c_{\text{sync}} = 6$ ms. The lowest and the highest query cost are the two Cypher figures Sonnenberg et al. [43] report, the lowest also the one the end-to-end run measures. The middle one lies between them and is not a measurement of a graph query. The column $\eta = 0.1$ of the first row is the base case of this evaluation, an export that batches ten change events, in which $c_{\text{maint}} = 6.64$ ms. Where a change must not wait for further change events, as for the fault detection of the case study, the column $\eta = 1$ applies.

c_{query}	Export Runs η			
	0	0.1	0.5	1
Resource time				
3 ms, simple value lookup	0.292	0.323	0.448	0.603
27 ms, an intermediate value	0.292	0.440	1.032	1.772
67 ms, query with parsing	0.292	0.635	2.006	3.721
Latency				
3 ms, simple value lookup	0.232	0.256	0.355	0.478
27 ms, an intermediate value	0.232	0.349	0.818	1.405
67 ms, query with parsing	0.232	0.503	1.590	2.949

The first column is a bound rather than a policy: with $\eta = 0$, the graph is kept synchronised, but the submodel is never refreshed, which fixes the smallest value c_{maint} can take. In the base case the pipeline repays its view maintenance from roughly one read per three change events onwards. The insensitivity to the export policy is real but conditional. Under the 3 ms reading, the threshold rises only to $r^* = 0.60$ even at $\eta = 1$, because the export then costs about as much as a synchronisation event, and under the

two higher readings, both thresholds pass unity at $\eta = 1$. What survives all three is the weaker statement that the pipeline repays from at most about four reads per change event, still three to four orders of magnitude below the operating point. The serialisation term stays small because it scales with the size of the *result* and not of the topology, which is the selectivity assumption of Section 4.2.

The installation studied here sits far to the right of r^*, on three rates that are themselves class A assumptions: adjacency is altered during maintenance and retrofit, of order one per week; fault state changes at the rate at which drives fail, for 200 drives of order one per hour; and the assistance system evaluates the failure condition on every production cycle, so f_{read} is of order one per second. Reads therefore exceed update events by three orders of magnitude. This presupposes that the replica ingests change events rather than periodic writes, since a path mirroring every cyclic status write would raise f_{update} to the cycle rate of the whole fleet and lose the margin; the deployment must supply change data capture, which is what the event-driven pipelines of Section 2.2 provide.

5.7. Sensitivity and Robustness

Since all timing parameters are assumptions, the relevant question is not the value of the ratio at the base point but its behaviour across the parameter space.

Influence of the individual parameters. Ten of the rows marked in Table 5 are swept here as eleven factors and five more below. The ranking is bandwidth (ratio 4.04 to 1.45), the median round-trip time (2.78 to 1.48), the fault rate (2.22 to 1.34) and client and server concurrency (2.79 to 1.95). Then follow the dispersion of the round-trip time, the fixed server cost per request, which alone spans 2.02 to 1.63, and the server time per element, of which the first two differ in the width of the range they span by about the seed dispersion of Section 5.10 and are therefore not ordered between themselves. Client-side parsing, the serialisation rate and the page size come last, all small, the last because the four faulted drives fit any page the API offers. The ratio stays above unity across all eleven, with a minimum of 1.34 at four times the fault rate. It is largest on narrow-band links, where the volume dominates, and it decays at high round-trip time for a structural reason: A3 waits on the maximum of two draws and C on one, so as latency dominates, both totals the ratio approaches the ratio between those two order statistics.

Bounded server concurrency. Reducing the server to serial processing raises the ratio to 2.79. Raising the server beyond two workers changes nothing, because the effective concurrency of a round is the smaller of the two limits, and from two workers onward, the round already holds both of its requests in one wave. Table 6 therefore reports the setting most favourable to the baseline.

Dependence on the assumed element counts. Varying the properties per node between 2 and 20 and the properties per drive submodel between 5 and 50 yields latency ratios between 1.61 and 2.16 (median 1.94) and volume ratios between 3.1 and 7.9 (median 5.45). Varying the properties an adjacency element carries, from 0 to 8, moves the latency ratio between 1.64 and 2.81. The node degree enters on both sides, since it sets how many adjacency elements the plant holds and, through the closure condition, how many the result holds; running it from 1 to 4 moves the ratios between 1.62 and 2.48 and between 5.0 and 6.6. That the degree acts on both sides is why its volume interval is the narrower and the two property counts, which move one side each, the wider. Neither ratio is a fixed quantity, and neither is reported as one; what the variation supports is that both stay above unity across the space examined.

Three further variations. The value-only serialisation (`$value`) applied to both scenarios reduces A3 to 103.4 KiB and C to 9.01 KiB, moving the latency ratio from 1.95 to 1.67 and the volume ratio from 5.92 to 11.5; the two move in opposite directions, and

the serialisation is no substitute for the approach because its compression is bought by discarding the annotation a consumer without prior knowledge of the schema depends on. Under a cold start, in which the first round pays a Transmission Control Protocol and a Transport Layer Security handshake, the ratio falls to 1.37; the handshake affects scenario C disproportionately because C issues one request, and since the model charges one handshake per scenario rather than one per connection, 1.37 is a lower bound. Raising the over-fetch factor from 1.0 to 5.0, so that the value filter returns five times as many drives as the analysis needs, moves the ratio to 2.12, so the assumed value is conservative.

Scaling with the size of the installation. Varying the drive count from 50 to 3200 moves scenario C from 23.5 ms to 103.1 ms and A3 from 33.9 ms to 439.7 ms, so the ratio runs from 1.44 to 4.27. The fault rate is held constant here, so the answer grows with the plant and both payloads are linear in it; what moves the ratio is the fixed round-trip floor, which dominates both totals at the small end. The advantage is therefore a function of plant size, and at the small end it is modest.

One quantity at a time. Of the four quantities that set the result and the request structure, only the number of controllers holding the topology is an assumption of its own. Selectivity, request count and result size are derived from the plant size, the fault rate, the node degree and the page size, and the sweeps above vary them only through those drivers. Table 10 varies each on its own, while everything else stays at the case study. The controllers barely matter for A3, which retrieves all topology submodels in one request, and adds one request each to A1. The result size is varied for C alone at a fixed baseline: C pays off up to between 80 and 160 elements, four to eight times the base case, the horizontal counterpart of the crossing in the vertical case. Selectivity, varied through the plant size at a fixed result, is separated from a growing answer and drives the ratio from 1.27 at $s = 0.24$ to 15.4 at $s = 0.003\,75$. The request count of the baseline's single round raises the ratio by about a fifth up to the effective concurrency of eight and steeply beyond it, since the round then needs a second wave. The result size and the request count could not vary independently in operation, since both follow from the plant and its fault state, which is why their variation is stated as decoupled.

The publication cost of the export is small without being negligible: at zero the break-even ratio would be 0.309 in resource time and at 23.5 ms, of the order of a retrieval over the link the comparison assumes, 0.423 against the base 0.323, and in the latency reading 0.245 and 0.335 against 0.256.

Table 10. Each of the four quantities varied on its own, everything else at the case study. Ratios are A3 over C, and A1 over C in the row that names A1.

Quantity	Variation	Range	Ratio to C	
			Latency	Volume
Controllers holding the topology, A3	direct	1 to 32	1.95 to 1.98	5.91 to 6.17
Controllers holding the topology, A1	direct	1 to 32	43.6 to 48.8	17.8 to 18.5
Result size m	C alone, baseline fixed	5 to 640	2.26 to 0.29	19.8 to 0.20
Selectivity s	plant size at a fixed result	0.24 to 0.00375	1.27 to 15.4	1.67 to 90.8
Requests in the round of A3	bytes spread over parallel requests	1 to 16	1.92 to 3.83	5.92

5.8. Partitioning the Topology at Submodel Level

The topology accounts for 95.9% of what A3 transfers, and the question is whether that share is a property of the language or of how the plant happens to be modelled. No condition selects adjacency elements by endpoint (Assumption 3), but a query may return

some Submodels and not others. If the topology were split across many Submodels instead of one per controller, a client that has already located the faulted drives could name only those covering them, and since the analysis looks for *adjacent* failures, the drives it finds are clustered, and few groups are touched. That is the opposite corner from the sparse-fault bound of Section 5.3, at which C is still priced, and it is chosen because the objection is strongest there. Scenario A5 models it, and Table 11 reports the outcome.

Table 11. Effect of partitioning the topology and selecting at Submodel level, horizontal problem. The Rounds column counts dependent rounds.

Scenario		Rounds	Payload	Median [ms]	Ratio to C
					Latency
A3	one Submodel per controller	1	244.9 KiB	53.22	1.95
A5	20 groups, one touched	2	22.08 KiB	48.91	1.79
A5	20 groups, two touched	2	34.13 KiB	50.38	1.84
A5	200 groups, eight touched	2	22.15 KiB	48.84	1.79
C	consolidated submodel	1	41.36 KiB	27.32	1.00

The share is reducible, and the advantage survives. Partitioning cuts the volume by an order of magnitude, from 244.9 KiB to 22.15 KiB at the finest partition, below what scenario C transfers. Latency does not follow: it falls only to 48.84 ms, so the ratio to C stays between 1.79 and 1.84 against 1.95 for A3, and an eleven-fold cut in volume cuts latency by under a tenth. The reason is the one Proposition 1 gives: a client cannot name the groups until it knows which drives failed, so the two retrievals are no longer independent and the single round becomes two, the second costing a further 16.54 ms of round-trip time, which is most of what the smaller payload saved. The dependency between rounds is what the language imposes and what no partitioning removes, whereas the payload is a property of the plant model and can be engineered.

Two further costs appear in no column. The partition has to exist, and a consumer cannot impose it, since how a plant is cut into Submodels is decided by whoever models it; and the partition, like a cache, is bound to one question. The volume figures of Table 6 therefore hold for the topology as modelled here and are not a lower bound over all models. What survives every partition examined is the round structure, which is the more fundamental of the two.

5.9. Demonstration of Consumability

The preceding sections quantify what the pipeline costs and saves. That the artefact can be used is a separate question, answered here by a working consumer. The consumer reads the generated submodel and reports whether a functionally critical failure pattern is present, and its construction is deliberately restrictive: no graph database, no query language, no graph library and no AAS library, reading the JSON serialisation directly. It contains no hard-coded `idShort` paths, elements being selected by `semanticId`, and it knows nothing of the plant structure, the drive designations or the language in which the producing system labels its elements. What it knows are the `semanticId` values of Table 3 and the `DerivedStatus` vocabulary, precisely the precondition semantic interoperability is meant to establish. The same consumer was applied to four representations of the same result (Table 12).

Table 12. Consumability of four representations of the same analysis result. Sizes are the compact UTF-8 serialisations.

Case	Representation	Size [B]	Outcome
1	pipeline output, *Normal* format	12,012	interpreted
2	identical, with German `idShort` values	12,045	interpreted, identical result
3	identical, in `$value` serialisation	2617	not interpretable
4	raw graph query export (Cypher result)	912	not interpretable

Case 2 is the substantive test: not one `idShort` coincides with case 1, since `Nodes` becomes `Knoten`, `ObservedStatus` becomes `BeobachteterZustand` and `Drive_1207` becomes `Antrieb_1207`, yet the unchanged consumer returns the same result, including the same critical pair. Element resolution is therefore semantic rather than nominal, and a consumer operating on `idShort` paths would fail. Cases 3 and 4 are negative controls, and both are *more* compact than the representation that succeeds. Case 3 is smaller by a factor of 4.59 but omits `semanticId` and `modelType`, so its compression is bought by discarding exactly the annotation the consumer depends on. Case 4 is the most compact of all and contains everything needed to reconstruct the finding, provided the reader knows the conventions of the producing system: that `assessment` carries the analysis outcome, that `start` and `end` refer to `elementId` and that `err` denotes a fault state. Those conventions appear nowhere in the document, and supplying the first two of them in a machine-resolvable form that a registered template could make common is what the template contributes: machine-resolvable *identity* for the elements of the finding. One thing it does not supply: the values stay the producer's, `ObservedStatus` carrying `err` unchanged, so only `DerivedStatus` is read against a vocabulary the template fixes. The shells of the affected assets, by contrast, are reached through the `AssetReference` of every node. What cases 3 and 4 establish is therefore what each representation withholds, not that the consumer could not have been written for them: a consumer built for the Cypher result would read case 4 and fail on case 1. Case 2 shows the consequence a `semanticId`-keyed consumer inherits from the metamodel: it reads two documents that share no `idShort`.

Three limitations bound the demonstration. Conformance is established separately by the two-stage procedure of Section 4.7 and is not asserted by the consumer. The four representations above are generated from hand-written query records of a small instance of three nodes and two edges, of which only two nodes are faulted, the third entering the result graph solely as the endpoint of an edge, which is what closure requires and why the second edge is uncritical. The end-to-end run of Section 4.7 repeats the evaluation of the first representation on the result of the executed query, read from the BaSyx server over HTTP. For each of the three fault configurations, the consumer reports the critical pairs the oracle predicts and names the shells of the affected drives from their `AssetReference`, and the run resolves every `AssetReference` of the base case to its shell on the server, which needs no convention of the producing system. Interpreting the submodel of the base case, 62.7 KiB, as the server returns it, takes the consumer 0.41 ms. Finally, the `semanticId` namespace is provisional, and submitting the template to the IDTA would be the obvious next step.

5.10. Threats to Validity

Construct validity. The model is an analytical cost account and not an observation. It cannot detect effects it does not represent, and its absolute values claim no correspondence to any particular installation; it supports statements about *relations* between architectures under stated parameters. Equation (9) charges maintenance work against the resource

time a read saves, and Section 5.6 reports the readings in latency and in processing alone beside it.

Internal validity. Contention is not modelled: requests exceeding the effective concurrency form additional waves but incur no queueing delay, and within a wave the server term is the mean over the parallel requests rather than the maximum, which understates the baseline's server time by 3.9 ms. Both favour the baseline; drawing the round-trip times of A3's two parallel requests independently does not, and on the path they share the ratio is the 1.75 of Section 6.3. Server time is linear in the number of elements, whereas real persistence layers exhibit discontinuities at cache and index boundaries; the sensitivity analysis covers the range but not the structure.

Calibration of the server model. The end-to-end run measures the export side. The query costs 2.98 ms, converting its records 0.14 ms, serialising the document 0.87 ms and writing it 1.21 ms, against 3 ms, the 0.386 ms of Equation (8) and 3 ms in the model. The document of the base case is 61.2 KiB as written rather than the 41.4 KiB of the model, because every unresolved property carries a qualifier and the `elementId` of Neo4j makes every `idShort` and `OriginId` longer, which is also part of why its serialisation exceeds the slope of Equation (8). The payload model prices resolved properties and identifiers as short as a business key, the remedy Section 6.5 names for the revision-bound `OriginId`, and stays the base case. Priced instead at the 62.7 KiB, the server returns for the document of the run; scenario C takes 29.29 ms. The latency ratio then falls from 1.95 to 1.82 and the volume ratio from 5.92 to 3.90, and the break-even ratio rises from 0.32 to 0.36 in resource time. With gzip the volume ratio falls from 2.67 to 2.33 and the latency ratio only from 1.40 to 1.39. Reading the consolidated submodel back takes 1.15 ms, whereas the model's server term for the same document is 3.81 ms. Since that deviation is substantial, the three server parameters were calibrated against BaSyx with its in-memory back-end on one Docker network: submodels of 10 to 2000 properties in two families of different size, per element give a fixed cost of 0.28 ms per request, 0.65 µs per element, and a serialisation rate of 180 MB s^{-1}. With them, scenario A3 takes 47.89 ms and C 24.21 ms, a ratio of 1.98 against 1.95 with the assumed values. Five repetitions on a later day, each on freshly started servers after 2000 warm-up iterations, passed every test and measured 3.03 ms to 3.22 ms for the query, 1.30 ms to 1.47 ms for the write and 1.21 ms to 1.27 ms for the read, and with their own calibrations the ratio is 1.97 to 1.98. Query, write and read of the run quoted here lie below these ranges, on servers that earlier runs had already warmed. The assumed values remain the base case, because an in-memory server without authentication is a lower bound for a repository with persistence and access control, and the calibrated ones show that the headline does not depend on them. A measurement inside one Docker network does not calibrate the network, so round-trip time and bandwidth remain assumptions. Requests issued from the host of the containers passed a port forwarding of Docker Desktop that, with the ports of the servers published on all interfaces, added more than 40 ms to every write, so all measurements were taken inside the Docker network.

One analysis question, and one topology partition. Scenario C is produced by one export query, so a second analysis question requires a second view, and Section 5.6 prices how the threshold moves with the number of views and of consumers. Scenarios A4, A5 and A6 carry the same debit and are set aside on the grounds Sections 5.4–6.3 give rather than on cost grounds, which matters because all three beat scenario C somewhere: the caching client is faster and also transfers less, the finest partition transfers less, and the encoded plant is faster on the pattern of the case study. Only the cache is borne by the party that benefits, the partition and the encoding falling to the plant modeller. This is the strongest objection to the comparison, and it is answered by argument rather than by arithmetic.

External validity. The case study is one class of installation. Of the quantities entering the break-even, η and c_{query} are assumptions reported over their range in Table 9, over which r^* runs from 0.292 to 3.721 in resource time and from 0.232 to 2.949 in latency. Composing the highest of these values with the two-factor corner below takes them to 13.83 in resource time and 12.68 in latency, whereas halving and doubling c_{sync} moves the base value only between 0.177 and 0.616 in resource time and between 0.140 and 0.488 in latency. What distinguishes c_{sync} is not the size of its effect but that it is the only quantity adopted from another deployment, so its range is bounded by nothing reported here. Since $r^* = (c_{\text{sync}} + \eta\, c_{\text{export}})/(W_{\text{A3}} - W_{\text{C}})$ is affine in it, and so is its latency reading, a reader can rescale exactly without re-running the model.

Freshness. Scenario A3 reads live data, and scenario C reads a view that lags behind the source. Setting $\eta = 1$ equalises the number of updates and not the age of what is read. The staleness of the view a consumer reads is the time for which the oldest change not yet reflected in it has been waiting. With one export per change event it is bounded by the publication delay $d_{\text{pub}} = c_{\text{sync}} + c_{\text{query}} + c_{\text{serialise}}(m) + c_{\text{write}}$, provided that successive change events lie at least $c_{\text{sync}} + c_{\text{export}}$ apart. With the values of the model, the bound is 12.39 ms, and for evenly spaced events, the condition holds below about 81 change events per second. With the export, the end-to-end run measures for the base case, 5.20 ms for the query, the conversion of its records, the serialisation and the write, the bound is 11.20 ms. Measured directly, from a status change in Neo4j to the first read of the export it triggers, the delay is 4.10 ms as the median over 100 change events of one drive that alternately fails and recovers, whose result graph is smaller than that of the base case or empty and whose source is Neo4j itself, so that no synchronisation enters. The five repetitions measured 4.55 ms to 4.97 ms. Bound and delay lie far below the read period of about 1 s in the case study. With fewer exports than change events, the bound does not hold. At $\eta = 0.1$ a change reaches the view only with the next export, after up to nine further change events, and since fault state changes about once per hour in the case study, that can take hours. At $\eta = 1$ the break-even ratio is 0.60 in resource time under the 3 ms query cost, and under the two higher query costs it passes unity. Short-lived faults can be missed, in three distinct cases. A fault shorter than the read period can be missed by both architectures, since both sample the plant at the rate of the assistance system. With one export per change event, the published view is the live state shifted by the publication delay, so the sequence of exports skips no fault, which appears later but with the same probability of being read. With fewer exports than change events, as in the base case $\eta = 0.1$, a fault that begins and ends between two exports never reaches the published view, although the replica has seen it. Triggering the export on the change events that can alter the result of the query, rather than on every tenth event, closes this case, and retaining a finding until it is acknowledged would close it as well but is a different analysis.

Single factors and pairs. The eleven factors ranked in Section 5.7 are swept singly, so the single-factor region examined is a cross-through parameter space; every pair was searched in addition, at both ends of both ranges. The least favourable pair, the server cost per element with the fault rate, takes the ratio to 1.07, and the pair narrowing the absolute gain most, bandwidth raised tenfold with four times the fault rate, takes the gain to 5.64 ms at a ratio of 1.18. Bandwidth raised tenfold with a round-trip time of low dispersion, a fast and steady link and the two-factor corner of this section takes the ratio to 1.25. A3f and multiplexing give 1.24 on the base link and 1.04 at the two-factor corner and 0.82 there at fourfold faults, where that client also transfers less than scenario C, the volume ratio falling to 0.78.

Reproducibility, and the precision it does not buy. All results derive from a fixed seed and are reproducible, but a median over 1000 runs is a sample statistic: over 40 seeds

A3's latency has a standard deviation of 0.41 ms and the ratio 0.021, a dispersion r^* inherits in proportion, so their last digits are unresolved, and the chosen seed lies on the side unfavourable to the pipeline. The location statistic matters more than the seed: on means rather than medians the ratio is 1.88, since A3 waits on the slower of two draws and scenario C on one. The gap is almost unmoved and r^* with it: in resource time, which leaves out the round-trip draw, the only random term, it stays at 0.323, and in the latency reading it moves to 0.254. The choice, therefore, bears on the headline factor and not on the amortisation. The model, parameters, experiments and the audit log documenting the defects found and corrected during development are provided in the material described in the Data Availability Statement.

6. Discussion

6.1. What the Results Support

Section 5 yields three findings that differ in how strongly they are supported, and separating them matters more than repeating them. The comparison against an optimised baseline is the weakest, because every timing quantity entering it is an assumption and the ratio moves between 1.44 and 4.27 over the plant sizes examined. It moves further still once the baseline optimisations conceded above are composed rather than taken one at a time. A client that filters what each object carries and multiplexes its two requests onto one connection reaches 1.24, and 0.82 at four times the fault rate in the corner of Section 5.10, and with compression it reaches parity at base parameters (Table 7). The headline is what the comparison yields against the optimised client on the base link, not a bound: among clients holding no replica and reading the plant model of Table 4, the ratio falls to 1.24 there, to 1.02 with compression, and below unity at that corner, where such a client also transfers less than the pipeline. Nor is latency the only scarce resource at the read rate of Section 5.6: the saving is a small fraction of a production cycle, and what the comparison establishes at the base parameters is a reduction in transferred volume, which is a statement about those parameters and not about the corner. What survives all of that is the round structure, not the ratio. The location of the difference is better supported because it survives the idealisation of zero server time and because Section 5.8 shows that the payload share can be engineered while the round structure cannot be. The location of the limit is the best supported of the three, since Proposition 1 alone settles it, with no appeal to Section 5.8.

Where the pipeline offers an advantage. Table 13 gathers the comparison. Against clients that hold no replica and read the plant model as it stands, the pipeline is faster by a factor of 1.95 than the value-filtered client and at parity, 1.02, with the fully composed one under compression, and at base parameters it transfers less in every case, from 5.9 times less than the uncompressed value-filtered client to 1.1 times less than the fully composed one with Brotli. It is not faster than a client that keeps a private replica of the topology, nor than a plant whose model carries the neighbour identifier, and it transfers more than the finest partition, which it outruns. Its advantages over those lie outside the ratio. One maintained replica serves every consumer, whereas a cache is maintained per consumer and is sound only while the question stays fixed. The pipeline requires no modelling convention of the AAS plant model, whereas partition and encoding are decided by whoever models the plant and fixes the questions they can serve. Its round structure does not grow with the depth of the pattern, and the artefact carries the identity of the query, the revision of the source and its validity. It does not pay for non-selective analyses, for results beyond about a hundred elements, for a single consumer that can cache, or for links fast enough that retrieval costs little on either side.

Table 13. Where the pipeline offers an advantage, at base parameters. Ratios are client over C, the last numeric column with gzip granted to both.

Client	Ratio to C			Beyond the Ratio
	Latency	Volume	Latency, Gzip	
A3, value filter	1.95	5.92	1.40	C needs one round at any depth of the pattern
A3f and multiplexing	1.24	2.86	1.02	as for A3
A4, topology cache	0.86	0.24	0.94	maintained per consumer, sound only for a fixed question
A5, finest partition	1.79	0.54	1.95	fixed when the plant is modelled
A6, plant-side encoding	0.87	0.29	0.95	fixed when the plant is modelled, deeper patterns need further rounds

6.2. Relation to Existing Work

The results permit a sharper statement of the position taken in Section 2.4. Scenario B quantifies what a graph back-end buys: it accelerates the queries that AAS-QL can express and leaves untouched those it cannot. An event-driven mirror and the export pipeline are complementary: the mirror maintains the graph that Listing 1 operates on and the export pipeline returns the result to consumers that possess no graph tooling. Neither substitutes for the other, because they solve inverse problems. Against the approaches that produce AAS content from an engineering artefact [49,50] or by inverting their own prior ingestion [48], the distinction is what determines the extent of the export: those are structure-preserving and fixed by a source document, whereas the present one is query-defined and enriching in the sense of Definition 2. The mapping rules of Section 4.6 are the vehicle of the contribution and not the contribution itself, which is why, taken on their own, they would amount to a format conversion.

6.3. Scope of the Comparison

The comparison holds the implementation technique fixed and varies only what the standardised interface allows a client to express. Server-side query planning, index tuning, transport compression, connection multiplexing and application-level caching are absent from the base scenarios, since none change which questions AAS-QL can answer and admitting some but not others would turn an architectural comparison into a contest between implementations. None of the five is neutral between the scenarios; however, since they help a client issuing many requests more than one issuing a single small request, the last three are priced as variants in Table 7. Multiplexing the two requests of A3 onto one connection replaces the order statistic of Table 6 with a single round trip, taking A3 from 53.22 ms to 47.85 ms and the ratio from 1.95 to 1.75. Were transport compression neutral between the two, it would act as an increase in bandwidth, and the sweep of Section 5.7 would bracket it. It is not neutral. The baseline carries two hundred near-copies of one adjacency element and the pipeline one heterogeneous document, so a general-purpose compressor removes more of the former, 61-fold against 25-fold with gzip. Priced on the measured bodies in Table 7, it takes the latency ratio to 1.40 and the volume ratio to 2.67, and composed with `$filters` and multiplexing to 1.02. It does not touch the round structure.

One technique deserves explicit treatment, because it is the one a reader is most likely to raise. A client could hold the plant topology in a local cache and retrieve only the status values on each cycle, since the topology changes rarely. Modelled as scenario A4, this removes the entire difference in the steady state and more, reaching 23.5 ms against 27.32 ms, a ratio of 0.86: a client that keeps a private replica of the topology is faster than the pipeline. Two considerations place it outside the comparison rather than inside it.

The first is what the technique concedes: a cache does not extend the query language but evades the inability to express a traversal by keeping a private replica and performing

the traversal outside the standard, which restates Proposition 1 rather than refuting it, and a consumer that must replicate the plant model before it can ask a question is in the position the AAS was introduced to remove. The second is symmetry. The same technique is available to the consolidated submodel and more cheaply, since that submodel is a single document carrying `GeneratedAt` and `ValidUntil`; granting a cache to both is not modelled and would add to each side an assumption the standard does not underwrite, since the AAS API defines no change notification. None of this argues that caching is bad engineering; it argues that caching belongs to the implementation of a consumer and not to the expressive power of the interface. A client that revalidates the cached topology with an HTTP validator each cycle is this scenario without its staleness, at the price of a further request each cycle and of a technique the AAS API does not define. What a cache does not do is scale: a replica held by a consumer must be maintained once per consumer, whereas the pipeline maintains one and publishes to every consumer, which is the cost side of the argument Section 1.2 makes about bilateral agreements.

6.4. On Unidirectionality

A recurring expectation of transformations between metamodels is that they be bidirectional. Proposition 2 shows why that cannot be met here and why the restriction is a property of the construction rather than of the implementation: the analysis submodel is a condensing projection, everything outside $N_q \cup E_q$ is absent from the image, several source graphs share one image, and a left inverse therefore does not exist.

The class of problem is known. Czarnecki et al. [72] state the operational case: given a target specified as a set of views over a source by a set of queries, the view update problem is to determine whether updates to the target can be intercepted and applied to the source unambiguously, such that re-running the queries regenerates the updated target exactly. The pipeline produces a target of exactly that form, and Proposition 2 states that the required unambiguity does not hold. Stevens [71] shows that any transformation language adequate for model-driven development must express non-bijective transformations, leaving a choice over backward propagation, and Hidaka et al. [73] classify the strategies that resolve it. A write-back channel is therefore not a matter of effort but of selecting an update policy, a design decision external to the transformation. This is the expected behaviour of a materialised view, which remains read-only until such a policy is stated. What is gained is that the semantics of the projection remain well defined; what is given up is write-back, which publishing an analysis result does not require.

6.5. Limitations

The approach is for selective analyses, and the vertical problem shows where that ends. Section 5 evaluates the horizontal problem. The vertical one, in which the pattern joins sibling leaves of a deep hierarchy, is governed by the same mechanism and yields the same qualitative result at the base parameters: the `$sm#semanticId` condition collapses five rounds into one and the value filter restricts the status retrieval, taking A2 from 3459.9 ms to A3's 439.3 ms against 274.8 ms for C, a latency ratio of 1.60 and a volume ratio of 1.75. What neither condition reaches is again the adjacency.

The vertical case differs in one respect that bounds the whole approach. Its baseline pays a structure cost fixed by the number of leaves, whereas scenario C pays for the result graph, which grows with the number of faulted leaves, so the two curves cross. Sweeping the eleven factors of Section 5.7 on the vertical pair takes the ratio below unity for two of them, the lowest value being 0.60. The node degree, which those eleven hold fixed and which sets the result graph, while the containment fan-out sets what the baseline transfers, moves the ratio between 0.92 and 2.53. At base values elsewhere, the two architectures

cost the same at a fault rate of 0.0362, 1.8 times the assumed 0.02. Above that rate the consolidated submodel is slower, its payload is larger from 0.0401, and no read rate repays it. The horizontal pair crosses nowhere on the single-factor sweep, because there the structure the baseline transfers is the whole topology and the result graph stays a small fraction of it; against the filtering, multiplexing client, it crosses at 0.0475. The approach is therefore for analyses that are *selective*: where the monitored pattern holds of a large share of the plant at once, the artefact is no longer small relative to what a client would have fetched, and the argument of this paper does not apply.

On holding the topology twice. The replicated graph is a second copy of the plant model and its maintenance is the price of the approach, expressed by Equation (9). Section 5.6 gives the threshold, the region over which it moves and why a break-even ratio decides when the second copy pays, and what matters here is that the price is declared rather than assumed away.

The assumption that one node is one asset. Rule R1 assumes that a node in the graph corresponds to a logical asset. In richer ontologies a single asset may be distributed over several nodes, and relationships may themselves be assets. Recognising such constructions automatically would require semantics-aware transformation rules beyond the scope of the rules given here, and Table 2 lists them as not covered.

Domain semantics are bounded by the source graph. The three-tier cascade of Section 4.5 resolves domain semantics from the graph where the data permit it. Where no dictionary reference is available, a deterministic URI is generated and marked by a qualifier. The analysis semantics remain well defined in all cases, but an outside consumer cannot interpret the domain attributes of a graph that carries no classification. That is a property of the data source, and the model declares it rather than concealing it.

The payload grows with the size of the result. The pipeline consolidates the result of one query into one submodel. For loosely filtered analyses this can lead to allocation problems or timeouts on the client. Pagination mechanisms for the export remain to be added, and the degree and fault-rate sweeps of Section 5.7 indicate where the limit becomes relevant.

Origin identifiers hold for one revision. Every node refers to the shell of its asset (rule R1), so a consumer reaches the affected assets without a convention of the producing system. The reference is only as good as the key the source graph holds, and the export is refused when a node carries neither a shell identifier nor a `globalAssetId`. The `OriginId`, by contrast, identifies its element only in the revision `SourceGraphRevision` records, since Neo4j guarantees an `elementId` only within one transaction and may reuse the identifiers of deleted elements [70]. A consumer that follows an `OriginId` into the source graph must therefore check the revision first, and a business key held in the graph would lift that restriction where the plant model has one.

A model with a measured export side. Sections 5.1 and 5.10 state the epistemic status of the evaluation and its threats. The export side is measured end to end and the server parameters are calibrated against one implementation, while the retrieval of the baseline remains modelled and the network quantities assumed. The query cost is carried over its full range rather than fixed at the measured value, which holds for one database size.

Faults between two exports. With fewer exports than change events, a fault that begins and ends between two exports never reaches the published view, and one that persists reaches it only with the next export (Section 5.10). The base case admits this, and the remedies are to trigger the export on the change events that can alter the result of the query, or to retain a finding until it is acknowledged, which would be a different analysis.

Computed values are marked twice, for two purposes. Every property rule R4 writes carries the `ValueOriginQualifier` of the IEC Common Data Dictionary with the value

calculated, which the IDTA time-series template lists among the qualifiers an element may carry [23], and every `SubmodelElement` is `Qualifiable` [6], so nothing confines it to time series. A consumer therefore recognises any computed property, not only the `DerivedStatus` the template names. The template still draws the distinction along `semanticId` as well, which Assumption 3 admits as a query condition, unlike a `Qualifier`, so a consumer may select on it instead of reading every element. The qualifier is found by reading, the right trade for a flag on an element already fetched and the wrong one for a distinction to select on.

In the terms of Angles et al. [38] [Section 2.1] deferred two properties of a database mapping until the apparatus was available. The pipeline is not *information preserving* for such a query, by Proposition 2, since everything outside $N_q \cup E_q$ has no image. Every output is a valid instance of the target schema, by Property 1, which is schema conformance and no more; the cascade of Section 4.5 is a separate and stronger property, namely that what the artefact means is fixed by the template and the dictionary rather than by the exporter. The combination is the ordinary one for a materialised view and is not a defect of the construction.

A vendor extension would answer the query in one round. The specification that states the limitation also names the way around it. Where the expressiveness of SPARQL or GQL is required, "a software vendor might additionally extend his service accordingly" [8] [Part 2, Query Language, Limitations]. A repository backed by a graph store and exposing a traversal operator of its own would answer the motivating query server-side, with one round trip, no replica and no export, which is cheaper than anything Section 5 prices. The ground on which it is excluded has to be stated carefully, since Section 3.3 concedes that the template proposed here is bilateral too until it is submitted. Both are bilateral today and differ in what can end that condition. A submodel template is the artefact the IDTA registration process exists for, and once registered, it binds every producer and consumer without further agreement, whereas the specification names a vendor extension as what lies *outside* the standardised interface, so no procedure makes one common. The objection is not that a convention is needed but that this one cannot stop being bilateral, which is a different objection from the one Section 3.3 raises against the `idShort` encoding, namely that an encoding leaves the language untouched where a traversal operator would extend it. The alternative is real, and a deployment may reasonably take it; what it gives up is the property that makes the standard worth having.

6.6. Future Work

Four directions follow.

Submitting the `TopologyAnalysisResult` template to the IDTA would turn a proposal into a referenceable artefact and replace the provisional `semanticId` namespace.

Coupling the export to the event-driven synchronisation of Section 2.2, so that only the change events that can alter the result of the query trigger it, would replace the export on every tenth change event and shift the break-even accordingly.

A bidirectional arrangement is possible, though not by extending the pipeline. Its non-injectivity is a property of the construction and no amount of implementation effort removes it (Section 6.4). What is conceivable is a composition in which the direction from the AAS into the graph is supplied by the approaches reviewed in Section 2.2, whether by event-driven mirroring or by a compiler, while the pipeline developed here supplies the direction back out. Each direction would then be well defined on its own, and no inverse of P_v would be required. Such a composition has neither been implemented nor tested here, and its behaviour under concurrent updates from both sides is an open question rather than a straightforward integration. It becomes of practical interest in exactly the setting sketched

in Section 1.2. Should the industry proposal of carrying the Digital Product Passport in AAS submodels [17,18] prevail under the framework the Ecodesign Regulation establishes [14], such templates would have to be taken into an organisation's own database, and results derived from them published back in the same format, which is one exchange with two directions rather than two unrelated transformations.

Finally, an empirical study on a production installation remains the most valuable addition. The reasons this evaluation is analytical are stated in Section 5.1, and they are conditions rather than objections. A physical measurement becomes possible once a network topology and hardware configuration may be disclosed and once the load of the surrounding traffic can be either controlled or recorded alongside the measurement. Under those conditions the cost model supplies the hypotheses to be tested, and each parameter it classifies as assumed identifies a quantity worth measuring first. Such a study would also address the gap that Huebner and Ihlenfeldt [24] identify for the field as a whole: the dominance of validation over evaluation research.

7. Conclusions

This paper has presented a unidirectional export pipeline that publishes the result of a topological graph analysis as a conformance-checked submodel of the Asset Administration Shell. Its starting point is a limitation of the standardised query language: the endpoints of a `RelationshipElement` are not among the queryable attributes (Assumption 3), and no production follows a `Reference` held in the value of a `SubmodelElement` (Assumption 4). Under the six premises of Section 3.3, a neighbourhood pattern spanning several assets is not expressible in one query, though a client can still evaluate it on data retrieved in bulk, and the cost of that retrieval is what this work addresses. The pipeline is the composition of an analysis query with an export function, and two properties separate it from a format conversion: the extent of the export is fixed by the query rather than by a source document, and the exported model may carry derived properties the source lacks. Non-injectivity, proved for non-exhaustive bound-determined local queries, makes it unidirectional in the sense that no left inverse exists and read-only until an update policy is stated, as a materialised view is; that the source is a mutable graph rather than a document is why the template makes query identity and source revision mandatory.

The evaluation is an analytical cost model whose export side is measured end-to-end against Neo4j and Eclipse BaSyx, and it classifies every quantity as established, published, assumed or derived. Against a client that paginates, parallelises and uses the value filter, with the topology held in one Submodel per controller and its adjacency modelled as a collection, the consolidated retrieval reduces latency at 200 drives 1.95-fold and transferred volume 5.92-fold, and 1.44- and 2.86-fold against one that also trims each returned object. Neither pair is a floor: a client that also multiplexes its two requests reaches 1.24, and 0.82 at four times the fault rate on a fast, steady link, where it transfers less than the pipeline as well; one holding a private replica of the topology is faster (0.86), at the price of keeping the plant model outside the standard and binding that replica to one question, and a plant that carries the neighbour identifier as an atomic value reaches 0.87 without any replica, at the price of fixing what a condition reaches when the plant is modelled. With compression granted to both sides, the volume ratio against the value-filtering client falls to 2.67 with gzip and 1.95 with Brotli, and the fully composed client reaches parity. What survives optimisation therefore matters more than the ratio. Of what the baseline transfers, 95.9% is topology whose adjacency elements no condition selects, and that share is reducible: a finely partitioned plant model cuts the volume below what the pipeline transfers, but not the latency, since naming those Submodels presupposes knowing which drives failed and hence a second dependent round. The round structure is what resists, and no store behind

the interface alters the gap: with the server term set to zero in every scenario alike, the baseline remains at 46.71 ms against 23.62 ms.

The replicated graph is priced as a break-even ratio: in resource time the pipeline repays its maintenance from roughly one read per three change events and from about four with one export per change event at the highest query cost. Composed with a fast, steady link, that corner takes the threshold to 13.8 in resource time and 12.7 in client latency, and at the base export policy on a fast link at four times the fault rate, where a read saves almost no resource time, the threshold reaches 25.5, against some 3600 reads per change event in the case study. The threshold per consumer falls with the number of consumers, whereas one consumer needing several analyses over the same topology loses the advantage from about four analyses on. The advantage is conditional, reversing on the vertical problem above 1.8 times the assumed fault rate. A consumer selecting only by `semanticId` reads the artefact without graph tooling and names the shells of the affected drives from their asset references, on the result of the executed query read from the server as on the demonstration instance, where it is unaffected by the renaming of every `idShort` and fails, as intended, on a value-only serialisation and on a raw graph export of the same result. What carries the interpretability of an analysis result is its annotation, not its volume or its serialisation format. The contribution complements the work that makes the AAS queryable by graph means rather than competing with it: those approaches use a graph as a query engine over AAS content, this one uses the AAS as a publication format for graph analyses, and an installation may reasonably operate both.

Author Contributions: Conceptualization, P.S., L.A.L. and H.A.S.S.; methodology, P.S., L.A.L. and H.A.S.S.; software, P.S.; validation, P.S., L.A.L. and H.A.S.S.; formal analysis, P.S.; investigation, P.S.; data curation, P.S.; writing—original draft preparation, P.S.; writing—review and editing, L.A.L., H.A.S.S., A.B. and M.W.; visualization, P.S.; supervision, A.B. and M.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The article processing charge was funded by the Open Access Publication Fund of SLUB/TU Dresden, which is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number 491177748.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The cost model, the experiment scripts, the payload measurement, the export and compression benchmarks, the submodel template, the reference implementation with its mapping configuration, the example instances, the analysis query, the consumer of Section 5.9, the validation reports and the end-to-end run, comprising the container definitions for Neo4j and Eclipse BaSyx, the generator of the plant graph, the run and calibration scripts and their raw measurements, are openly available at https://doi.org/10.5281/zenodo.23598515, under the MIT licence for the code and CC BY 4.0 for the data. The results were produced with exporter version 1.0.0 and mapping configuration version 1.0.0, whose digest the provenance block of every exported submodel record, and with Neo4j 5.26.8 and Eclipse BaSyx aas-environment 2.0.0 milestone-15, whose image digests the deposit lists. A single entry point reproduces every data table of this article. Monte Carlo results use the fixed seed stated in Section 5.3, save the sweep of Section 5.10, and reproduce on any machine. The exceptions are the serialisation slope of Equation (8), which enters the export cost, and through it the break-even ratios of Table 9, and wherever else they are quoted, the compression rates behind Tables 7 and 13, and the measurements of the end-to-end run, all of which are properties of the machine they were measured on. No new data were created beyond those artefacts.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AAS-QL	Asset Administration Shell Query Language	AAS	Asset Administration Shell
API	Application Programming Interface	CDD	Common Data Dictionary
ECLASS	Electronic Classification Standard	HTTP	Hypertext Transfer Protocol
GQL	Graph Query Language, ISO/IEC 39075:2024	JSON	JavaScript Object Notation
IDTA	Industrial Digital Twin Association	LLM	Large Language Model
IRDI	International Registration Data Identifier	LPG	Labeled Property Graph
IRI	Internationalized Resource Identifier	OBDA	Ontology-Based Data Access
PLC	Programmable Logic Controller	OPC UA	OPC Unified Architecture
RAMI 4.0	Reference Architecture Model Industrie 4.0	OWL	Web Ontology Language
RDF	Resource Description Framework	RTT	Round-Trip Time
REST	Representational State Transfer	SMC	SubmodelElementCollection
SPARQL	SPARQL Protocol and RDF Query Language	SQL	Structured Query Language
WSL	Windows Subsystem for Linux	URI	Uniform Resource Identifier

References

1. Jazdi, N. Cyber Physical Systems in the Context of Industry 4.0. In Proceedings of the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 22–24 May 2014; pp. 1–4. [CrossRef]
2. Grieves, M. Digital Twin: Manufacturing Excellence through Virtual Factory Replication. White Paper, Florida Institute of Technology, 2014. Available online: https://theengineer.markallengroup.com/production/content/uploads/2014/12/Digital_Twin_White_Paper_Dr_Grieves.pdf (accessed on 17 August 2026).
3. Tao, F.; Zhang, H.; Liu, A.; Nee, A.Y.C. Digital Twin in Industry: State-of-the-Art. *IEEE Trans. Ind. Inform.* **2019**, *15*, 2405–2415. [CrossRef]
4. Kritzinger, W.; Karner, M.; Traar, G.; Henjes, J.; Sihn, W. Digital Twin in Manufacturing: A Categorical Literature Review and Classification. *IFAC-PapersOnLine* **2018**, *51*, 1016–1022. [CrossRef]
5. Zheng, X.; Lu, J.; Kiritsis, D. The Emergence of Cognitive Digital Twin: Vision, Challenges and Opportunities. *Int. J. Prod. Res.* **2022**, *60*, 7610–7632. [CrossRef]
6. Industrial Digital Twin Association. *Specification of the Asset Administration Shell, Part 1: Metamodel*; Technical Report IDTA 01001-3-2; Version 3.2; IDTA: Frankfurt am Main, Germany, 2026. Available online: https://industrialdigitaltwin.io/aas-specifications/IDTA-01001/v3.2/index.html (accessed on 31 July 2026).
7. *DIN SPEC 91345:2016-04*; Reference Architecture Model Industrie 4.0 (RAMI4.0). Technical Report; Deutsches Institut für Normung: Normung, Germany, 2016. [CrossRef]
8. Industrial Digital Twin Association. *Specification of the Asset Administration Shell, Part 2: Application Programming Interfaces*; Technical Report IDTA 01002-3-2; Version 3.2; IDTA: Frankfurt am Main, Germany, 2026. Available online: https://industrialdigitaltwin.io/aas-specifications/IDTA-01002/v3.2/index.html (accessed on 31 July 2026).
9. Niemz, P.; Teischinger, A.; Sandberg, D. (Eds.) *Springer Handbook of Wood Science and Technology*; Springer International Publishing: Cham, Switzerland, 2023. [CrossRef]
10. *ISO/IEC 39075:2024*; Information Technology—Database Languages—GQL. Technical Report; International Organization for Standardization: Geneva, Switzerland, 2024. Available online: https://www.iso.org/standard/76120.html (accessed on 31 July 2026).
11. Steinmetz, C.; Glöwing, K.; Schroeder, G.N.; Binotto, A.; Rettberg, A.; Pereira, C.E. Semantische digitale Zwillinge. Eine Methodik zur Digital-Twin-Erstellung mit Hilfe von Wissensgraphen. *Atp. Mag.* **2023**, *65*, 69–77. (In German) [CrossRef]
12. Braun, D.; Müller, T.; Sahlab, N.; Jazdi, N.; Schlögl, W.; Weyrich, M. A Graph-Based Knowledge Representation and Pattern Mining Supporting the Digital Twin Creation of Existing Manufacturing Systems. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany, 6–9 September 2022; pp. 1–4. [CrossRef]
13. Suwinski, P.; Braune, A. Wissensbasierter Datenraum auf Basis von Metamodellen für Digitale Zwillinge Elektronischer Kombinationstextilien. In Proceedings of the 26th Leitkongress der Mess- und Automatisierungstechnik—Automation 2025: Human-Centric Automation, Düsseldorf, Germany, 1–2 July 2025; VDI-Berichte; Volume 2457, pp. 27–40. (In German) [CrossRef]
14. European Parliament; Council of the European Union. Regulation (EU) 2024/1781 of the European Parliament and of the Council Establishing a Framework for the Setting of Ecodesign Requirements for Sustainable Products. Official Journal of the European Union, L Series, 2024. Available online: http://data.europa.eu/eli/reg/2024/1781/oj (accessed on 31 July 2026).

15. European Parliament and Council of the European Union. Regulation (EU) 2023/1542 of the European Parliament and of the Council Concerning Batteries and Waste Batteries. Official Journal of the European Union, L 191, 2023. Available online: http://data.europa.eu/eli/reg/2023/1542/oj (accessed on 31 July 2026).
16. *ISO/IEC 15459-6:2014*; Information Technology—Automatic Identification and Data Capture Techniques—Unique Identification—Part 6: Groupings. International Organization for Standardization: Geneva, Switzerland, 2014. Available online: https://www.iso.org/standard/54786.html (accessed on 25 October 2026).
17. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Digital Product Passport, Part 1 Metadata*; Technical Report IDTA 02099-1; Version 1.0; IDTA: Frankfurt am Main, Germany, 2026. Available online: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2026/05/IDTA-02099-1_Submodel-Digital-Product-Passport-Part-1.pdf (accessed on 17 August 2026).
18. Industrial Digital Twin Association; Catena-X; BatteryPass Consortium. *Digital Battery Passport: Use Case Guideline of the Asset Administration Shell*; Technical Report; Version 1.0; Industrial Digital Twin Association (IDTA): Frankfurt am Main, Germany, 2026. Available online: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2026/02/IDTA_Catena-X_Guideline_Digital_Battery_Passport.pdf (accessed on 17 August 2026).
19. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Carbon Footprint*; Technical Report IDTA 02023; Version 1.0; IDTA: Frankfurt am Main, Germany, 2025. Available online: https://industrialdigitaltwin.org/wp-content/uploads/2025/03/IDTA-02023_Submodel_CarbonFootprint.pdf (accessed on 17 August 2026).
20. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Predictive Maintenance*; Technical Report IDTA 02048; Version 1.0; IDTA: Frankfurt am Main, Germany, 2025. Available online: https://industrialdigitaltwin.org/wp-content/uploads/2025/06/IDTA-02048_Submodel_PredictiveMaintenance.pdf (accessed on 17 August 2026).
21. Löcklin, A.; Vietz, H.; White, D.; Ruppert, T.; Jazdi, N.; Weyrich, M. Data Administration Shell for Data-Science-Driven Development. *Procedia CIRP* **2021**, *100*, 115–120. [CrossRef]
22. Rauber, A.; Gösswein, B.; Zwölf, C.M.; Schubert, C.; Wörister, F.; Duncan, J.; Flicker, K.; Zettsu, K.; Meixner, K.; McIntosh, L.D.; et al. Precisely and Persistently Identifying and Citing Arbitrary Subsets of Dynamic Data. *Harv. Data Sci. Rev.* **2021**, *3*, 3–55. [CrossRef]
23. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Time Series Data*; Technical Report IDTA 02008-1-1; Version 1.1; IDTA: Frankfurt am Main, Germany, 2023. Available online: https://industrialdigitaltwin.org/wp-content/uploads/2023/03/IDTA-02008-1-1_Submodel_TimeSeriesData.pdf (accessed on 17 August 2026).
24. Huebner, J.E.; Ihlenfeldt, S. Semantic Interoperability in Industry 4.0: A Systematic Mapping Study on Integrating Knowledge Graphs with AAS and OPC UA. *J. Mach. Eng.* **2026**, *26*, 87–103. [CrossRef]
25. Grangel-González, I.; Halilaj, L.; Coşkun, G.; Auer, S.; Collarana, D.; Hoffmeister, M. Towards a Semantic Administrative Shell for Industry 4.0 Components. In Proceedings of the 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), Laguna Hills, CA, USA, 3–6 February 2016; pp. 230–237. [CrossRef]
26. Bader, S.R.; Maleshkova, M. The Semantic Asset Administration Shell. In *Semantic Systems. The Power of AI and Knowledge Graphs (SEMANTiCS 2019)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11702, pp. 159–174. [CrossRef]
27. Bader, S.R.; Grangel-González, I.; Nanjappa, P.; Vidal, M.E.; Maleshkova, M. A Knowledge Graph for Industry 4.0. In *Proceedings of the 17th Extended Semantic Web Conference (ESWC 2020), Online, 31 May–4 June 2020*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12123, pp. 465–480. [CrossRef]
28. Huang, Y.; Dhouib, S.; Palacios Medinacelli, L.; Malenfant, J. Enabling Semantic Interoperability of Asset Administration Shells through an Ontology-Based Modeling Method. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, Montreal, QC, Canada, 23–28 October 2022*; ACM: New York, NY, USA, 2022; pp. 497–502. [CrossRef]
29. Rimaz, M.H.; Plociennik, C.; Ruskowski, M. Semantic Asset Administration Shell for Circular Economy. In Proceedings of the Knowledge Graphs for Sustainability Workshop (KG4S) at ESWC, Crete, Greece, 27 May 2024; CEUR Workshop Proceedings; Volume 3753. Available online: https://ceur-ws.org/Vol-3753/paper1.pdf (accessed on 17 August 2026).
30. Hurtado, E.; Burgos, A.; Armentia, A.; Casquero, O. Self-Configurable Manufacturing Industrial Agents (SMIA): A Standardized Approach for Digitizing Manufacturing Assets. *J. Ind. Inf. Integr.* **2025**, *47*, 100915. [CrossRef]
31. Metović, A.; Maisch, N.; Ajdinović, S.; Lechler, A.; Wortmann, A.; Riedel, O. Industrial Semantics-Aware Digital Twins: A Hybrid Graph Matching Approach for Asset Administration Shells. In Proceedings of the 2026 IEEE 23rd Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2026. [CrossRef]
32. Schaper, S.; Listl, F.G.; Schenk, T.; Grimm, S.; Ast, A.; Verl, A. Automatic Behavior Simulation Model Generation for Virtual Commissioning of Machine Tools through Engineering Data Integration. In Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), Porto, Portugal, 9–12 September 2025. [CrossRef]

33. Beden, S.; Cao, Q.; Beckmann, A. Semantic Asset Administration Shells in Industry 4.0: A Survey. In Proceedings of the 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), Victoria, BC, Canada, 10–12 May 2021; pp. 31–38. [CrossRef]
34. Karabulut, E.; Pileggi, S.F.; Groth, P.; Degeler, V. Ontologies in Digital Twins: A Systematic Literature Review. *Future Gener. Comput. Syst.* **2024**, *153*, 442–456. [CrossRef]
35. Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; Rosati, R. Linking Data to Ontologies. *J. Data Semant. X* **2008**, *4900*, 133–173. [CrossRef]
36. Calvanese, D.; Cogrel, B.; Komla-Ebri, S.; Kontchakov, R.; Lanti, D.; Rezk, M.; Rodriguez-Muro, M.; Xiao, G. Ontop: Answering SPARQL Queries over Relational Databases. *Semant. Web* **2017**, *8*, 471–487. [CrossRef]
37. Dimou, A.; Vander Sande, M.; Colpaert, P.; Verborgh, R.; Mannens, E.; Van de Walle, R. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the 7th Workshop on Linked Data on the Web (LDOW 2014), Seoul, Republic of Korea, 8 April 2014; CEUR Workshop Proceedings; Volume 1184. Available online: https://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf (accessed on 31 July 2026).
38. Angles, R.; Thakkar, H.; Tomaszuk, D. Mapping RDF Databases to Property Graph Databases. *IEEE Access* **2020**, *8*, 86091–86110. [CrossRef]
39. Eclipse Tractus-X. Tractus-X Knowledge Agents AAS Bridges (KA-AAS): SPARQL-to-AAS Bridge, 2026. Implements an AAS Server and Registry Interface Backed by SPARQL Agents. Available online: https://github.com/eclipse-tractusx/knowledge-agents-aas-bridge (accessed on 31 July 2026).
40. Fraunhofer Institute for Mechanics of Materials IWM. KG2AAS-Prototype: Transforming RDF-Based Knowledge Graphs into Asset Administration Shells. SPARQL Construct with JSON-LD Framing Against the AAS JSON Schema. 2026. Available online: https://github.com/MI-FraunhoferIWM/KG2AAS-prototype (accessed on 31 July 2026).
41. Rongen, S.; Nikolova, N.; van der Pas, M. Modelling with AAS and RDF in Industry 4.0. *Comput. Ind.* **2023**, *148*, 103910. [CrossRef]
42. Luxenburger, A.; Porta, D.; Knoch, S.; Mohr, J.; Schwartz, T. A Service Infrastructure for Industrie 4.0 Testbeds Based on Asset Administration Shells. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–8. [CrossRef]
43. Sonnenberg, G.; Stein, P.; Espinosa, F.; Porta, D. Queryable AAS Graphs for AI Agents: An Event-Driven Knowledge Graph Integration for AAS Environments. In Proceedings of the 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), Porto, Portugal, 9–12 September 2025. [CrossRef]
44. German Research Center for Artificial Intelligence (DFKI). AAS Neo4j Integration, 2025. The Service Accompanying the Work of Sonnenberg et al. Available online: https://github.com/dfkibasys/aas_neo4j_integration (accessed on 31 July 2026).
45. Rübel, P.; Moarefvand, N.; Motsch, W.; Wagner, A.; Ruskowski, M. Enabling Fault Diagnosis in Skill-Based Production Environments. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–8. [CrossRef]
46. Rübel, P.; Motsch, W.; Bernhard, A.; Jungbluth, S.; Ruskowski, M. Agent-Based Communication for Fault Diagnosis in Skill-Based Production Environments Using Messages Based on I4.0 Language and Asset Administration Shells. In *Advances in Artificial Intelligence in Manufacturing II*; Springer: Berlin/Heidelberg, Germany, 2025; pp. 157–170. [CrossRef]
47. Sapel, P.; Garoufali, A.; Hopmann, C. Leveraging Ontologies and Asset Administration Shells for Decision-Support: A Case Study on Production Planning within the Injection Molding Domain. *Internet Things* **2025**, *34*, 101739. [CrossRef]
48. RWTH Aachen University, Chair of Information and Automation Systems (IAT). rwth-iat/neo4aas: A Proof-of-Concept Library for Mapping Asset Administration Shell Data to a Neo4j Graph Database, 2026. The Project Calls Itself aas4graph in Its Documentation. Available online: https://github.com/rwth-iat/neo4aas (accessed on 31 July 2026).
49. SAP SE. aas-transformation-library: Transforming Content Following Open Standards Such as AutomationML into Asset Administration Shell Format, 2023. Archived Read-Only. Available online: https://github.com/SAP-archive/aas-transformation-library (accessed on 17 August 2026).
50. Platenius-Mohr, M.; Malakuti, S.; Grüner, S.; Schmitt, J.; Goldschmidt, T. File- and API-Based Interoperability of Digital Twins by Model Transformation: An IIoT Case Study Using Asset Administration Shell. *Future Gener. Comput. Syst.* **2020**, *113*, 94–105. [CrossRef]
51. Lüder, A.; Behnert, A.K.; Rinker, F.; Biffl, S. Generating Industry 4.0 Asset Administration Shells with Data from Engineering Data Logistics. In Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; pp. 867–874. [CrossRef]
52. Miny, T.; Thies, M.; Epple, U.; Diedrich, C. Model Transformation for Asset Administration Shells. In Proceedings of the IECON 2020: The 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 18–21 October 2020; pp. 2207–2212. [CrossRef]

53. Eichelberger, H.; Weber, A. Model-Driven Realization of IDTA Submodel Specifications: The Good, the Bad, the Incompatible? In Proceedings of the 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 10–13 September 2024. [CrossRef]
54. Xia, Y.; Xiao, Z.; Jazdi, N.; Weyrich, M. Generation of Asset Administration Shell with Large Language Model Agents: Toward Semantic Interoperability in Digital Twins in the Context of Industry 4.0. *IEEE Access* **2024**, *12*, 84863–84877. [CrossRef]
55. Schiekofer, R.; Grimm, S.; Milicic Brandt, M.; Weyrich, M. A Formal Mapping Between OPC UA and the Semantic Web. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; pp. 33–40. [CrossRef]
56. Li, X.; Zhang, S.; Jiang, P.; Deng, M.; Wang, X.V.; Yin, C. Knowledge Graph Based OPC UA Information Model Automatic Construction Method for Heterogeneous Devices Integration. *Robot. Comput.-Integr. Manuf.* **2024**, *88*, 102736. [CrossRef]
57. Ji, F.; Wang, Z.; Vogel-Heuser, B. Leveraging Weighted Knowledge Graphs for Modeling and Analyzing Information Dependencies in Automated Production Systems. In Proceedings of the 2025 IEEE 8th International Conference on Industrial Cyber-Physical Systems (ICPS), Emden, Germany, 12–15 May 2025; pp. 1–8. [CrossRef]
58. van Leeuwen, C.; Bouter, C.; Hindriks, R.; Wilterdink, R. Representing the Virtual: Using AAS to Expose Digital Assets. In Proceedings of the 3rd International Workshop on Semantic Digital Twins (SeDiT 2022) Co-Located with the 19th Extended Semantic Web Conference (ESWC 2022), Hersonissos, Greece, 29 May–2 June 2022; CEUR Workshop Proceedings; Volume 3291, pp. 1–10. Available online: https://ceur-ws.org/Vol-3291/paper5.pdf (accessed on 31 July 2026).
59. Cavalieri, S.; Salafia, M.G. A Model for Predictive Maintenance Based on Asset Administration Shell. *Sensors* **2020**, *20*, 6028. [CrossRef] [PubMed]
60. de Oliveira, V.F.; Pessoa, M.A.d.O.; Junqueira, F.; Miyagi, P.E. SQL and NoSQL Databases in the Context of Industry 4.0. *Machines* **2022**, *10*, 20. [CrossRef]
61. Angles, R. The Property Graph Database Model. In Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW), Cali, Colombia, 21–25 May 2018; Volume 2100. Available online: https://ceur-ws.org/Vol-2100/paper26.pdf (accessed on 31 July 2026).
62. Angles, R.; Gutierrez, C. Survey of Graph Database Models. *ACM Comput. Surv.* **2008**, *40*, 1–39. [CrossRef]
63. Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; de Melo, G.; Gutierrez, C.; Kirrane, S.; Labra Gayo, J.E.; Navigli, R.; Neumaier, S.; et al. Knowledge Graphs. *ACM Comput. Surv.* **2022**, *54*, 1–37. [CrossRef]
64. Francis, N.; Green, A.; Guagliardo, P.; Libkin, L.; Lindaaker, T.; Marsault, V.; Plantikow, S.; Rydberg, M.; Selmer, P.; Taylor, A. Cypher: An Evolving Query Language for Property Graphs. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD), Houston, TX, USA, 10–15 June 2018; pp. 1433–1445. [CrossRef]
65. Deutsch, A.; Francis, N.; Green, A.; Hare, K.; Li, B.; Libkin, L.; Lindaaker, T.; Marsault, V.; Martens, W.; Michels, J.; et al. Graph Pattern Matching in GQL and SQL/PGQ. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD), Philadelphia, PA, USA, 12–17 June 2022; pp. 2246–2258. [CrossRef]
66. ECLASS e.V. *ECLASS Standard: Standardised Material and Service Classification*; Technical Report; ECLASS e.V.: Cologne, Germany, 2026. Available online: https://eclass.eu (accessed on 18 August 2026).
67. Gupta, A.; Mumick, I.S. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.* **1995**, *18*, 3–18.
68. Halevy, A.Y. Answering Queries Using Views: A Survey. *VLDB J.* **2001**, *10*, 270–294. [CrossRef]
69. Kleppmann, M. *Designing Data-Intensive Applications*; O'Reilly Media: Sebastopol, CA, USA, 2017.
70. Neo4j, Inc. Cypher Manual 5: Scalar Functions, ElementId(), 2026. Available online: https://neo4j.com/docs/cypher-manual/5/functions/scalar/ (accessed on 18 October 2026).
71. Stevens, P. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. *Softw. Syst. Model.* **2010**, *9*, 7–20. [CrossRef]
72. Czarnecki, K.; Foster, J.N.; Hu, Z.; Lämmel, R.; Schürr, A.; Terwilliger, J.F. Bidirectional Transformations: A Cross-Discipline Perspective. In *Proceedings of the Theory and Practice of Model Transformations (ICMT 2009)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5563, pp. 260–283. [CrossRef]
73. Hidaka, S.; Tisi, M.; Cabot, J.; Hu, Z. Feature-Based Classification of Bidirectional Transformation Approaches. *Softw. Syst. Model.* **2016**, *15*, 907–928. [CrossRef]
74. Katti, B.; Plociennik, C.; Ruskowski, M.; Schweitzer, M. Bidirectional Transformation of MES Source Code and Ontologies. *Procedia Manuf.* **2020**, *42*, 197–204. [CrossRef]
75. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Hierarchical Structures Enabling Bills of Material*; Technical Report IDTA 02011-1-1; IDTA: Frankfurt am Main, Germany, 2024. Available online: https://industrialdigitaltwin.org/wp-content/uploads/2024/06/IDTA-02011-1-1_Submodel_HierarchicalStructuresEnablingBoM.pdf (accessed on 31 July 2026).
76. Groth, P.; Gibson, A.; Velterop, J. The Anatomy of a Nanopublication. *Inf. Serv. Use* **2010**, *30*, 51–56. [CrossRef]

77. Lebo, T.; Sahoo, S.; McGuinness, D. PROV-O: The PROV Ontology. W3C Recommendation, World Wide Web Consortium. 2013. Available online: https://www.w3.org/TR/prov-o/ (accessed on 31 July 2026).
78. *ISO/TS 29002-5:2009*; Industrial Automation Systems and Integration—Exchange of Characteristic Data—Part 5: Identification Scheme. Technical Report; ISO (International Organization for Standardization): Geneva, Switzerland, 2009.
79. Duerst, M.; Suignard, M. *Internationalized Resource Identifiers (IRIs)*; Technical Report RFC 3987; Internet Engineering Task Force: Wilmington, DE, USA, 2005. [CrossRef]
80. Industrial Digital Twin Association. *Submodel Template of the Asset Administration Shell: Generic Frame for Technical Data for Industrial Equipment in Manufacturing*; Technical Report IDTA 02003; Version 2.0; IDTA: Frankfurt am Main, Germany, March 2025. Available online: https://industrialdigitaltwin.org/wp-content/uploads/2025/03/IDTA-02003_Generic-Frame-for-Technical-Data.pdf (accessed on 17 August 2026).
81. Neo4j, Inc. Neo4j Graph Database Platform. 2026. Available online: https://neo4j.com/ (accessed on 31 July 2026).
82. Robinson, I.; Webber, J.; Eifrem, E. *Graph Databases: New Opportunities for Connected Data*, 2nd ed.; O'Reilly Media: Sebastopol, CA, USA, 2015.
83. Eclipse Foundation. Eclipse BaSyx: Open Source Platform for Industrie 4.0. 2026. Available online: https://wiki.basyx.org/ (accessed on 31 July 2026).
84. Aas-Core-Works Contributors. aas-core-works: Tools and Libraries to Work with Asset Administration Shells. 2026. Available online: https://github.com/aas-core-works (accessed on 31 July 2026).
85. Harinarayan, V.; Rajaraman, A.; Ullman, J.D. Implementing Data Cubes Efficiently. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, QC, Canada, 4–6 June 1996; pp. 205–216. [CrossRef]
86. Gupta, H.; Mumick, I.S. Selection of Views to Materialize in a Data Warehouse. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 24–43. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.