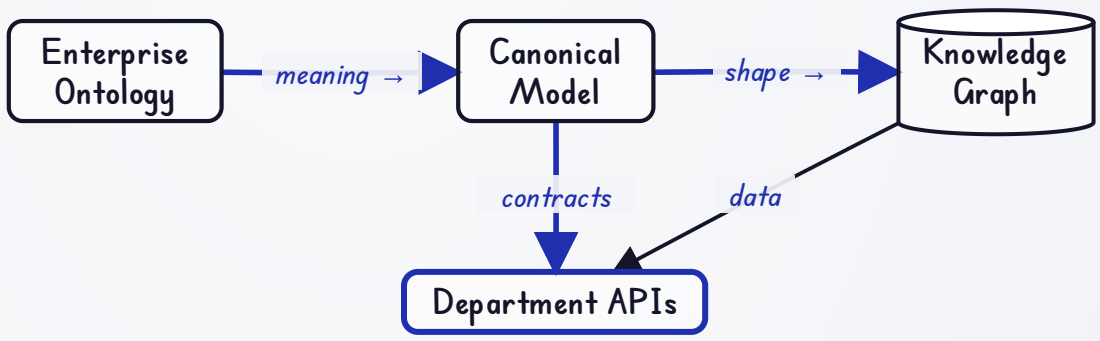
Enterprise Ontology, Knowledge Graph & Canonical Model

Complete Practitioner Study Notes

Foundations • Kinds of Models • Canonical Model • Industry Implementation • Knowledge Graphs • Department APIs • Accuracy & Consistency • Governance & Coordination



Every page sourced: W3C • ISO • OMG / EDM Council • peer-reviewed literature

Prepared by: Rajesh Kumar Gupta

104 Pages

Edition: November 2026

Rajesh Kumar Gupta

Read This First

0.1 Who This Notebook Is For

- Enterprise & data architects
- Ontologists & knowledge engineers
- Data stewards & domain owners
- API, integration & platform teams
- AI / GraphRAG engineers

0.2 How to Read It

1. **Parts A-B** : vocabulary & model types
2. **Part C** : the canonical model
3. **Part D** : build the ontology at work
4. **Part E** : the knowledge graph
5. **Part F** : APIs & departments
6. **Parts G-H** : prove it right, govern it

0.3 Source Policy

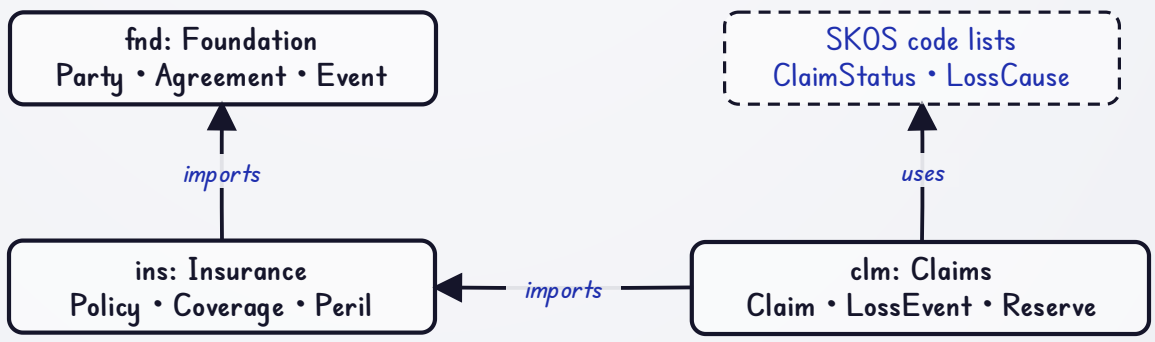
Every page ends with a **Sources:** line. Facts come from:

- W3C Recommendations & Notes (RDF, OWL 2, SKOS, SHACL, SPARQL, JSON-LD, PROV-O, DCAT, R2RML)
- ISO / IEC standards (21838 BFO, 39075 GQL, 25012, 11179)
- OMG & EDM Council (FIBO), ACORD, HL7, ISO 20022
- Peer-reviewed papers & standard textbooks (listed on pp. 102-104)

Note: Numbers marked *illustrative* are teaching examples, not measured results. Spec status changes — check w3.org / iso.org for the latest version.

0.4 The Running Example (fictional insurer)

All examples use one invented enterprise with three modules under **https://ontology.example.com/**. It is not any real company's model.



Key Takeaway: Ontology gives meaning, the canonical model turns meaning into contracts, the knowledge graph holds the connected data — and governance keeps all three aligned.

Sources: See pp. 102-104 for the full reference list

Rajesh Kumar Gupta

Contents (1/2)

Bold number = topic (circled on each page). Topic N is on page N + 4.

A Foundations of Ontology pp. 5-16

1 Data, Information, Knowledge, Meaning • **2** What is an Ontology? • **3** Why Enterprises Need Ontology • **4** The Semantic Spectrum • **5** Building Blocks • **6** Open World vs Closed World • **7** RDF: Triples, IRIs, Graphs • **8** RDFS & OWL 2 Essentials • **9** SKOS for Code Lists • **10** SHACL: Shapes as Data Contracts • **11** SPARQL Essentials • **12** Part A Recap + Checklist

B Kinds of Models pp. 17-28

13 Conceptual, Logical, Physical • **14** Entity-Relationship Modeling • **15** UML Class Diagrams vs OWL • **16** Dimensional Models • **17** Data Vault 2.0 • **18** Document Models: JSON & Avro • **19** Labeled Property Graphs & GQL • **20** RDF Graph vs Property Graph • **21** Upper / Foundational Ontologies • **22** Upper, Core, Domain, Task, Application • **23** Ontology Design Patterns • **24** Model Comparison Matrix

C The Canonical Model pp. 29-40

25 What is a Canonical Model? • **26** Point-to-Point vs Canonical • **27** Canonical vs Ontology • **28** Canonical Message Design • **29** Ontology → Canonical → Schemas • **30** Industry Canonical Standards • **31** Identity, Keys & MDM • **32** Reference Data & Code Lists • **33** Making the Ontology Succeed • **34** Canonical Anti-Patterns • **35** Bounded Contexts vs Canonical • **36** Part C Recap & Checklist

D Implementing Enterprise Ontology in Industry pp. 41-52

37 Ontology Methodologies • **38** Competency Questions • **39** Scoping & Modular Architecture • **40** Reuse & Alignment • **41** Naming, IRI & Namespaces • **42** Annotations & Definitions • **43** Tooling Landscape • **44** Ontology-as-Code CI/CD • **45** Mapping Source Data • **46** Worked Example I: Claims Classes • **47** Worked Example II: Axioms & Shapes • **48** Part D Recap & Roadmap

Rajesh Kumar Gupta

Contents (2/2)

Bold number = topic (circled on each page). Topic N is on page N + 4.

E Knowledge Graphs pp. 53-64

49 What is a KG? • **50** The KG in Enterprise Architecture • **51** Ontology + Canonical + KG • **52** KG Construction Pipeline • **53** Entity Resolution • **54** Storage, Named Graphs & Provenance • **55** Reasoning & Inference • **56** Querying, Federation, Virtual KGs • **57** KG + LLMs: GraphRAG • **58** Catalogs & Data Products • **59** Scale & Operations • **60** Part E Recap

F Consuming the Ontology: APIs & Departments pp. 65-76

61 The Semantic Layer Concept • **62** JSON-LD: Ontology Terms in API JSON • **63** Generating OpenAPI & JSON Schema • **64** GraphQL over the Knowledge Graph • **65** Linked-Data REST & Content Negotiation • **66** Event-Driven Consumption • **67** How Each Department Consumes the Ontology • **68** Reuse: Enterprise → Domain → API • **69** Versioning & Backward Compatibility • **70** Contracts & API Governance • **71** Data Mesh & Federated Governance • **72** Security & Access

G Accuracy, Consistency & Quality pp. 77-88

73 Quality Dimensions • **74** Logical Consistency with Reasoners • **75** Explaining Inconsistency • **76** OntoClean • **77** Common Modeling Pitfalls • **78** SHACL Validation at Scale • **79** Competency Questions as Unit Tests • **80** Conformance & Drift Detection • **81** Expert Review & Business Validation • **82** Ontology Metrics & Coverage • **83** Regression & Release Gates • **84** Part G Recap & Quality Gates

H Governance, Coordination & Best Practices pp. 89-104

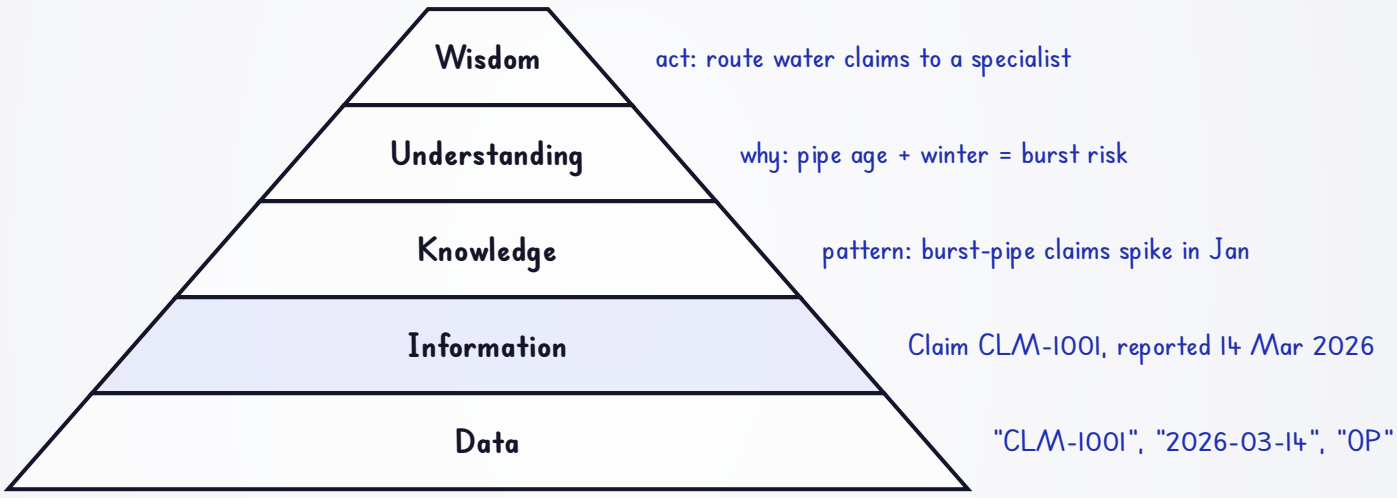
85 Operating Models • **86** Roles & Responsibilities • **87** RACI for Changes • **88** Change Request Lifecycle • **89** Design Authority & ADRs • **90** Glossary ↔ Ontology ↔ Canonical • **91** Onboarding Playbook • **92** Adoption & Enforcement • **93** KPIs & Measuring Value • **94** Why Programs Fail • **95** Best Practices: Canonical • **96** Best Practices: Ontology • **97** Master Glossary • **98** References I • **99** References II • **100** References III

① Data, Information, Knowledge, Meaning

Part A · Foundations of Ontology

1.1 The DIKW ladder

Ackoff (1989) described a hierarchy from **data** to **wisdom** (his original list also includes *understanding*). Each step adds context and purpose. Insurance example:

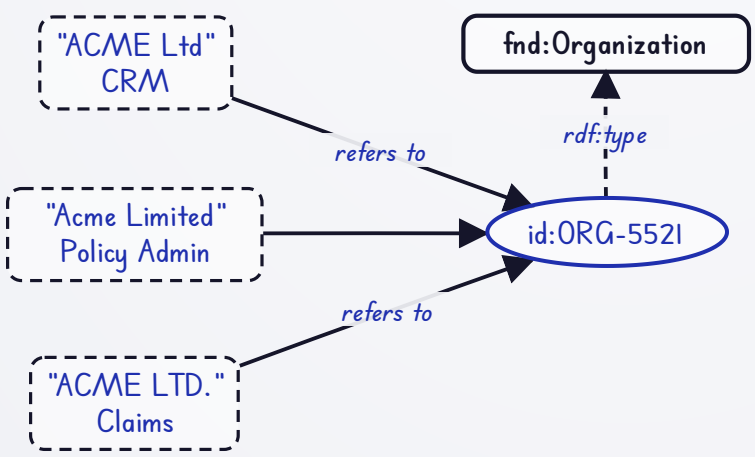


1.2 Meaning = data + agreed context

Data become **information** when we agree *what each value refers to* (a claim, a date, a status) and **knowledge** when we capture rules and relationships that hold across many records. Ontology is the discipline of writing that agreement down explicitly.

1.3 Strings vs Things

Singhal (Google, 2012) summed up the Knowledge Graph idea as **"things, not strings"**: stop matching text, start identifying the real-world entity behind it.



1.4 Where meaning hides today

- **Column names** : *CUST_TYP_CD* means nothing outside the team that built it.
- **Code values** : "OP" = Open in one system, "Open Peril" in another.
- **Tribal knowledge** : the rule lives in one steward's head or a spreadsheet.
- **Application code** : "a claim is late if..." buried in Java/SQL.

An ontology moves this meaning out of code and heads into a shared, machine-readable model that every system can look up.

1.5 Monday-morning actions

1. **Pick 10 business terms** : that cause the most arguments (Customer, Claim, Loss Date...)
2. **Trace each term** : through 3 systems: name, codes, rules; note the conflicts
3. **Give each real-world thing one identifier** : an IRI, not a free-text name

Key Takeaway: Data alone are strings; meaning appears only when context is explicit and shared. Ontology is how we make that context explicit.

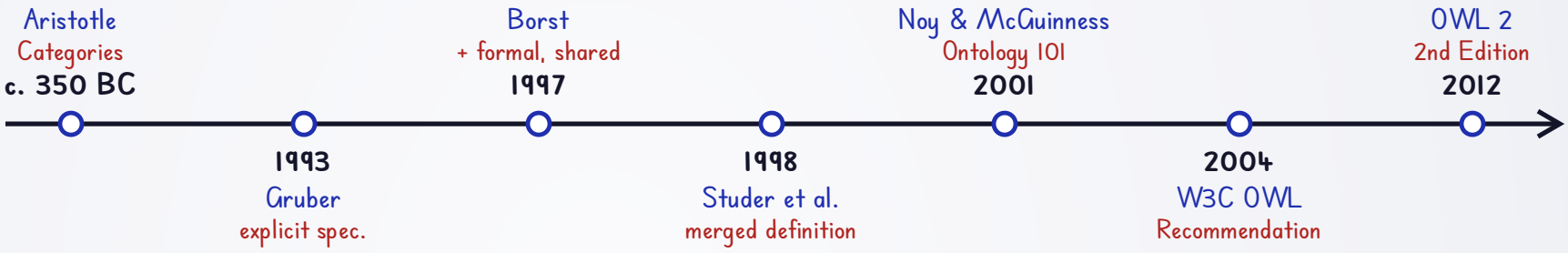
Sources: Ackoff, "From Data to Wisdom", J. Applied Systems Analysis 16 (1989) · Singhal, "Introducing the Knowledge Graph: things, not strings", Google blog (16 May 2012)

Rajesh Kumar Gupta

② What is an Ontology?

Part A · Foundations of Ontology

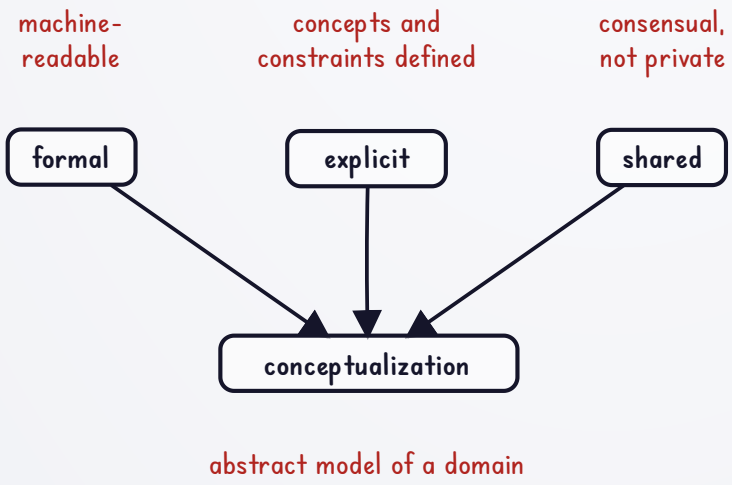
2.1 A 2,400-year lineage



2.2 The definitions practitioners quote

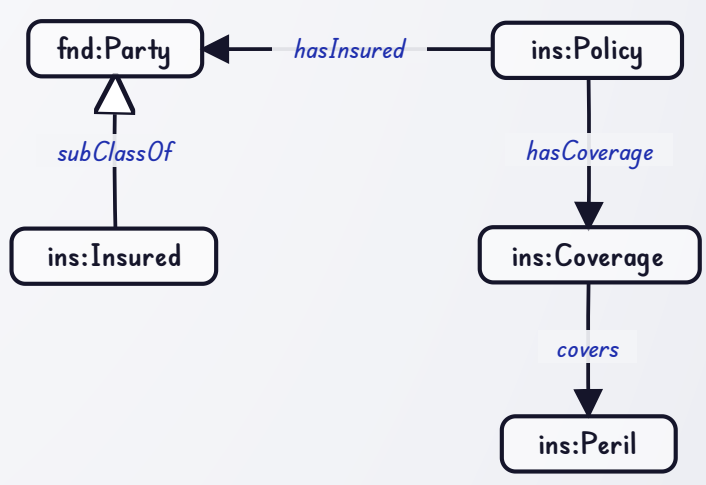
Who	Definition (verbatim or paraphrased)	What it adds
Gruber 1993	"an explicit specification of a conceptualization"	writing the model down
Borst 1997	paraphrase: a *formal* specification of a *shared* conceptualization	machine-readable + consensus
Studer, Benjamins & Fensel 1998	"a formal, explicit specification of a shared conceptualization"	the merged, most-cited form
Noy & McGuinness 2001	"a formal explicit description of concepts in a domain of discourse" (classes, slots, facets)	practical: ontology + instances = knowledge base

2.3 Unpacking the definition



Studer et al. explain each word; my one-line glosses are in red.

2.4 A tiny ontology



Classes + relations + definitions, **independent of any one database.**

2.5 What an ontology is NOT

- **Not a database schema** : a schema serves one store; an ontology states meaning for all of them.
- **Not just a taxonomy** : a tree of terms has no properties, constraints or logic.
- **Not the truth** : it is an agreed *model*, versioned and governed like any asset.

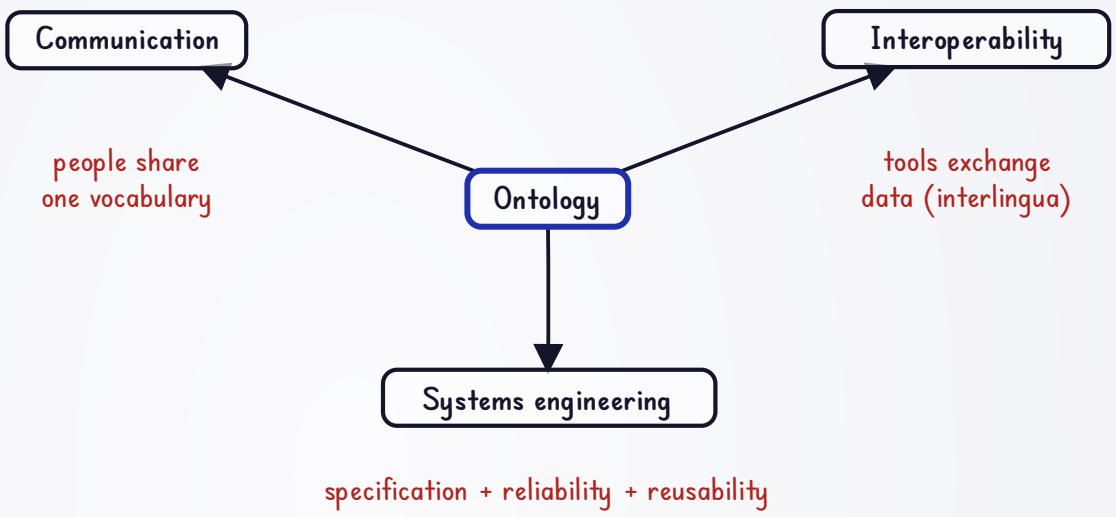
Key Takeaway: Use the Studer et al. wording in charters: **formal** (machines can process it), **explicit** (nothing implicit), **shared** (owned by the business, not one team).

Sources: Gruber, "A translation approach to portable ontology specifications", Knowledge Acquisition 5(2) (1993) · Borst, PhD thesis, Univ. of Twente (1997) · Studer, Benjamins & Fensel, "Knowledge engineering: principles and methods", Data & Knowledge Eng. 25 (1998) · Noy & McGuinness, "Ontology Development 101", Stanford KSL (2001)

③ Why Enterprises Need Ontology

Part A · Foundations of Ontology

3.1 Three classic uses (Uschold & Gruninger 1996)



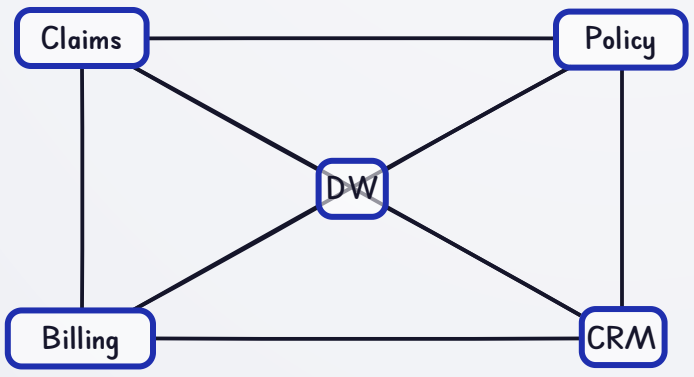
3.2 Five reasons to build one (Noy & McGuinness 2001)

1. Share a common understanding of the structure of information (people **and** software agents)
2. Enable reuse of domain knowledge
3. Make domain assumptions explicit
4. Separate domain knowledge from operational (application) knowledge
5. Analyze domain knowledge

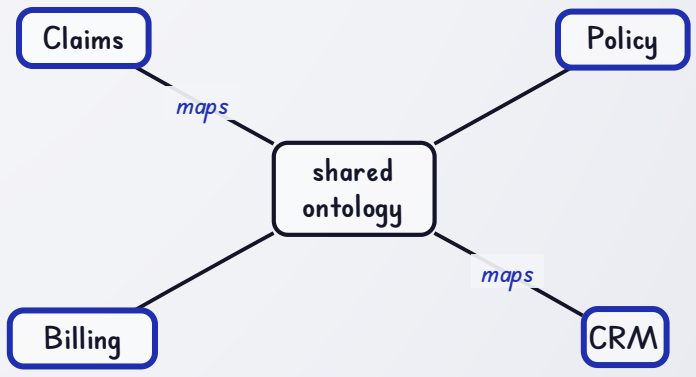
3.3 Enterprise purposes → artefacts

Purpose	Pain today	Ontology delivers
Shared meaning	"Customer" means 4 things	one definition + IRI per concept
Integration	mapping every pair of systems	map each system once to the model
Reasoning	rules copied into many apps	axioms a reasoner applies everywhere
Reuse	each project re-models Party	import fnd: instead of re-inventing
Governance	no owner for definitions	stewarded, versioned terms
AI grounding	LLM guesses field meaning	typed entities + definitions for retrieval

3.4 Without: pairwise mappings



3.5 With: one shared model



Decision rule: Do not start an ontology when only one application uses the terms. Start when **several systems or teams must agree** on meaning. That is where the value is.

Sources: Uschold & Gruninger, "Ontologies: principles, methods and applications", Knowledge Eng. Review 11(2) (1996) · Noy & McGuinness, "Ontology Development 101", Stanford KSL-01-05 (2001)

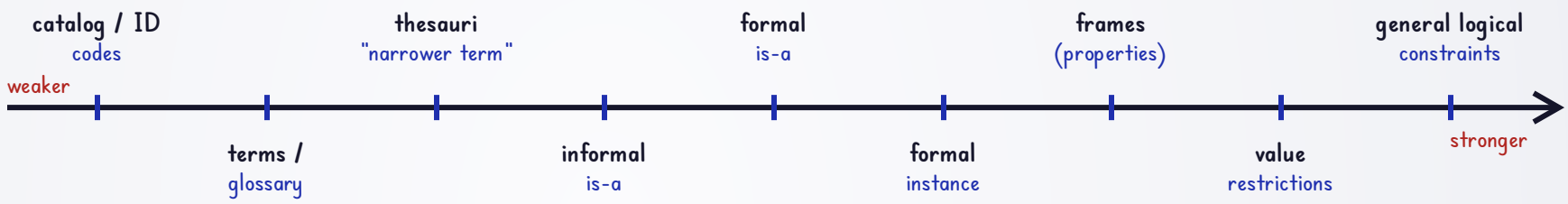
Rajesh Kumar Gupta

④ The Semantic Spectrum

Part A · Foundations of Ontology

4.1 Lassila & McGuinness spectrum (redrawn)

Lassila & McGuinness (2001) arranged "ontologies" from simple to expressive. They set the minimum bar at **strict (formal) is-a** subclass hierarchies; frames, properties and value restrictions are typical but not mandatory.



4.2 The practitioner ladder

Ontology (OWL 2)
classes + properties + axioms a reasoner can use

Schema / data model
entities, attributes, keys for one store

Thesaurus (SKOS)
broader / narrower / related + synonyms

Taxonomy
a single is-a or broader tree

Glossary
terms + natural-language definitions

Controlled list
allowed values, e.g. ClaimStatus codes

4.3 Decision rule

Climb only as high as your **competency questions** require:

- Need allowed values? → **list**.
- Need agreed wording? → **glossary**.
- Need search & navigation? → **taxonomy / thesaurus**.
- Need integration across stores? → **schema + mappings**.
- Need inference, consistency checks, rich queries? → **ontology**.

Most enterprises run **all of them at once**: SKOS code lists inside an OWL ontology, published as JSON Schema for APIs.

4.4 Same concept at each level (insurance)

Level	Example for "Water Damage"	Typical standard
List	peril code *WD*	CSV, code table
Glossary	"Damage caused by escape of water…"	business glossary
Taxonomy / thesaurus	WD broader *Property Peril*; altLabel "Escape of water"	SKOS
Schema	COVERAGE.PERIL_CD char(2) FK	SQL DDL, JSON Schema
Ontology	ins:WaterDamage subClassOf ins:Peril; disjoint with ins:Flood	RDFS / OWL 2

Allemang, Hendler & Gandon stress that on the Web *anyone can say anything about any topic* (their "AAA" slogan), so a model must tolerate partial, multi-source data. That is why the upper levels use open-world logic (topic 6).

Key Takeaway: An ontology is not "better" than a taxonomy. It is **more expressive and more expensive**. Choose the level by the questions you must answer, then plan to grow.

Sources: Lassila & McGuinness, "The Role of Frame-Based Representation on the Semantic Web", Stanford KSL-01-02 (2001) · Allemang, Hendler & Gandon, Semantic Web for the Working Ontologist, 3rd ed., ACM Books (2020) · W3C SKOS Reference (2009)

⑤ Building Blocks

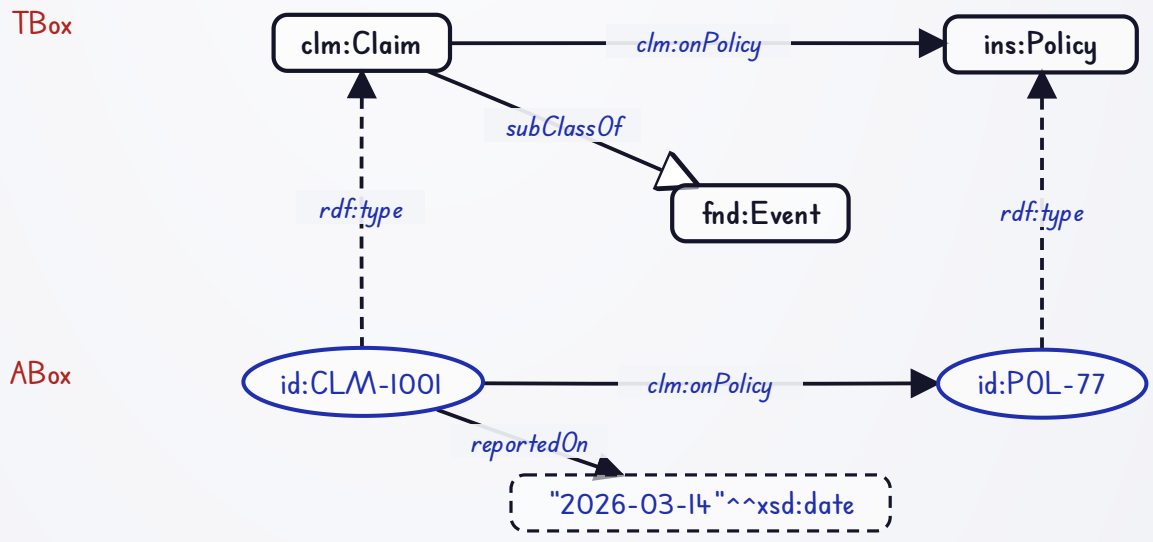
Part A · Foundations of Ontology

5.1 The OWL 2 entity kinds

Block	What it is	Claims example
Class	a set of things sharing meaning	clm:Claim, ins:Policy
Individual	one member of a class	id:CLM-1001
Object property	links individual → individual	clm:onPolicy
Data property	links individual → literal value	clm:reportedOn (xsd:date)
Annotation property	human / admin metadata; no logic	rdfs:label, skos:definition
Datatype	a value space	xsd:date, xsd:decimal
Axiom	a statement the model asserts	Claim subClassOf Event

5.2 TBox vs ABox in one picture

Description-logic terminology: the **TBox** holds terminology (classes, properties, axioms); the **ABox** holds assertions about individuals. In RDF both are just triples, so keep them in **separate files / named graphs** with separate owners.



5.3 The same in Turtle

```
# prefixes omitted (see 7.3)
# ---- TBox: the vocabulary -------------------
clm:Claim a owl:Class ;
    rdfs:label "Claim"@en ;
    skos:definition "A request for payment under a policy for
a loss."@en .
clm:onPolicy a owl:ObjectProperty ;
    rdfs:domain clm:Claim ; rdfs:range ins:Policy .
clm:reportedOn a owl:DatatypeProperty ;
    rdfs:domain clm:Claim ; rdfs:range xsd:date .
clm:steward a owl:AnnotationProperty .
# ---- ABox: facts about individuals ----------
id:CLM-1001 a clm:Claim ;
    clm:onPolicy id:POL-77 ;
    clm:reportedOn "2026-03-14"^^xsd:date .
```

5.4 Axiom types you will use

- **subClassOf / equivalentClass**
- **disjointWith** (Person vs Organization)
- **domain / range** (beware: they infer, see 6)
- **restrictions**: some / only / cardinality
- **property traits**: inverse, functional, transitive
- **hasKey** for business keys

Tip: Annotation properties carry definitions and stewardship but are ignored by reasoners. Use them freely for governance metadata.

Key Takeaway: Every concept needs: an IRI, a label, a definition, a parent, and an owner. Classes and properties go in the TBox; data go in the ABox.

Sources: W3C OWL 2 Structural Specification, 2nd ed. (2012) · Baader et al., The Description Logic Handbook, CUP (2003) · W3C RDF 1.1 Turtle (2014)

⑥ Open World vs Closed World

Part A · Foundations of Ontology

6.1 Two assumptions, two worlds

	Database / SHACL mindset	OWL mindset
Missing fact	**Closed world**: not stored ⇒ false	**Open world**: not stated ⇒ unknown
Names	**Unique names**: 2 keys = 2 rows	**No UNA**: 2 IRIs may be the same thing
Constraint breach	reject / error	infer new facts, or inconsistency
Good for	data quality, contracts, APIs	meaning, integration, classification

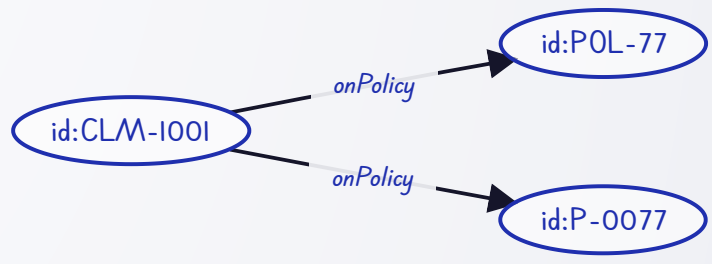
The OWL 2 Primer: in a database a missing fact is usually taken as false, while in OWL it "may simply be missing (but possibly true)". It also states that OWL does not assume different names denote different individuals.

6.2 Surprise 1: range infers

```
# prefixes omitted
clm:onPolicy rdfs:range ins:Policy .
id:CLM-1001 clm:onPolicy id:XYZ-9 .
# Reasoner INFERS:  id:XYZ-9 a ins:Policy .
# It does NOT report "XYZ-9 is not a Policy".
```

rdfs:range is **not a check**. It is a rule that adds a type.

6.3 Surprise 2: cardinality merges



onPolicy is functional, so
reasoner infers POL-77 sameAs P-0077

No error, unless you also assert *owl:differentFrom*. Then you get an inconsistency.

6.4 So who checks the data?

OWL 2 (meaning + inference) → **reasoner** (adds implied facts) → SHACL (closed-world checks) → **report** (Violation / Warning)

6.5 Practical rules

1. **Model meaning in OWL** : definitions, hierarchy, disjointness, inverse
2. **Enforce data rules in SHACL** : required fields, cardinality, allowed codes
3. **Assert distinctness where it matters** : owl:differentFrom / owl:AllDifferent for key individuals
4. **Treat owl:sameAs as a strong claim** : every fact of one IRI flows to the other; prefer a reviewed link

Key Takeaway: "Missing" in OWL is not "wrong". If your requirement says *must have*, write a SHACL shape, not an OWL restriction.

Sources: W3C OWL 2 Web Ontology Language Primer, 2nd ed. (2012) · W3C SHACL Recommendation (2017) · Allemang, Hendler & Gandon, Semantic Web for the Working Ontologist (2020)

Rajesh Kumar Gupta

⑦ RDF: Triples, IRIs, Graphs

Part A · Foundations of Ontology

7.1 One triple = one fact



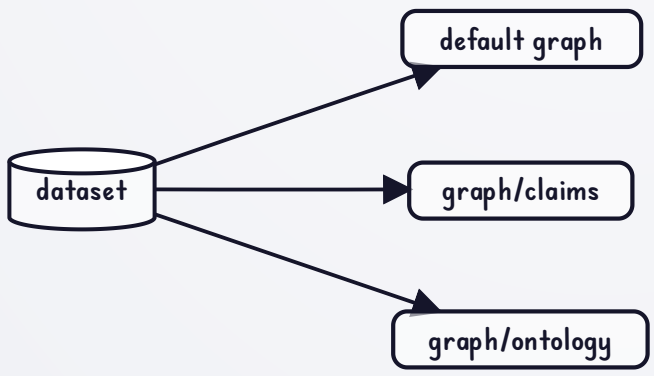
7.2 RDF terms

Term	Use	Example
IRI	global name of a thing or property	<https://ontology.example.com/clm/Claim>
Literal	value + datatype or language tag	"2026-03-14"^^xsd:date, "Open"@en
Blank node	local, unnamed node	[fnd:city "Trenton"]
Triple term (RDF 1.2)	a triple used as the object of another triple	statements about statements

7.3 Turtle: the format to learn first

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix fnd:  <https://ontology.example.com/fnd/> .
@prefix ins:  <https://ontology.example.com/ins/> .
@prefix clm:  <https://ontology.example.com/clm/> .
@prefix id:   <https://data.example.com/id/> .   # ABox
id:CLM-1001 a clm:Claim ;                  # 'a' = rdf:type
    clm:onPolicy   id:POL-77 ;           # ';' same subject
    clm:reportedOn "2026-03-14"^^xsd:date ;
    rdfs:label     "Burst pipe, kitchen"@en , "Rohrbruch, Küche"@de ;
    clm:lossAddress [ fnd:city "Trenton" ] .  # [ ] blank node
```

7.4 Graphs & datasets



A **dataset** = one default graph + named graphs. Put the ontology, each source and each load in its own graph for provenance and access control.

7.5 Serializations

- **Turtle / TriG** : human-readable (TriG adds graphs)
- **N-Triples / N-Quads** : one line per triple; bulk loads, diffs
- **JSON-LD** : JSON for APIs (Part F)
- **RDF/XML** : legacy, still common in tools

Status: RDF 1.1 (2014) is the current Recommendation. RDF 1.2 Concepts reached Candidate Recommendation (Apr 2026); RDF 1.2 Turtle was still a Working Draft in late 2026. Check current status at w3.org before adopting triple terms.

Key Takeaway: Same IRI = same thing, in every system. Mint IRIs only under a domain your organization controls (this book uses the reserved example domains: ontology.example.com for terms, data.example.com for data).

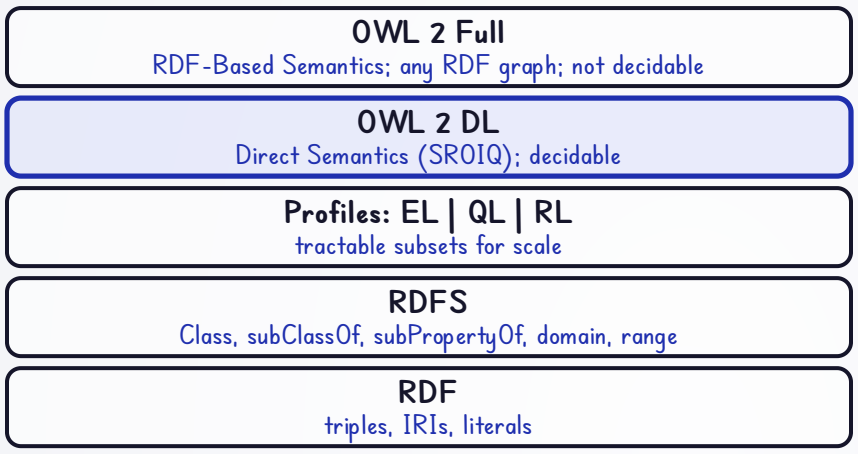
Sources: W3C RDF 1.1 Concepts and Abstract Syntax (2014) · W3C RDF 1.1 Turtle (2014) · W3C RDF 1.2 Concepts, Candidate Recommendation Snapshot (7 Apr 2026)

Rajesh Kumar Gupta

⑧ RDFS & OWL 2 Essentials

Part A · Foundations of Ontology

8.1 The stack



OWL 2 has **two semantics**: the **Direct Semantics** (description logic SROIQ, used by DL reasoners) and the **RDF-Based Semantics** (applies to any RDF graph). Profiles are syntactic subsets.

8.2 RDFS: the minimum vocabulary

- **rdfs:Class**, **rdfs:subClassOf**
- **rdfs:subPropertyOf**
- **rdfs:domain**, **rdfs:range** (inference!)
- **rdfs:label**, **rdfs:comment**, **rdfs:seeAlso**

Many enterprise models start in RDFS + SKOS and add OWL axioms only where a competency question needs them.

8.3 OWL 2 constructs used most

- equivalentClass, disjointWith, inverseOf
- Functional / Transitive / Symmetric properties
- someValuesFrom, allValuesFrom, qualified cardinality
- hasKey, propertyChainAxiom, owl:deprecated

8.4 Axioms in Turtle

```
# prefixes omitted
clm:Claim rdfs:subClassOf fnd:Event ,
   [ a owl:Restriction ;                # every claim is on
     owl:onProperty clm:onPolicy ;      # exactly one policy
     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
     owl:onClass ins:Policy ] .
clm:hasClaimLine owl:inverseOf clm:partOfClaim .
fnd:Person owl:disjointWith fnd:Organization .
clm:Claim owl:hasKey ( clm:claimNumber ) .
```

8.5 OWL 2 Profiles (W3C 2012): pick by workload

Profile	Built for	Notable restrictions	Enterprise fit
EL	very large class hierarchies; polynomial reasoning. Spec names SNOMED CT	no allValuesFrom, cardinality, union, inverse	big terminologies, product / peril catalogs
QL	query rewriting over relational DBs (LogSpace)	no cardinality, chains, functional, keys	virtual KG over existing SQL (OBDA)
RL	rule engines working directly on triples	restricted positions; no disjoint union	triplestore materialization of inferences
DL	full expressivity with decidability	slower; tableau reasoners	design-time checking in the editor

Decision rule: Design in OWL 2 DL, check with a reasoner in CI, then deploy the subset your runtime supports (often RL rules in the triplestore, or QL for virtualization).

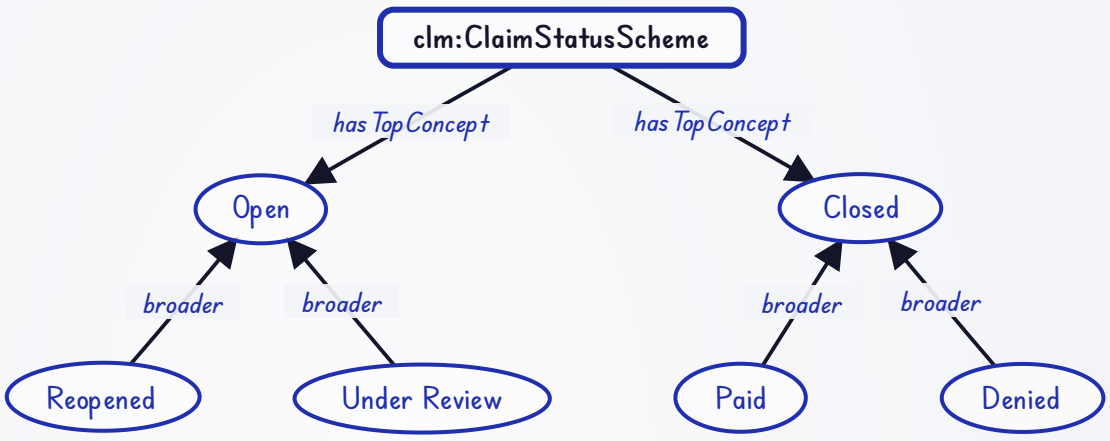
Sources: W3C OWL 2 Document Overview, 2nd ed. (11 Dec 2012) · W3C OWL 2 Profiles, 2nd ed. (2012) · W3C RDF Schema 1.1 (2014)

Rajesh Kumar Gupta

⑨ SKOS for Code Lists

Part A · Foundations of Ontology

9.1 A code list as a concept scheme



SKOS (W3C Recommendation, 2009) models **concepts**, not classes: perfect for statuses, reasons, perils, lines of business and other reference data that stewards change often.

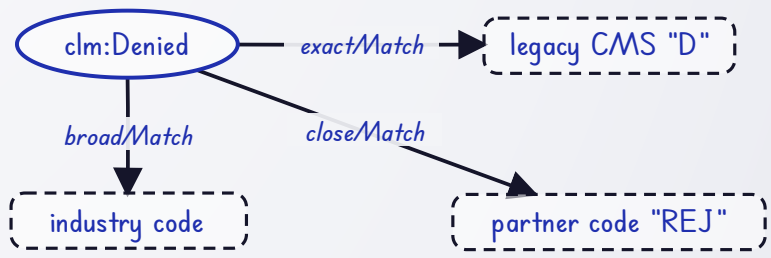
9.2 Turtle

```
# prefixes omitted
clm:ClaimStatusScheme a skos:ConceptScheme ;
   skos:prefLabel "Claim Status"@en ;
   skos:hasTopConcept clm:Open , clm:Closed .
clm:Open a skos:Concept ;
   skos:inScheme clm:ClaimStatusScheme ;
   skos:topConceptOf clm:ClaimStatusScheme ;
   skos:prefLabel "Open"@en ; skos:altLabel "Active"@en ;
   skos:notation "OPEN" ;
   skos:definition "Claim accepted for handling; not yet settled."@en .
clm:Denied a skos:Concept ;
   skos:inScheme clm:ClaimStatusScheme ;
   skos:prefLabel "Denied"@en ; skos:broader clm:Closed ;
   skos:exactMatch <https://codes.example.com/legacy-cms/status/D> .
```

9.3 Labels & notations

Property	Use
skos:prefLabel	display name; max one per language
skos:altLabel	synonyms, old names
skos:hiddenLabel	misspellings, for search only
skos:notation	the code value, e.g. "OPEN"
skos:definition	steward-approved meaning

9.4 Crosswalks to source codes



Mapping properties: exactMatch, closeMatch, broadMatch, narrowMatch, relatedMatch. The external code IRIs above are illustrative.

9.5 Gotchas & decision rule

- **skos:broader is not transitive**; query with skos:broaderTransitive or a path *skos:broader+*.
- **Concept vs class**: if members need their own properties and axioms, make an OWL class; if it is a value picked from a list, make a SKOS concept.
- Prefer **exactMatch** only after steward review; it is transitive, so errors spread.

Key Takeaway: Model enterprise code lists as SKOS concept schemes with one steward per scheme, and keep every source-system code as a mapped concept, never a hard-coded string.

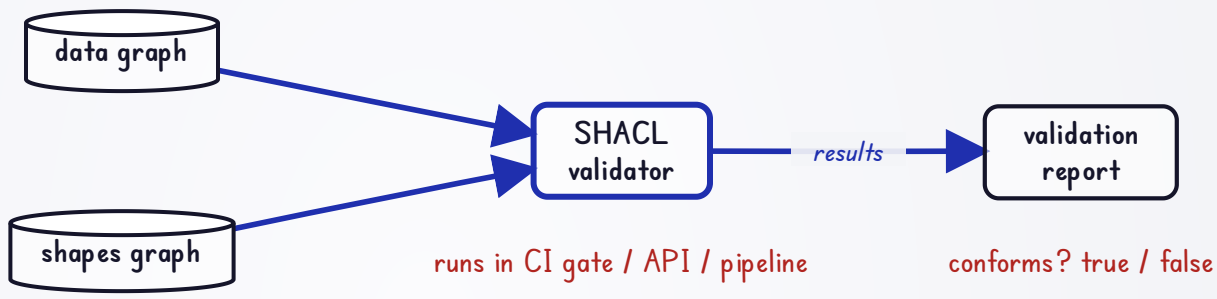
Sources: W3C SKOS Simple Knowledge Organization System Reference (18 Aug 2009) · W3C SKOS Primer (2009)

Rajesh Kumar Gupta

⑩ SHACL: Shapes as Data Contracts

Part A · Foundations of Ontology

10.1 How validation works



SHACL (W3C Recommendation, July 2017) checks an RDF **data graph** against a **shapes graph** under **closed-world** conditions. Shapes are themselves RDF, so they are versioned and reviewed like the ontology. Think of them as the data contract.

10.2 A claim shape

```
# prefixes omitted  (sh: = SHACL, clmsh: = clm shapes)
clmsh:ClaimShape a sh:NodeShape ;
    sh:targetClass clm:Claim ;
    sh:property [
        sh:path clm:onPolicy ;
        sh:minCount 1 ; sh:maxCount 1 ;
        sh:class ins:Policy ;
        sh:message "Exactly one policy required." ] ;
    sh:property [
        sh:path clm:reportedOn ;
        sh:datatype xsd:date ; sh:minCount 1 ] ;
    sh:property [
        sh:path clm:hasClaimStatus ;
        sh:in ( clm:Open clm:Reopened
                clm:Closed clm:Denied ) ;
        sh:severity sh:Warning ] .
```

10.3 Report (excerpt)

```
[ a sh:ValidationReport ;
  sh:conforms false ;
  sh:result [ a sh:ValidationResult ;
      sh:focusNode id:CLM-1002 ;
      sh:resultPath clm:onPolicy ;
      sh:resultSeverity sh:Violation ;
      sh:sourceConstraintComponent
          sh:MinCountConstraintComponent ;
      sh:sourceShape _:b0 ;
      sh:resultMessage "Exactly one policy required." ]
] .
```

Shape	Validates
NodeShape	the focus node itself
PropertyShape	values reached by sh:path

Targets: sh:targetClass, sh:targetNode, sh:targetSubjectsOf, sh:targetObjectsOf.

10.4 Severity: route, do not just fail

Severity	Meaning	Typical action
sh:Violation	default; a broken rule	block release / reject message
sh:Warning	non-critical	load, open a data-quality ticket
sh:Info	informative	dashboard only

SHACL 1.2 Core is a Working Draft (Nov 2026) and lists extra levels (sh:Trace, sh:Debug); check its status at w3.org before relying on them.

Key Takeaway: OWL says what a claim *is*; SHACL says what a claim record *must contain* for a given consumer. Keep one shape set per contract (API, feed, report).

Sources: W3C Shapes Constraint Language (SHACL) Recommendation (20 Jul 2017) · W3C SHACL 1.2 Core, Working Draft (2026)

⑪ SPARQL Essentials

Part A · Foundations of Ontology

11.1 Query forms

Form	Returns	Use at work
SELECT	table of variable bindings	reports, APIs, data extracts
CONSTRUCT	a new RDF graph	reshape to canonical / JSON-LD
ASK	true / false	tests, competency questions in CI
DESCRIBE	graph about a resource (engine-defined)	browsing, debugging
UPDATE	INSERT / DELETE data	controlled loads, fixes (SPARQL 1.1 Update)

11.2 SELECT with a property path

```
# prefixes omitted
SELECT ?claim ?policy ?reported
WHERE {
  ?claim a/rdfs:subClassOf* clm:Claim ;      # property path
         clm:onPolicy    ?policy ;
         clm:hasClaimStatus clm:Open ;
         clm:reportedOn ?reported .
  FILTER (?reported >= "2026-01-01"^^xsd:date)
}
ORDER BY DESC(?reported)  LIMIT 100
```

11.3 CONSTRUCT

```
CONSTRUCT { ?c fnd:hasIdentifier ?n }   # reshape
WHERE     { ?c a clm:Claim ; clm:claimNumber ?n }
```

11.4 ASK as a unit test

```
ASK { ?c a clm:Claim .                  # CQ test:
      FILTER NOT EXISTS { ?c clm:onPolicy ?p } }
# expected answer: false
```

11.5 Property paths

- **a/rdfs:subClassOf*** : instances of a class or its subclasses
- **skos:broader+** : all ancestors in a code list
- **^clm:onPolicy** : inverse (claims of a policy)
- **p1|p2** : alternatives

Paths replace hand-written recursion; watch cost of * on big graphs.

11.6 Federation with SERVICE

```
SELECT ?claim ?name WHERE {
  ?claim clm:hasClaimant ?party .
  SERVICE <https://party.example.com/sparql> {
    ?party fnd:legalName ?name } }
```



SERVICE sends part of the query to a remote endpoint

Status: SPARQL 1.1 (Mar 2013) is the Recommendation used by today's triplestores. SPARQL 1.2 was a Working Draft in Nov 2026; check w3.org before using 1.2 features.

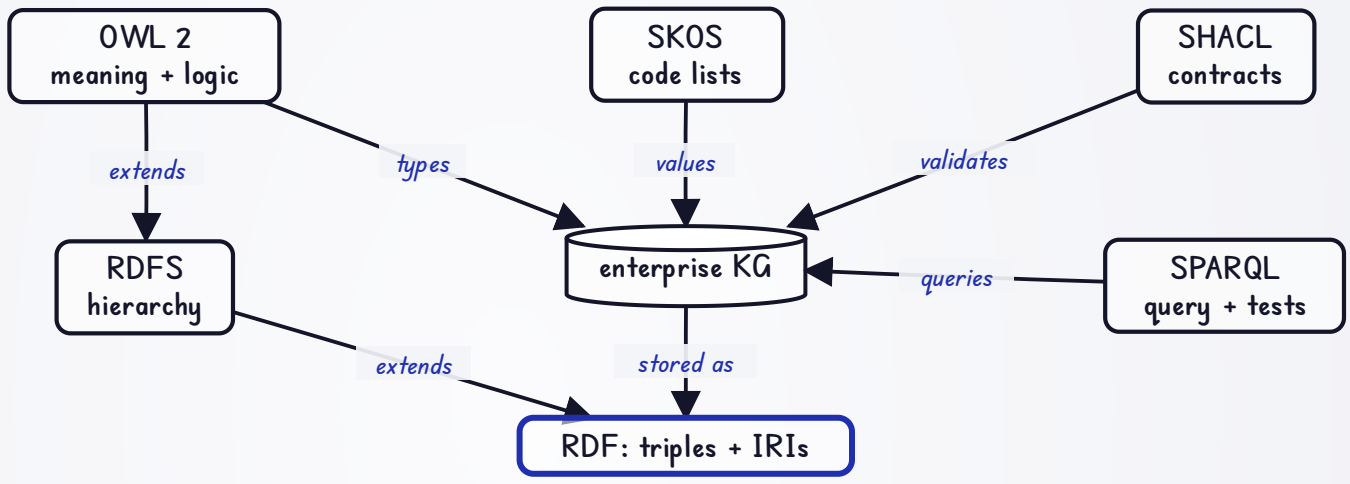
Sources: W3C SPARQL 1.1 Query Language (21 Mar 2013) · W3C SPARQL 1.1 Federated Query (2013) · W3C SPARQL 1.1 Update (2013) · W3C SPARQL 1.2 Query, Working Draft (2026)

Rajesh Kumar Gupta

⑫ Part A Recap + Checklist

Part A · Foundations of Ontology

12.1 The W3C toolkit on one page



12.2 Which standard for which job?

Job	Use	Not
Define what a Claim *is*	OWL 2 class + skos:definition	a JSON field comment
Allowed status codes	SKOS concept scheme	an OWL class per code (usually)
"Claim must have one policy"	SHACL sh:minCount / sh:maxCount	OWL cardinality alone
Find / test / reshape data	SPARQL SELECT / ASK / CONSTRUCT	custom code per app
Exchange & store facts	RDF (Turtle, N-Quads, JSON-LD)	proprietary dumps

12.3 Architect's checklist

1. Business terms traced; one IRI per real thing
2. Studer et al. definition in the charter
3. Level chosen on the spectrum by CQs
4. TBox and ABox in separate graphs
5. Every term: label, definition, parent, owner
6. OWA understood; SHACL for "must"
7. Code lists as SKOS with mappings
8. Profile chosen (EL/QL/RL/DL)
9. CQs as SPARQL ASK tests in CI

12.4 Common mistakes

- Using rdfs:range as validation
- owl:sameAs for "similar" records
- One OWL class per code value
- Labels without definitions
- Mixing instance data into the ontology file
- Building RDF 1.2 / SPARQL 1.2 features into production before checking their W3C status

Key Takeaway: Part A gives the vocabulary; Part B compares it with other model types; Part C turns it into a canonical model that running systems can use.

Sources: W3C OWL 2 Overview (2012) · W3C SKOS Reference (2009) · W3C SHACL (2017) · W3C SPARQL 1.1 (2013)

⑬ Conceptual, Logical, Physical

Part B · Kinds of Models

13.1 Three levels every architect uses

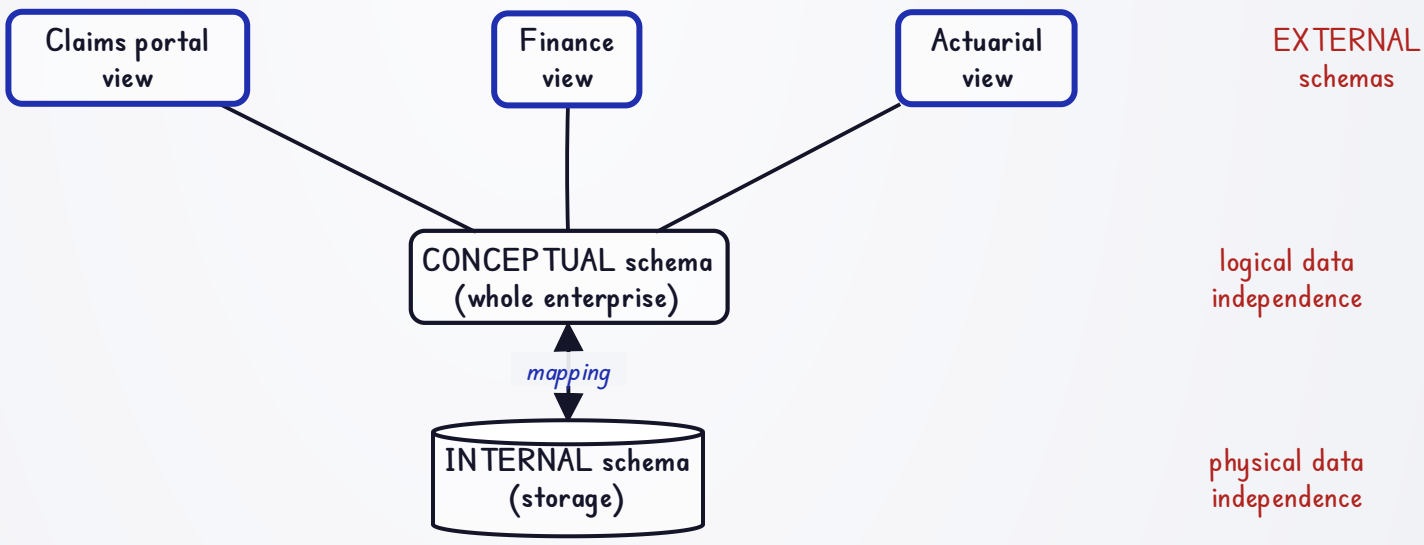
Conceptual model
business things & relationships • no keys, no types • audience: business

Logical model
entities, attributes, keys, cardinality, normal form • technology-neutral

Physical model
tables, columns, data types, indexes, partitions • one DBMS

DAMA-DMBOK2 (ch. 5, Data Modeling & Design) describes models at these three levels. Each level **adds decisions**: conceptual = *what* things exist; logical = *how* they are structured and identified; physical = *where/how* they are stored in one product.

13.2 Root: ANSI/X3/SPARC three-schema (1975)



The ANSI/X3/SPARC interim report (1975) split a database into **external** views, one **conceptual** schema and an **internal** (storage) schema, so storage can change without breaking users (data independence). It was a reference architecture, never adopted as a formal standard.

Naming trap: ANSI/SPARC "conceptual schema" ≈ today's **logical** model of the whole enterprise. Say which meaning you use in workshops.

13.3 Same example, three levels

Level	Same insurance fact: *a Party holds a Policy; a Claim is made on a Policy*
Conceptual	**Party** –holds→ **Policy** ←made on– **Claim** (boxes & verbs only)
Logical	POLICY(policy_id, policy_number, effective_date); CLAIM(claim_id, policy_id FK, loss_date); PARTY_POLICY_ROLE(party_id, policy_id, role_code)
Physical	**CREATE TABLE** claim (claim_id BIGINT PRIMARY KEY, policy_id BIGINT NOT NULL REFERENCES policy, loss_date DATE) … + index on policy_id, partition by loss year

13.4 Where does the ontology sit?

- An ontology is a **machine-readable conceptual model** with formal semantics (classes, properties, axioms).
- Logical & physical models are **derived from** or **mapped to** it – they do not replace it.
- Monday rule: every logical entity should trace to an ontology class (keep a mapping column / annotation).

Key Takeaway: Keep the levels separate: a conceptual model that shows foreign keys, or a physical model used as "the business model", is the most common source of confusion.

Sources: ANSI/X3/SPARC Study Group on DBMS, Interim Report (1975) • DAMA International, DAMA-DMBOK2 (2017)

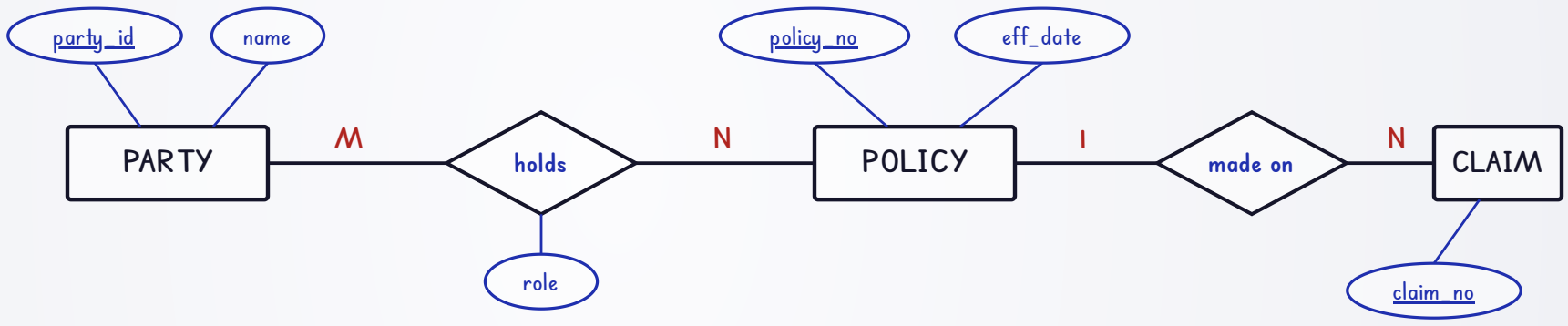
Rajesh Kumar Gupta

⑭ Entity-Relationship Modeling

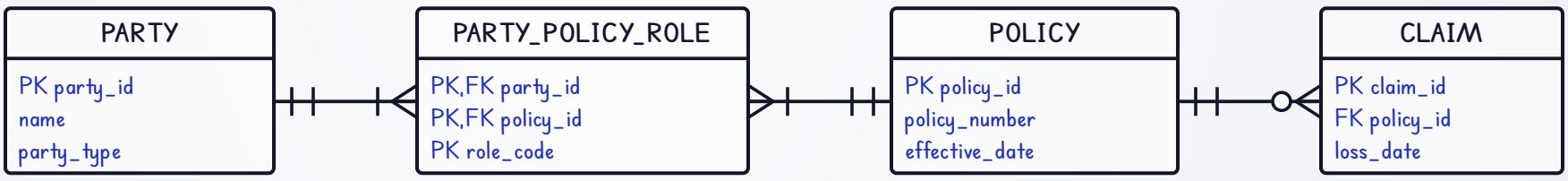
Part B · Kinds of Models

14.1 Chen notation (1976)

Peter Chen's ACM TODS paper introduced the **entity-relationship model**: entities (rectangles), relationships (diamonds) and attributes (ovals). Relationships are first-class and may carry attributes – here *role* sits on *holds*.



14.2 Crow's-foot (IE) notation: same model, logical level



legend: || exactly one · |< one or many · o< zero or many

The M:N *holds* becomes an **associative entity** PARTY_POLICY_ROLE with a role_code (policyholder, additional insured, payer...). Keep this in mind: it is the same idea as the ontology **role pattern** (topic 23).

14.3 Normalization basics (Codd)

Normal form	Rule (informal)	Insurance violation → fix
1NF	atomic values, no repeating groups	CLAIM.payment1, payment2... → PAYMENT table
2NF	non-key attributes depend on the *whole* key	role table holds policy_number → move to POLICY
3NF	no non-key → non-key dependency	CLAIM.status_code + status_name → STATUS lookup

14.4 ER model vs ontology – at work

- ER: **closed world**, keys and NOT NULL are hard constraints.
- ER: one entity per row; no inference.
- Names are local to one schema.
- OWL: open world; constraints go in **SHACL**.
- An individual can have many types; classes can be inferred.
- Terms are global IRIs, reusable everywhere.

Key Takeaway: Reverse-engineer existing ER models into a draft class list (entity → class, FK → object property, column → datatype property), then **review meaning** with stewards – never ship the raw conversion.

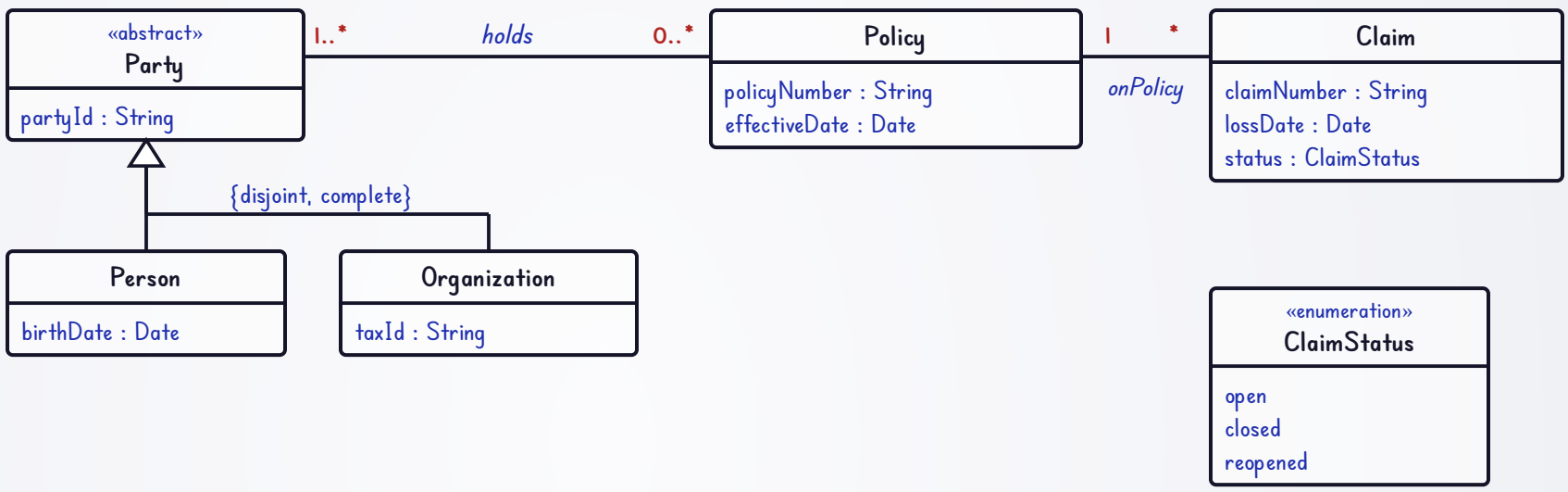
Sources: Chen, "The Entity-Relationship Model – Toward a Unified View of Data", ACM TODS 1(1) (1976) · Codd, "A Relational Model of Data for Large Shared Data Banks", CACM (1970) · DAMA-DMBOK2 (2017)

Rajesh Kumar Gupta

⑮ UML Class Diagrams vs OWL

Part B · Kinds of Models

15.1 The running example as a UML class diagram



15.2 What maps – and what does not

UML 2.5.1 element	OWL 2 / RDF counterpart	Watch out
Class	owl:Class	OO objects usually have one class; OWL individuals may have many types
Attribute	owl:DatatypeProperty (+ rdfs:domain/range)	OWL properties are **global**, not owned by a class
Association (end)	owl:ObjectProperty, owl:inverseOf	name both directions; domain/range *infer*, never reject
Multiplicity 1..*	owl:minQualifiedCardinality 1 … / **sh:minCount 1**	OWL (open world) will not flag a missing value – SHACL will
Generalization	rdfs:subClassOf	{disjoint, complete} → owl:disjointUnionOf
Enumeration	skos:ConceptScheme (or owl:oneOf)	code lists change – prefer SKOS
Association class	n-ary relation pattern (reified class)	see 23.2
Operations, visibility, abstract	no counterpart	keep in code / docs

15.3 Same model in OWL + SHACL

```
# prefixes omitted: owl rdfs xsd sh fnd ins clm clmsh
fnd:Party owl:disjointUnionOf
    ( fnd:Person fnd:Organization ) .
clm:onPolicy a owl:ObjectProperty ;
    rdfs:domain clm:Claim ;
    rdfs:range ins:Policy .
# UML multiplicity "1" -> SHACL
clmsh:ClaimShape a sh:NodeShape ;
    sh:targetClass clm:Claim ;
    sh:property [ sh:path clm:onPolicy ;
        sh:minCount 1 ; sh:maxCount 1 ] .
```

15.4 OMG bridge: ODM

The OMG **Ontology Definition Metamodel** (ODM 1.1, 2014) gives metamodels and profiles for RDF and OWL, so UML tools can carry ontology models. Useful when your EA team lives in a UML repository.

- UML = design of **software**.
- OWL = meaning of **things**.

Key Takeaway: Multiplicities belong in SHACL shapes (data contracts); class meaning belongs in OWL. Splitting them avoids the classic "OWL did not catch my missing field" surprise.

Sources: OMG UML 2.5.1 (Dec 2017) · OMG Ontology Definition Metamodel 1.1 (2014) · W3C OWL 2 Primer, 2nd ed. (2012) · W3C SHACL Recommendation (2017)

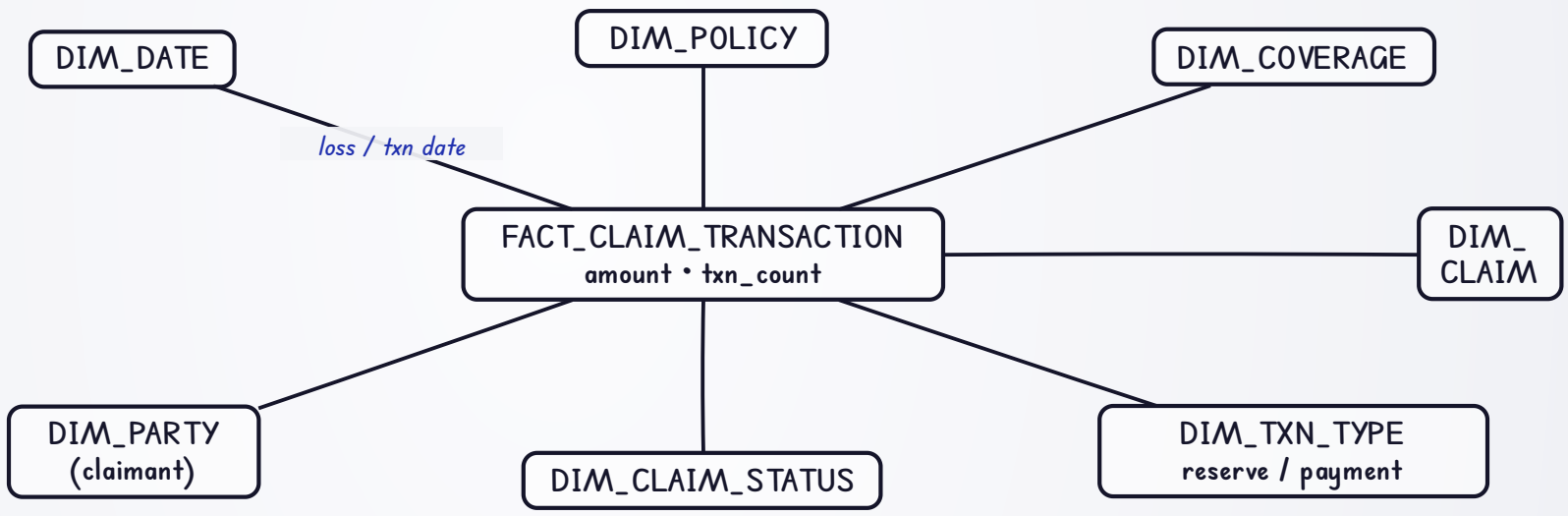
⑯ Dimensional Models

Part B · Kinds of Models

16.1 Kimball's four-step design

1 Select the business process — claim handling → 2 Declare the grain — one row per… → 3 Identify the dimensions — who, what, when → 4 Identify the facts — numbers

16.2 Star schema for claims (illustrative)



Grain (illustrative): *one row per financial transaction (reserve change or payment) on one claim line.* Declare it in one sentence before choosing columns.

16.3 Facts & dimensions

- **Facts** = measurements at the grain: amount, count.
- Additive: payment amount. **Semi-additive**: reserve *balance* (sum over claims, not over time).
- **Dimensions** = context; flat, wide, denormalized.
- SCD Type 1 overwrites; Type 2 adds a row with effective dates to keep history.

16.4 Star vs snowflake

- **Star**: each dimension one table — simple, fast for BI.
- **Snowflake**: dimensions normalized into sub-tables (DIM_COVERAGE → COVERAGE_LINE).
- Kimball generally recommends star for usability.

16.5 Conformed dimensions = the bus matrix

Business process ↓ / dimension →	Date	Policy	Party	Coverage	Agent	Claim
Policy transactions	✓	✓	✓	✓	✓	
Premium billing	✓	✓	✓		✓	
Claim transactions	✓	✓	✓	✓		✓

A **conformed** dimension means the same thing (same keys, same labels) in every star. That is exactly what an ontology + SKOS code lists give you: DIM_CLAIM_STATUS rows = concepts of clm:ClaimStatusScheme.

Key Takeaway: Derive conformed dimensions from the ontology and canonical code lists; then "Party" and "Coverage" agree across underwriting, billing and claims marts.

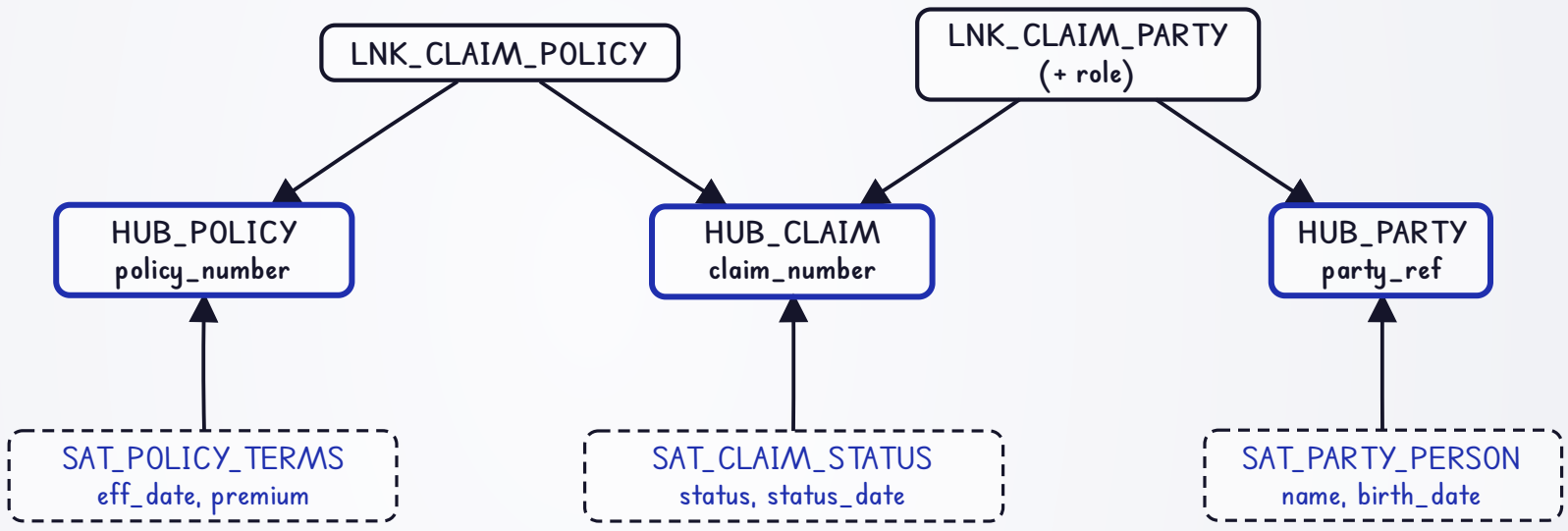
Sources: Kimball & Ross, The Data Warehouse Toolkit, 3rd ed., Wiley (2013) · Kimball Group, "Four-Step Dimensional Design Process" (kimballgroup.com)

⑰ Data Vault 2.0

Part B · Kinds of Models

17.1 Three building blocks

- **Hub** = a list of unique **business keys** for one core concept (Policy, Claim, Party).
- **Link** = an association between hubs (often M:N); its grain = the hub keys it joins.
- **Satellite** = descriptive attributes + their **full history** (insert-only) for a hub or link.



17.2 Standard columns

```
-- Data Vault 2.0 style (illustrative)
CREATE TABLE hub_claim (
  -- hash of the business key
  claim_hk      CHAR(32) PRIMARY KEY,
  claim_number  VARCHAR(20) NOT NULL,
  load_dts      TIMESTAMP NOT NULL,
  record_source VARCHAR(50) NOT NULL );
CREATE TABLE sat_claim_status (
  claim_hk  CHAR(32) REFERENCES hub_claim,
  load_dts  TIMESTAMP,
  -- hash of attributes: change detection
  hash_diff CHAR(32),
  status_code   VARCHAR(10),
  record_source VARCHAR(50),
  PRIMARY KEY (claim_hk, load_dts) );
```

17.3 What 2.0 added

- **Hash keys** computed from business keys replace sequence numbers → parallel, cross-platform loading.
- **Raw Vault** stores data as received; **Business Vault** adds derived/cleansed structures.
- **PIT** (point-in-time) and **bridge** tables speed up queries over many satellites.
- Also covers methodology & architecture, not only modeling.

17.4 Ontology ↔ Data Vault

- Hub ↔ class with an identity rule (business key = identifier policy).
- Link ↔ object property, or n-ary pattern if it has attributes.
- Satellite ↔ datatype properties, versioned in time.

17.5 Where it sits

Sources (PAS, claims, CRM) → Staging (+ hash keys) → Raw Vault (as received) → Business Vault (rules, PIT, bridge) → Marts (stars, APIs)

Key Takeaway: Let the ontology name the hubs: one hub per concept that has a stable business key. A hub per source table is a smell — the vault becomes a copy of sources, not an integration layer.

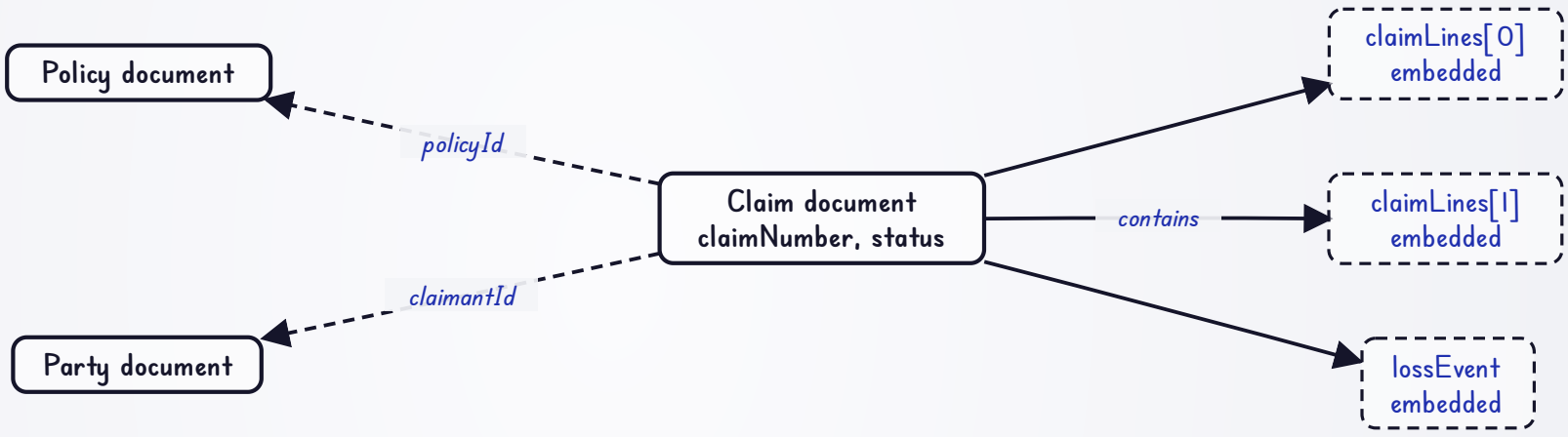
Sources: Linstedt & Olschimke, Building a Scalable Data Warehouse with Data Vault 2.0, Morgan Kaufmann (2015)

⑱ Document Models: JSON & Avro

Part B · Kinds of Models

18.1 Document model: aggregates

A document store keeps one **aggregate** per document. Embed what is owned and read together (claim lines, loss event); **reference** (dashed) what has its own life cycle (policy, party).



18.2 JSON Schema (API payload)

```
{ "$schema": "https://json-schema.org/draft/2020-
12/schema",
  "$id": "https://ontology.example.com/schema/clm/Claim",
  "type": "object",
  "required": [ "claimNumber", "policyId", "status"],
  "properties": {
    "claimNumber": { "type": "string" },
    "policyId":    { "type": "string", "format": "iri" },
    "lossDate":    { "type": "string", "format": "date" },
    "status": { "enum": [ "open", "closed", "reopened"] },
    "claimLines": { "type": "array",
                    "items": { "$ref": "#/$defs/ClaimLine" } }
},
  "$defs": { "ClaimLine": { "type": "object",
    "properties": { "amount": { "type": "number" } } } } }
```

18.3 Avro (Kafka event)

```
{ "type": "record",
  "name": "ClaimReported",
  "namespace": "com.example.ontology.clm",
  "fields": [
  { "name": "claimNumber", "type": "string" },
  { "name": "policyId", "type": "string" },
  { "name": "lossDate", "type":
    { "type": "int", "logicalType": "date" } },
  { "name": "status", "type":
    { "type": "enum", "name": "ClaimStatus",
      "symbols": ["OPEN","CLOSED","REOPENED"] } },
  { "name": "adjusterId",
    "type": ["null", "string"],
    "default": null } ] }
```

- Adding a field with a **default** lets readers on the new schema still read old data.
- logicalType "date" = days since 1970-01-01 in an int.

18.4 Side by side

	JSON Schema 2020-12	Avro 1.12	OWL + SHACL
Purpose	validate JSON documents / APIs	compact binary records, events	meaning + data contract
Identity	none built in (strings)	none built in	IRIs
Optional field	omit from "required"	union with "null" + default	sh:minCount 0
Evolution	by convention	reader/writer schema resolution rules	versionIRI, owl:deprecated

Monday tip: Generate both schemas from the canonical model (Part C) and put the ontology IRI in "$id" / namespace or a custom annotation, so every field traces back to one meaning.

Sources: JSON Schema 2020-12 (json-schema.org) · Apache Avro Specification 1.12.0 (avro.apache.org)

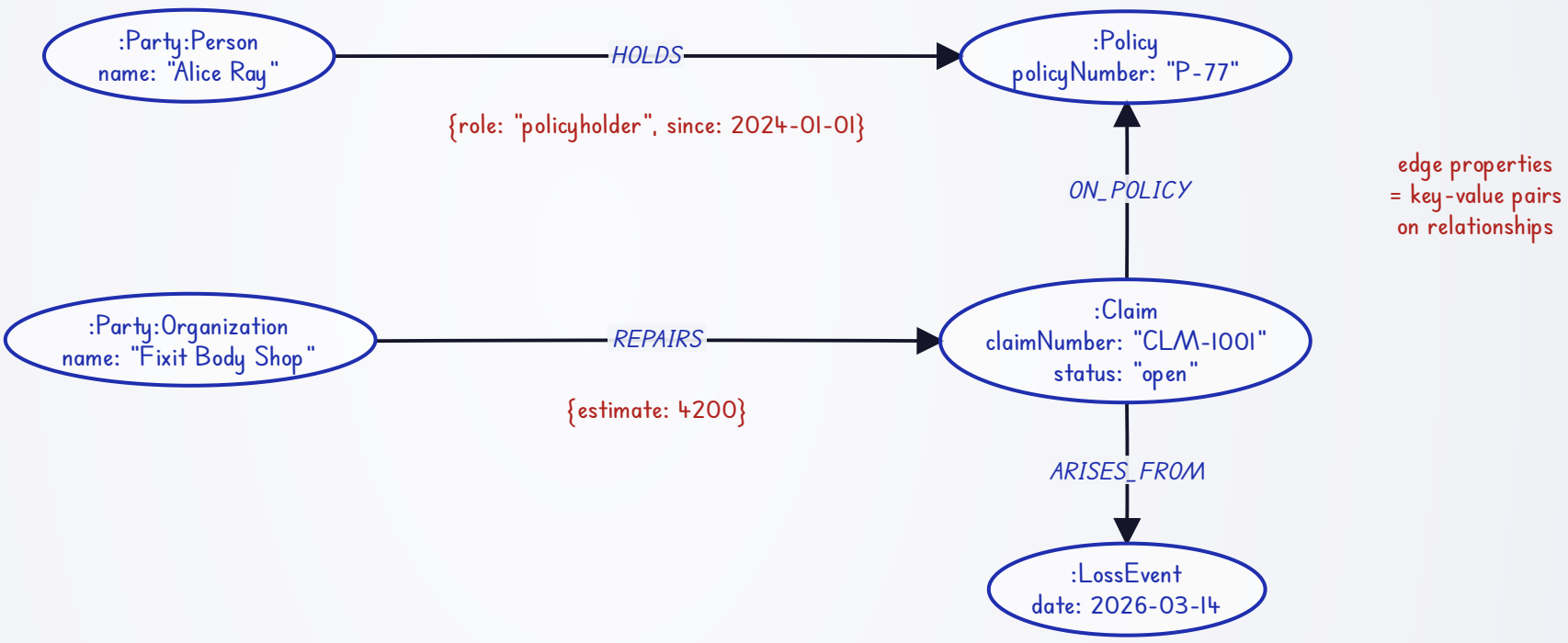
Rajesh Kumar Gupta

⑲ Labeled Property Graphs & GQL

Part B · Kinds of Models

19.1 The model in one sentence

A **labeled property graph** (LPG) has **nodes** with zero or more **labels**, **relationships** with a type and a direction, and **key-value properties** on both.



19.2 Write it

```
// Cypher (Neo4j) – create
CREATE (a:Party:Person {name:'Alice Ray'})
  -[:HOLDS {role:'policyholder',
            since:date('2024-01-01')}]->
  (p:Policy {policyNumber:'P-77'})
CREATE (c:Claim {claimNumber:'CLM-1001',
                 status:'open'})
  -[:ON_POLICY]->(p)
```

19.3 Query it

```
// ISO GQL – query (same pattern style)
MATCH (a:Person)-[h:HOLDS]->(p:Policy)
      <-[:ON_POLICY]-(c:Claim)
WHERE c.status = 'open'
RETURN a.name, h.role, p.policyNumber
```

19.4 Standards – now real

Standard	What it is
ISO/IEC 39075:2024 **GQL**	stand-alone property-graph query language (JTC 1/SC 32), published April 2024
ISO/IEC 9075-16:2023 **SQL/PGQ**	property-graph queries *inside* SQL over existing tables
openCypher / Cypher	Neo4j-originated language; strongly influenced GQL pattern syntax

19.5 Strengths at work

- Edge properties are natural (role, since, estimate).
- Fast path traversal: fraud rings, repair networks, claimant ↔ provider links.
- Developer-friendly; many visual tools.

19.6 Gaps to plan for

- IDs are database-local – no global IRIs.
- Labels carry no formal semantics or inference.
- Schema optional (GQL adds *graph types*); validation is product-specific.

Key Takeaway: Use the ontology to **name** labels and relationship types (Policy, ON_POLICY = clm:onPolicy) and keep that mapping in the canonical model; the LPG then stays aligned with the enterprise meaning.

Sources: ISO/IEC 39075:2024 Information technology – Database languages – GQL · ISO/IEC 9075-16:2023 SQL/PGQ · Neo4j Cypher Manual (neo4j.com/docs)

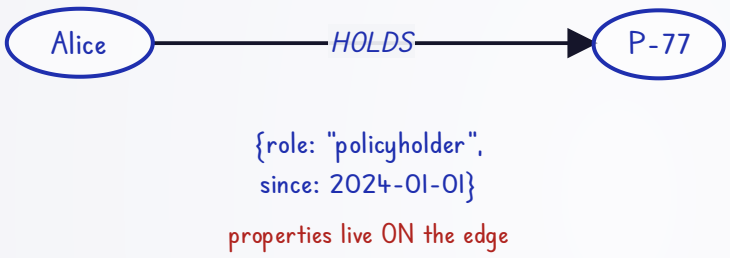
20 RDF Graph vs Property Graph

Part B · Kinds of Models

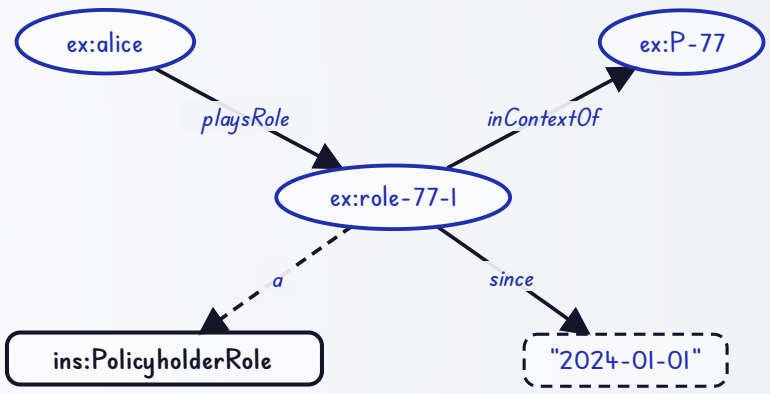
20.1 One fact, two graph models

Fact: *Alice holds policy P-77 as policyholder since 2024-01-01*

LPG



RDF – role pattern (topic 23)



20.2 RDF 1.2: statements about statements

```
# RDF 1.2 Turtle – annotate an asserted triple (prefixes omitted)
VERSION "1.2"
ex:alice ins:holdsPolicy ex:P-77 ~ ex:h1
   {| ins:role ins:Policyholder ; ins:since "2024-01-01"^^xsd:date |} .
# means: the triple is asserted, and reifier ex:h1  rdf:reifies  <<( ex:alice ins:holdsPolicy ex:P-77 )>>
```

RDF 1.2 adds **triple terms** and **reifiers** (the RDF-star work). RDF 1.2 Concepts reached Candidate Recommendation in April 2026; Turtle 1.2 was still a Working Draft in Nov 2026 – check current status at w3.org and your store's support before relying on it.

20.3 Side-by-side

Aspect	RDF graph	Labeled property graph
Identity	global IRIs, mergeable across systems	internal node IDs
Meaning	RDFS/OWL semantics, inference	labels only; meaning in code/docs
Edge data	n-ary node, or RDF 1.2 reifiers	native edge properties
Validation	SHACL (W3C)	DB-specific constraints, GQL graph types
Query	SPARQL 1.1 (1.2 in progress)	Cypher, ISO GQL, SQL/PGQ
Exchange	Turtle, JSON-LD, N-Triples ...	vendor formats, CSV, JSON
Federation	SPARQL SERVICE, linked data	limited / product-specific

20.4 Decision rule

- Integration, shared meaning, reasoning, cross-org exchange → **RDF + OWL + SHACL**.
- Deep traversal analytics inside one application team → **LPG** is often simpler.
- Both? Keep the ontology as the source of names; project the RDF KG into an LPG (or query LPG data through mappings) – do not maintain two meanings.

Key Takeaway: The choice is about **identity and meaning across systems**, not about "graph speed".

Sources: W3C RDF 1.2 Concepts and Abstract Data Model, CR (Apr 2026) · W3C RDF 1.2 Turtle, WD (2026) · W3C SPARQL 1.1 Query Language (2013) · ISO/IEC 39075:2024 GQL

Rajesh Kumar Gupta

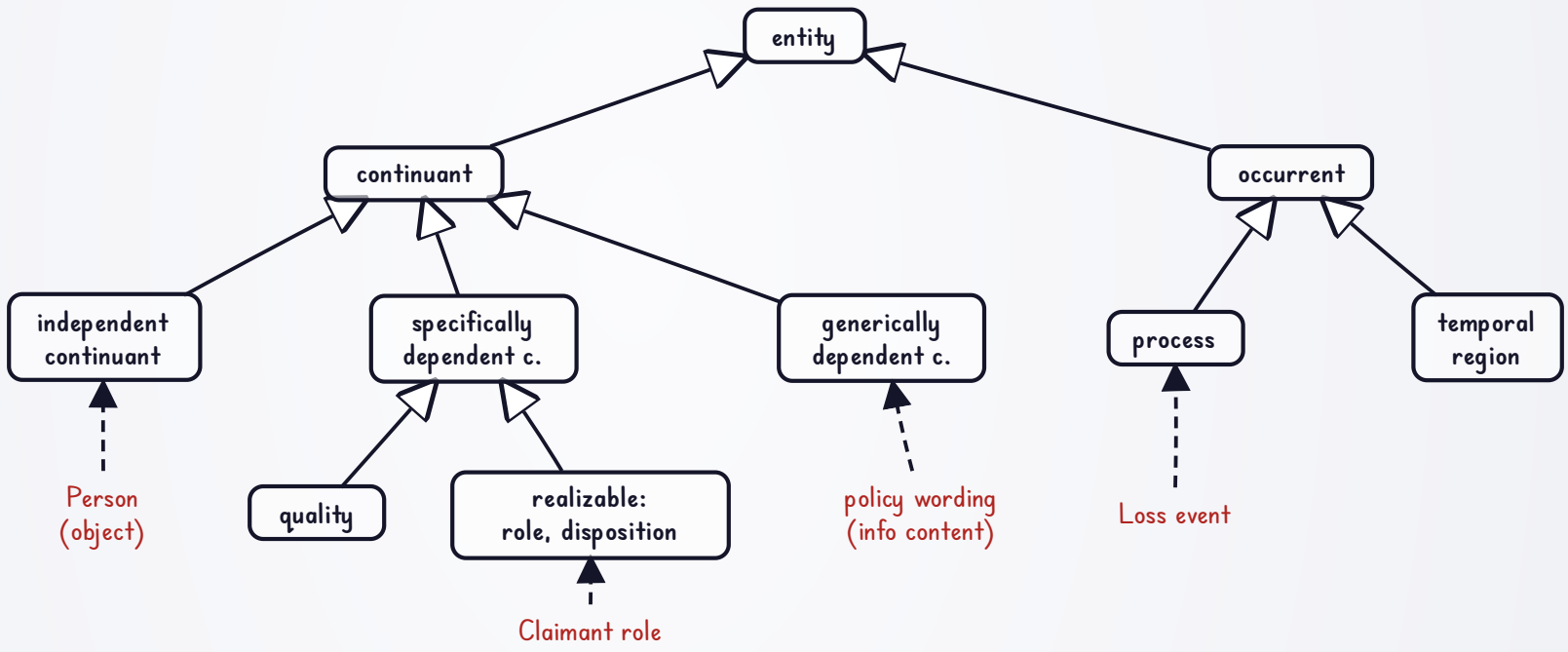
㉑ Upper / Foundational Ontologies

Part B · Kinds of Models

21.1 Why an upper ontology?

- Gives every domain module the **same top categories** (thing vs event vs role vs information).
- Forces early answers: is a Policy a *document*, an *agreement*, or both?
- Makes separately built modules easier to align later.

21.2 BFO skeleton with insurance examples (red = our classes)



BFO 2020 is deliberately small (a few dozen classes). Roles are **specifically dependent** on their bearer: the claimant role exists only while a Person bears it. Placements shown are modeling choices — verify with BFO guidance (Arp, Smith & Spear 2015) and your experts.

21.3 BFO, DOLCE, gist compared

	BFO	DOLCE	gist
Owner / form	ISO/IEC 21838-2:2021 (BFO 2020)	ISO/IEC 21838-3:2023	Semantic Arts, CC BY 4.0, OWL 2 DL
Stance	realist: types in reality	descriptive: categories of language & common sense	pragmatic: minimal business primitives
Top split	continuant / occurrent	endurant / perdurant / quality / abstract	e.g. Agreement, Commitment, Event, Person, Organization, Category, Magnitude
Typical home	biomedical (OBO Foundry) and other BFO-based suites	research; basis of DUL (DOLCE+DnS Ultralite) used by many ODPs	enterprise data / IT

21.4 How to choose (practical)

1. **Partners** : reuse what your industry ontologies already import (check their owl:imports).
2. **Team skill** : BFO/DOLCE need ontology training; gist is quicker for business IT teams.
3. **Scope** : importing a whole upper ontology is not required — you may align only your top classes (rdfs:subClassOf) and document it.

Key Takeaway: Decide the upper ontology **once**, at the enterprise layer (fnd:), never per project.

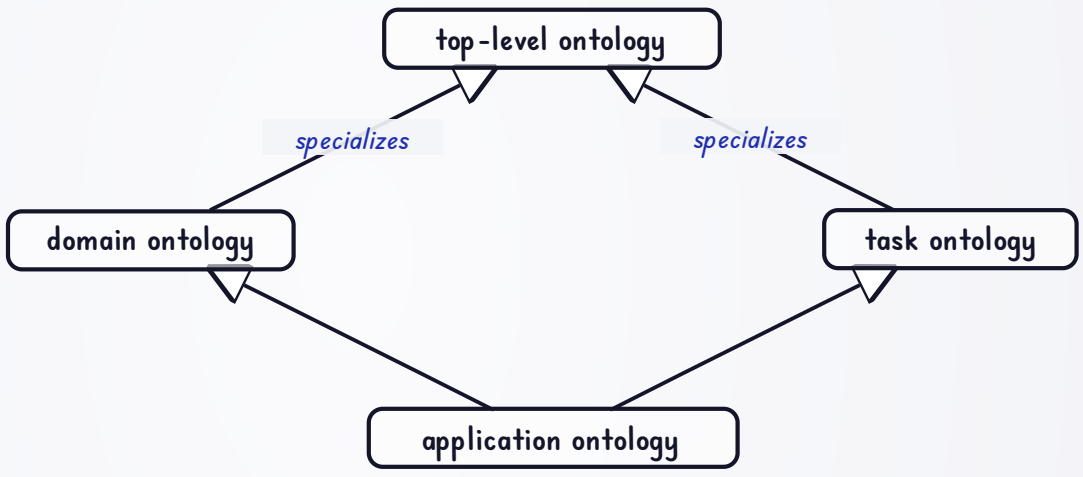
Sources: ISO/IEC 21838-2:2021 (BFO) · Arp, Smith & Spear, Building Ontologies with Basic Formal Ontology, MIT Press (2015) · Borgo et al., "DOLCE: A Descriptive Ontology for Linguistic and Cognitive Engineering", Applied Ontology (2022); ISO/IEC 21838-3:2023 · Semantic Arts, gist (github.com/semanticarts/gist)

㉒ Upper, Core, Domain, Task, Application

Part B · Kinds of Models

22.1 Guarino's kinds of ontology

Guarino (FOIS 1998) defines an ontology as "*a logical theory accounting for the intended meaning of a formal vocabulary*" and orders ontologies by generality. Domain and task ontologies specialize the top level; application ontologies specialize both.

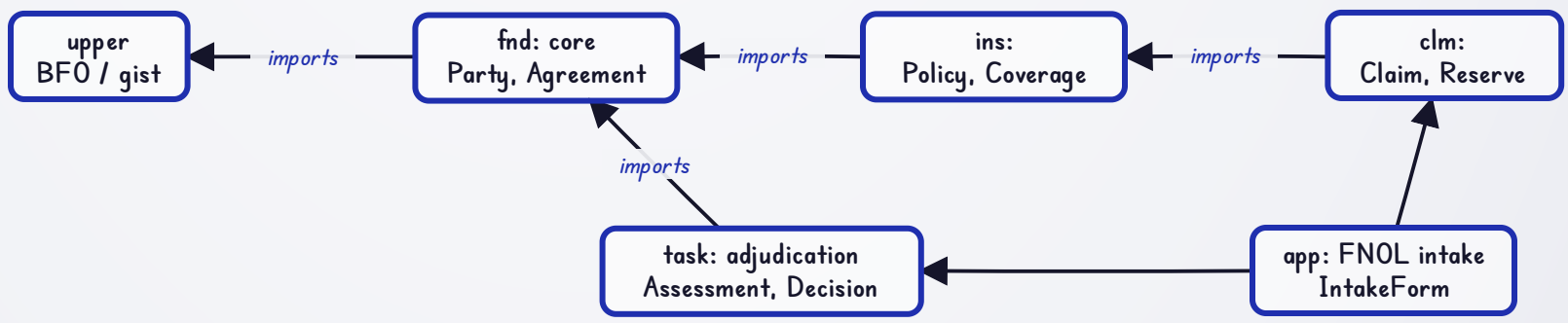


22.2 Each kind, insurance-style

Kind (Guarino 1998)	What it covers	Insurance example
Top-level	very general: space, time, object, event, action	BFO / DOLCE / gist
*Core / enterprise**	shared business primitives	fnd: Party, Agreement, Event, Money
Domain	vocabulary of a generic domain	ins: Policy, Coverage; clm: Claim, LossEvent
Task	vocabulary of a generic task or activity	adjudication: Assessment, Evidence, Decision
Application	one domain + one task, for one system	FNOL app: IntakeForm, ReplaceablePart

* "Core/enterprise" is a practitioner layer between top-level and domain; it is **not** in Guarino's 1998 figure, but most enterprise programs need it.

22.3 The same layers as an import graph



22.4 Placement rules for Monday

1. **Most general true layer** : put each term at the highest layer where its definition is still true (Party in fnd:, not clm:).
2. **Imports point upward only** : domain never imports an application ontology; no cycles.
3. **Task ≠ domain** : adjudication terms (Decision, Evidence) are reusable for underwriting too — keep them out of clm:.
4. **Application = thin** : mostly SHACL shapes & labels over shared classes; new classes need a reason.

Key Takeaway: A clear layer map is what lets departments reuse concepts instead of re-inventing them.

Sources: Guarino, "Formal Ontology and Information Systems", FOIS'98, IOS Press (1998) · W3C OWL 2 Structural Specification (2012) — owl:imports

Rajesh Kumar Gupta

23 Ontology Design Patterns

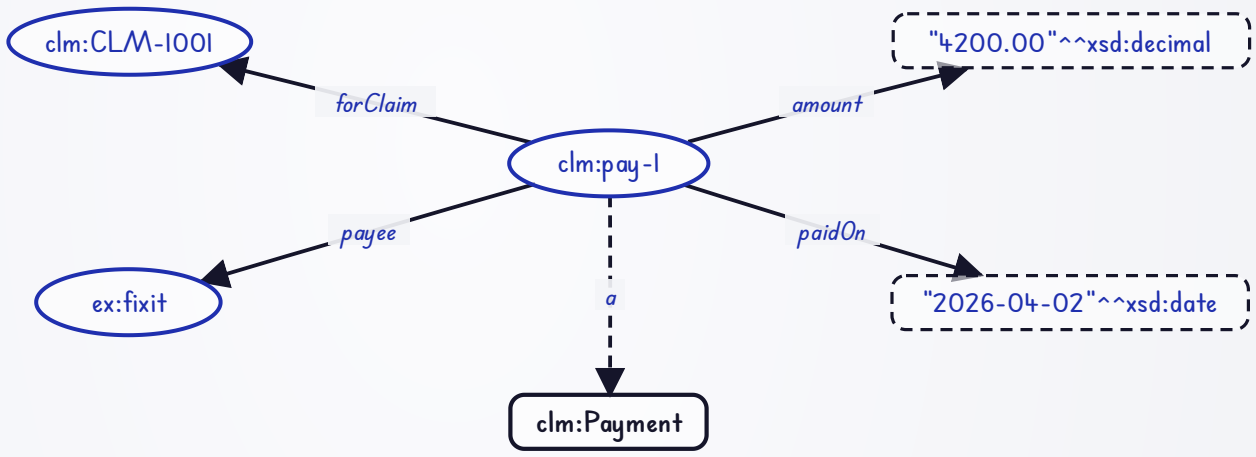
Part B · Kinds of Models

23.1 What is an ODP?

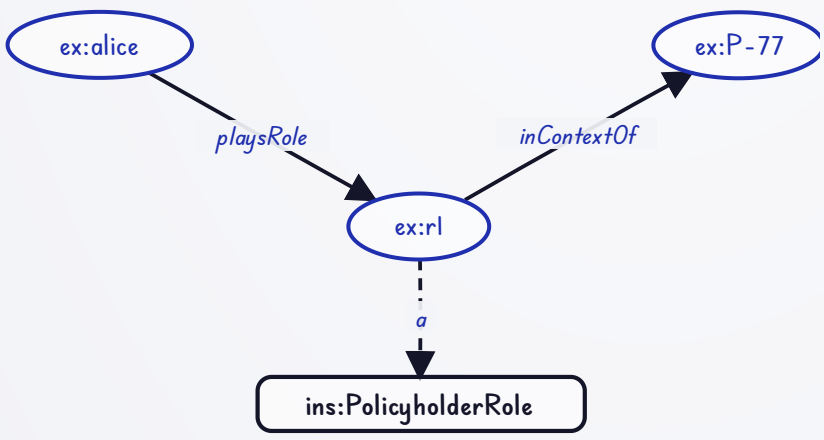
An **ontology design pattern** is a small, reusable modeling solution to a recurring problem (Hitzler et al. 2016; catalog at ontologydesignpatterns.org). Use patterns so different teams model the same situation the same way.

23.2 N-ary relation: a claim payment

"Claim CLM-1001 paid Fixit 4200 on 2 Apr 2026" has four participants. W3C Note pattern 1: **turn the relation into a class** (clm:Payment) with one property per participant.



23.3 Role pattern



Party stays a Party; the **role** carries type and context. This mirrors the ER associative entity (14.2).

23.4 Value partition

```
# Value partition (class pattern) — prefixes omitted
clm:Severity owl:disjointUnionOf
  ( clm:LowSeverity clm:MediumSeverity
    clm:HighSeverity ) .
clm:hasSeverity a owl:ObjectProperty ,
  owl:FunctionalProperty ;
  rdfs:range clm:Severity .
```

W3C Note: classes (partition) or individuals (value set). For changing business code lists prefer a **SKOS concept scheme**; keep partitions for stable, reasoning-relevant values.

23.5 Pattern cheat-sheet

Pattern	Problem it solves	Insurance use
N-ary relation	a relation with > 2 participants or its own attributes	Payment, PolicyParticipation
Role / participation	same party, many hats, in a context	policyholder, claimant, payee
Value partition	a closed set of exclusive qualities	severity, risk tier
Event	something happens, with time & participants	LossEvent, Notification
Part-whole	composition of things	Claim hasPart ClaimLine; Policy hasPart Coverage

Watch out: Part-whole needs care: make hasPart transitive only when the business really means it, and never mix it with membership (a Party is a *member* of a group, not a part).

Sources: W3C, Defining N-ary Relations on the Semantic Web, WG Note (2006) · W3C, Representing Specified Values in OWL: value partitions and value sets, WG Note (2005) · Hitzler, Gangemi, Janowicz, Krisnadhi & Presutti (eds.), Ontology Engineering with Ontology Design Patterns, IOS Press (2016) · ontologydesignpatterns.org

24 Model Comparison Matrix

Part B · Kinds of Models

24.1 The comparison matrix

Model	Best for	Semantics	Identity	Change cost*	Insurance example
Conceptual / ER	scoping, business agreement	informal	business keys	low	claims context diagram
Logical / 3NF	OLTP design	keys & constraints	PK / FK	medium	claims admin DB
UML class	software design	OO semantics	object refs	medium	claims service code
Dimensional	BI, reporting	facts & dimensions	surrogate keys	medium	loss-ratio mart
Data Vault 2.0	auditable integration history	hub / link / sat	hash of BK	low (additive)	claims history vault
JSON / Avro	APIs, events	structural	none built in	low–medium	FNOL API, ClaimReported event
LPG	traversal analytics	labels only	DB-local	low	fraud-ring detection
RDF + OWL + SHACL	shared meaning, integration, reasoning	formal (DL)	global IRIs	governed	enterprise ontology & KG

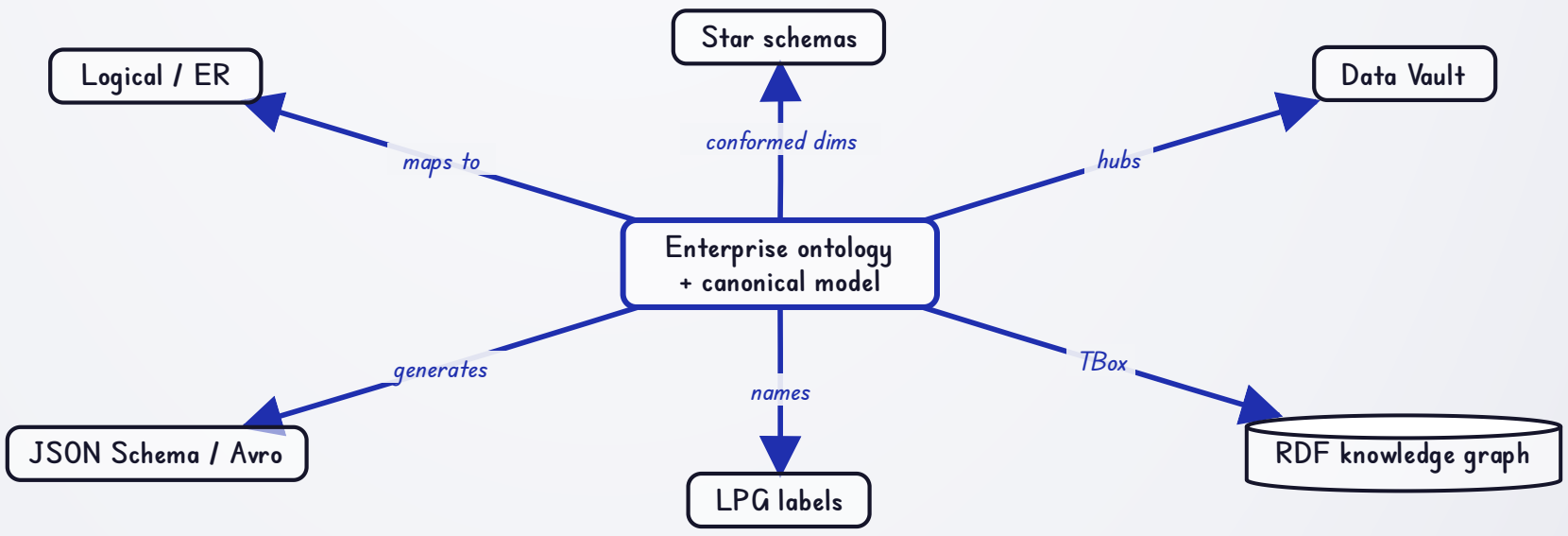
* change cost = author's rule of thumb, not a measured figure.

24.2 Quick decision path



...and in every case: names, codes and keys come from the ontology + canonical model

24.3 One meaning, many models



24.4 Rules of thumb

- Models are **complementary**; large enterprises typically run several side by side.
- Pick per **workload**, not per fashion.
- Only one of them should own **meaning** — the ontology.

24.5 Monday checklist

- List every model in your landscape and its owner.
- Add an "ontology term" column to each model's dictionary.
- Flag entities with no ontology mapping → backlog.

Key Takeaway: Every model on this page answers a different question; the canonical model (Part C) is the bridge that keeps their meanings consistent.

Sources: DAMA-DMBOK2 (2017) · Kimball & Ross (2013) · Linstedt & Olschimke (2015) · ISO/IEC 39075:2024 GQL

Rajesh Kumar Gupta

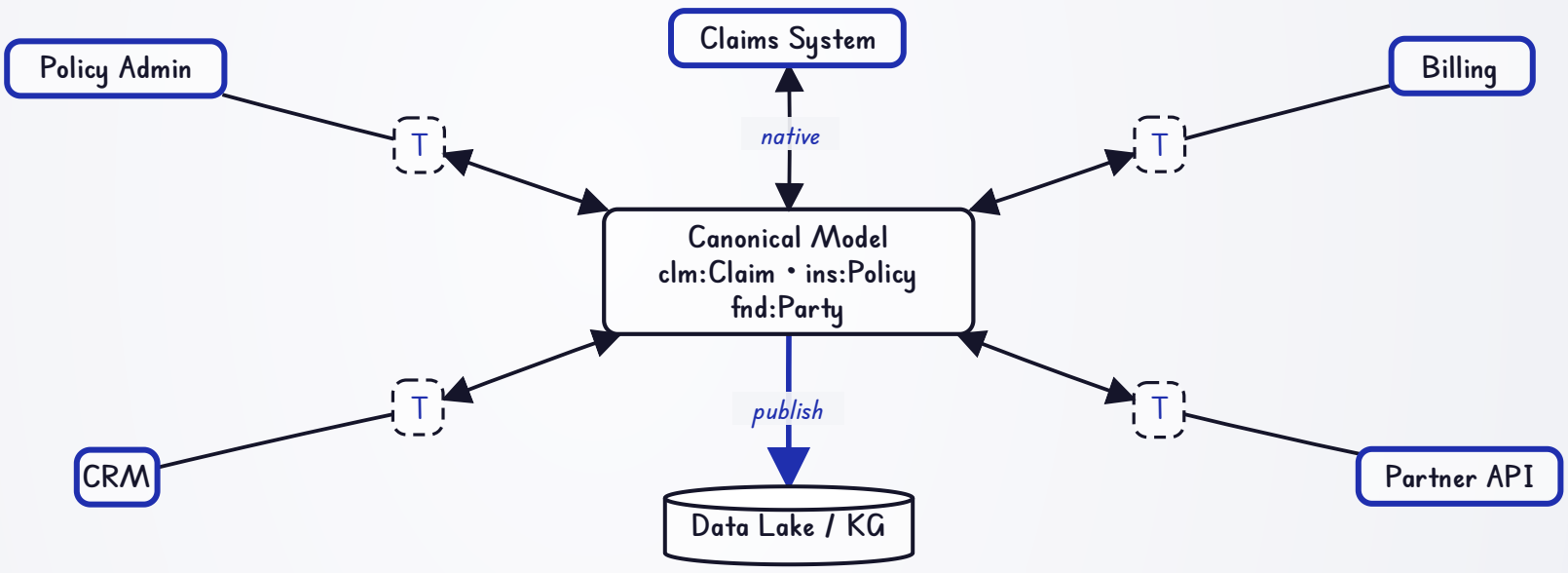
25 What is a Canonical Model?

Part C · The Canonical Model

25.1 The pattern (Hohpe & Woolf 2003)

Problem as the pattern states it: *"How can you minimize dependencies when integrating applications that use different data formats?"* **Solution**: design a Canonical Data Model that is *"independent from any specific application"* and require every application to produce and consume messages in that common format.

25.2 Picture: one hub language, a translator per system



T = Message Translator. A system that already speaks canonical (Claims, above) needs none.

25.3 In plain words

- A **shared, application-neutral vocabulary for data in motion**: the structure of the messages, events and API payloads that cross system boundaries.
- Each system keeps its own internal model; it only **translates at its edge** (T boxes above).
- It is a **contract**, not a database: nobody has to store data in canonical form.
- In this notebook the canonical model is **derived from the enterprise ontology** (fnd/ins/clm), so every field has a defined meaning.

25.4 Patterns that travel with it

EIP pattern	Role next to the canonical model
Message Translator	one adapter per app: app format ⇄ canonical
Normalizer	many incoming formats → one canonical format
Envelope Wrapper	header/metadata around the business body
Format Indicator	says which version/format a message is in

25.5 What it is NOT

- **Not** the union of every field of every system (that is a "god model", see p. 38).
- **Not** a replacement for the ontology: the ontology says what things *mean*; the canonical says how they are *shaped on the wire*.

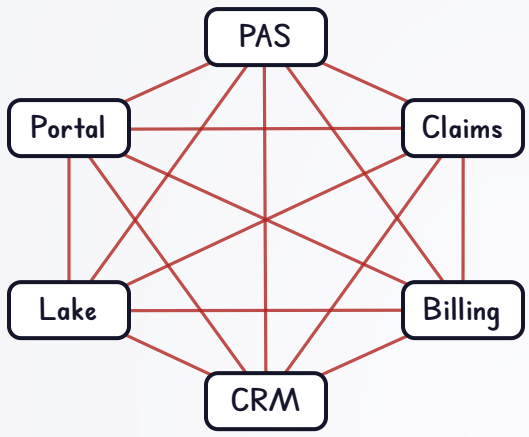
Key Takeaway: a canonical model trades one extra translation hop for far fewer pairwise dependencies; it pays off when many systems share the same business objects.

Sources: Hohpe & Woolf, *Enterprise Integration Patterns* (Addison-Wesley 2003) — Canonical Data Model pattern · enterpriseintegrationpatterns.com/patterns/messaging/CanonicalDataModel.html

㉖ Point-to-Point vs Canonical

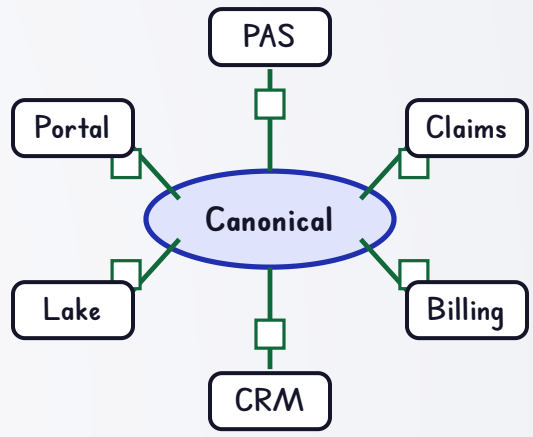
Part C · The Canonical Model

26.1 Point-to-point (n = 6)



15 links × 2 directions = 30 translators

26.2 Canonical hub (n = 6)



6 apps × (in + out) = 12 translators

26.3 The math

Point-to-point: every **ordered pair** (sender, receiver) needs its own translator:

$$T_{p2p} = n(n - 1)$$

Canonical: every app needs one translator **into** and one **out of** the canonical format:

$$T_{can} = 2n$$

Break-even: $n(n-1) = 2n \rightarrow \mathbf{n = 3}$. These match the pattern page: 2 apps → 2 vs 4; 3 apps → 6 vs 6; 6 apps → 30 vs 12.

Cost of change: adding one more app costs **2(n−1)** new translators point-to-point, but only **2** with a canonical model. This incremental cost is the argument that convinces budget holders.

26.4 Worked table

n apps	P2P n(n−1)	Canonical 2n	ratio (n−1)/2	add app #n+1: P2P	add app: canonical
2	2	4	0.5×	2	2
3	6	6	1×	4	2
4	12	8	1.5×	6	2
6	30	12	2.5×	10	2
10	90	20	4.5×	18	2
20	380	40	9.5×	38	2
50	2450	100	24.5×	98	2

26.5 Honest caveats (say these in the design review)

- Real estates are not fully connected: count the **actual** interfaces in your integration inventory, not n(n−1).
- A translator is not a unit of effort; a canonical translator is often **bigger** (it maps to a richer model).
- Canonical adds a hop: two translations, extra latency, and possible **semantic loss** if the canonical is too coarse.
- For 2-3 tightly coupled systems a direct, well-documented interface can be the simpler choice (the pattern page admits it looks complicated for only a few applications).

Key Takeaway: show stakeholders the incremental formula (2 vs 2(n−1)), not just the total; and use your real interface count.

Sources: Hohpe & Woolf, *Enterprise Integration Patterns* (Addison-Wesley 2003) — Canonical Data Model pattern · Numbers 2/4, 6/6, 30/12 are from the pattern page; other rows are computed from the formulas

Rajesh Kumar Gupta

27 Canonical vs Ontology

Part C · The Canonical Model

27.1 The layer cake

Enterprise Ontology (OWL + SKOS)
what things MEAN · open world · fnd / ins / clm

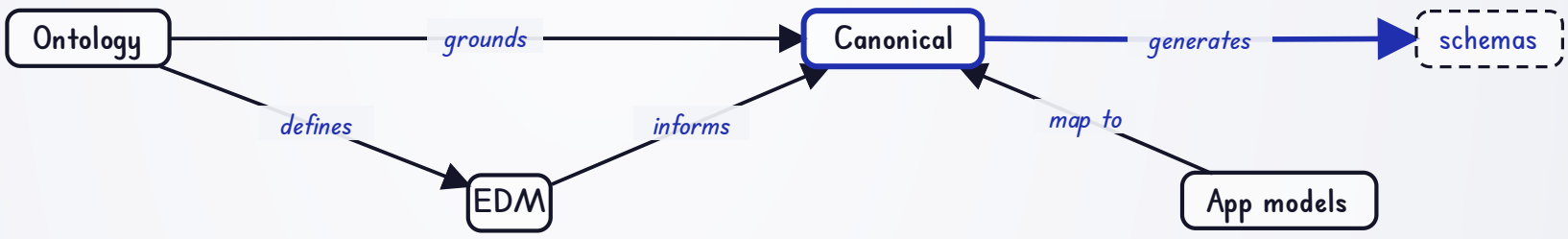
Enterprise Data Model (conceptual / logical)
entities, keys, relationships for the whole enterprise

Canonical Model (exchange contract)
shape of data in motion: messages, events, APIs

Application Logical Models
each system's own entities (bounded contexts)

Physical Schemas
JSON Schema · Avro · OpenAPI · SQL DDL · SHACL

27.2 How they relate



27.3 Side by side

	Enterprise Ontology	Enterprise Data Model	Canonical Model	App Logical Model
Question	What does it mean?	What do we keep?	What do we exchange?	How does this app work?
Form	OWL, RDFS, SKOS, SHACL	ER / UML (DMBOK levels)	JSON Schema, Avro, XSD, OpenAPI	tables, classes
Assumption	open world, no UNA	closed world	closed world, strict	closed world
Scope	enterprise + domains	enterprise	boundaries between systems	one system
Change rate	slow	slow	versioned, governed	fast
Example	clm:Claim ⊑ fnd:Event	CLAIM entity	ClaimOpened event v2	claims_hdr table

27.4 Decision rules for architects

1. **Meaning lives once** : definitions belong in the ontology; the canonical **references** them (slot_uri, @context) instead of re-defining them.
2. **EDM is optional, not skipped** : if you already have an EDM, treat it as a **view** of the ontology and keep both aligned; do not run three competing "truths".
3. **Canonical ≠ storage** : never force an app to persist the canonical shape; it is for the wire.
4. **Generate downwards** : physical schemas are generated from the canonical, never hand-edited (p. 33).

Key Takeaway: the ontology is the dictionary, the canonical is the grammar for one conversation pattern, the schemas are the printed forms.

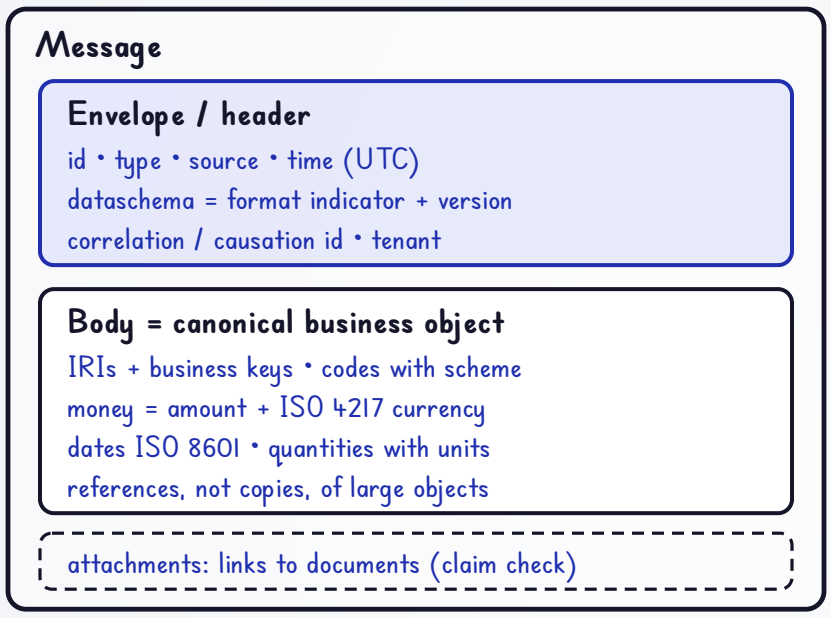
Sources: DAMA International, *DAMA-DMBOK* 2nd ed. (2017) · W3C OWL 2 Overview (2nd ed., 2012) · Hohpe & Woolf, *Enterprise Integration Patterns* (2003)

Rajesh Kumar Gupta

28 Canonical Message Design

Part C · The Canonical Model

28.1 Anatomy of a canonical message



28.2 Header standard: reuse one

Do not invent an envelope. **CloudEvents** (CNCF, spec v1.0.2; graduated Jan 2024) requires only *id*, *source*, *specversion*, *type* and adds optional *time*, *dataschema*, *subject*, *datacontenttype*.

Hohpe & Woolf name the pieces: **Envelope Wrapper** (header around body), **Format Indicator** (version tag), **Correlation Identifier** (request ↔ reply), **Claim Check** (store big payloads, send a reference).

28.3 Example: ClaimOpened v2 (CloudEvents structured JSON)

```
{ "specversion": "1.0",
  "id": "9f1c-0042",
  "type": "com.example.clm.ClaimOpened.v2",
  "source": "/claims/fnol",
  "time": "2026-03-14T09:30:00Z",
  "dataschema": "https://ontology.example.com/schema/clm/ClaimOpened/2.1",
  "datacontenttype": "application/json",
  "data": {
    "claim": { "@id": "https://data.example.com/id/claim/CLM-1001",
               "claimNumber": "CLM-1001",
               "claimStatus": { "code": "OPEN",
                 "scheme": "https://ontology.example.com/clm/ClaimStatusScheme" },
               "lossDate": "2026-03-12",
               "onPolicy": "https://data.example.com/id/policy/POL-77",
               "reserve": { "amount": "2500.00", "currency": "USD" } } } }
```

28.4 Field design rules

Concern	Canonical rule	Why
Identifier	enterprise IRI **and** source business key	joins + traceability (p. 35)
Money	decimal as string + ISO 4217 code	no float rounding, no implied currency
Time	ISO 8601 / RFC 3339 with offset; events in UTC	no local-time ambiguity
Codes	code + scheme IRI (SKOS)	codes mean nothing without their list
Absent	omit the field; define if null is allowed	open vs closed meaning
Version	major in type/schema URI; minor = additive	consumers can route & upgrade

Key Takeaway: design the body from ontology terms, the header from a public standard, and put the version in a place routers can read without parsing the body.

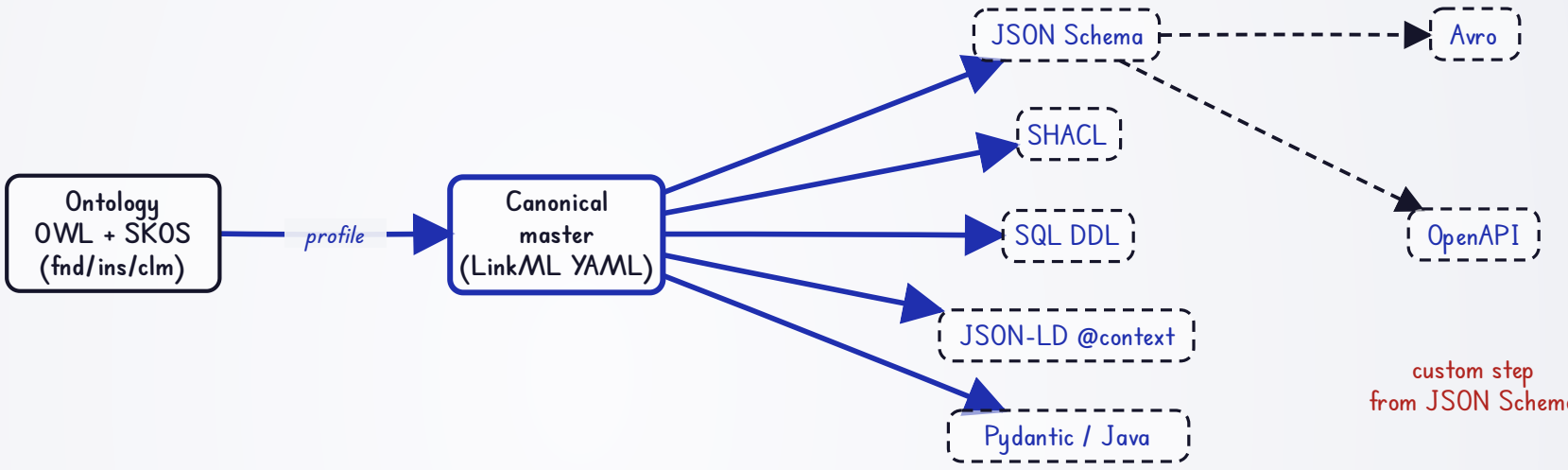
Sources: CNCF CloudEvents Specification v1.0.2 · Hohpe & Woolf, *Enterprise Integration Patterns* (2003) · ISO 8601-1:2019; IETF RFC 3339 (2002) · ISO 4217:2015 currency codes

Rajesh Kumar Gupta

㉙ Ontology → Canonical → Schemas

Part C · The Canonical Model

29.1 One source, many generated contracts



Solid blue arrows = generated by LinkML. Dashed arrows: the LinkML generator list has **no Avro generator**, so write a small converter (e.g. from the JSON Schema). For OpenAPI, the docs list an OpenAPI generator but it was not in the CLI build I tested; OpenAPI 3.1 can also embed the generated JSON Schema directly (pin its dialect, p. 67).

29.2 Canonical master (tested)

```
# clm_canonical.yaml  (LinkML schema = canonical master)
id: https://ontology.example.com/clm/canonical
name: clm_canonical
prefixes:
  linkml: https://w3id.org/linkml/
  clm: https://ontology.example.com/clm/
default_prefix: clm
imports: [linkml:types]
classes:
  Claim:
    class_uri: clm:Claim          # <- ontology class
    slots: [claimNumber, claimStatus, lossDate, onPolicy]
slots:
  claimNumber: {slot_uri: clm:claimNumber, identifier: true}
  claimStatus: {slot_uri: clm:hasClaimStatus, range:
ClaimStatusEnum, required: true}
  lossDate:    {slot_uri: clm:lossDate, range: date}
  onPolicy:    {slot_uri: clm:onPolicy, range: uriorcurie}
enums:
  ClaimStatusEnum:
    permissible_values:
      OPEN:   {meaning: clm:Open}     # <- SKOS concept
      CLOSED: {meaning: clm:Closed}
```

29.3 Generate (CI step)

```
# real LinkML CLI generators
gen-json-schema clm_canonical.yaml > clm.schema.json
gen-shacl       clm_canonical.yaml > clm.shacl.ttl
gen-sqltables   clm_canonical.yaml > clm.sql
gen-jsonld-context clm_canonical.yaml > clm.context.jsonld
gen-pydantic    clm_canonical.yaml > clm_model.py
gen-owl         clm_canonical.yaml > clm_check.owl.ttl
```

```
# excerpt of gen-sqltables output
CREATE TABLE "Claim" (
  "claimNumber" TEXT NOT NULL,
  "claimStatus" VARCHAR(6) NOT NULL,
  "lossDate" DATE, ...
```

29.4 Rules

- **class_uri / slot_uri / meaning** tie every canonical element to an ontology IRI: this is the traceability link.
- Generated files are **build artifacts**: never hand-edit; regenerate in CI on every change.
- Decide which is master: ontology-first (profile OWL into LinkML) or LinkML-first (gen-owl back). Pick one per module.

Key Takeaway: if a field cannot point to an ontology IRI, it is either a missing concept (raise it) or app-local data (keep it out of the canonical).

Sources: LinkML documentation, linkml.io (generators, accessed 2026) · JSON Schema 2020-12; OpenAPI Specification 3.1 · W3C JSON-LD 1.1 Recommendation (2020) · W3C SHACL Recommendation (2017)

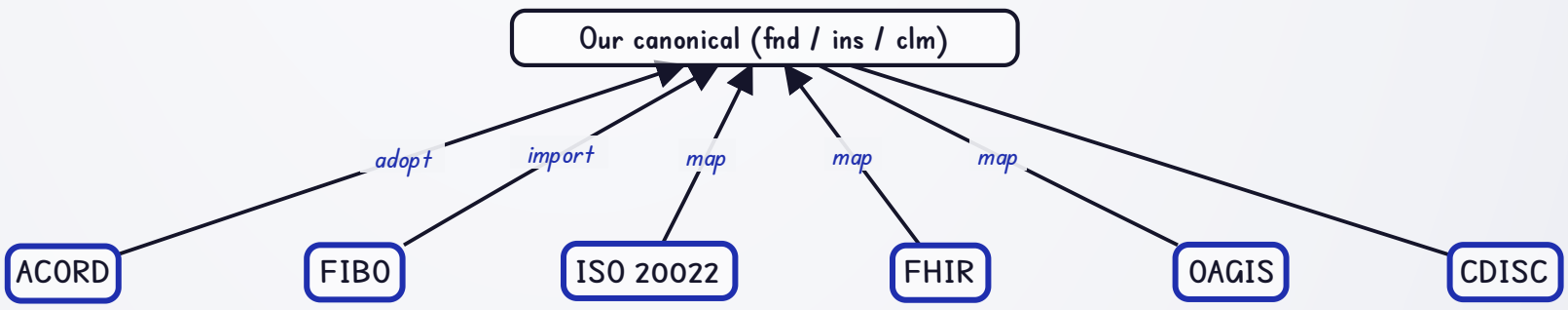
30 Industry Canonical Standards

Part C · The Canonical Model

30.1 The six you will meet (verified on official sites, Nov 2026)

Standard	Domain	Owner / steward	Form	Use it for
ACORD	insurance (P&C, L&A, reinsurance)	ACORD – non-profit, industry-owned standards body	XML (P&C, L&A, GRLC), AL3, JSON (GRLC), Next-Gen Digital Standards for APIs; Reference Architecture (Glossary, Information, Data, Capability models)	message shapes & insurance terms; our ins/clm alignment
FIBO	financial business concepts	EDM Council (open community since 2020); also standardized at OMG	OWL ontologies, quarterly releases (latest seen: 2026Q3)	reuse of party, contract, currency, legal-entity semantics
ISO 20022	payments, securities, trade, FX	ISO TC 68; SWIFT acts as Registration Authority	syntax-neutral business model + repository → XML / ASN.1 schemas	claim payments, premium collection messages
HL7 FHIR	healthcare	HL7 International	Resources in JSON, XML, RDF; R5 (5.0.0, Mar 2023) current published; R4 widely deployed; R6 in ballot	health / medical claims, injury data
OAGIS (now *connectSpec*)	ERP, supply chain, manufacturing	Open Applications Group, Inc. (OAGi)	Business Object Documents (verb + noun), 10.x line; tool Score → connectCenter	procurement, invoices, vendor data
CDISC	clinical research	CDISC, 501(c)(3) non-profit	SDTM, ADaM, CDASH, SEND, ODM, Define-XML, Controlled Terminology	life-science data exchange; FDA requires several

30.2 How they plug into an insurance canonical



30.3 Adopt, adapt or align? A decision rule

1. **Adopt** : when you exchange with partners who already speak it (e.g. ACORD with brokers, ISO 20022 with banks). Use it **as is** at that boundary.
2. **Adapt** : take the standard as the starting canonical and add a **governed extension** namespace for what it lacks.
3. **Align** : keep your own canonical but map each term (skos:exactMatch / closeMatch, owl:equivalentClass only when truly equal).

- Check the **licence**: some standards need membership to use the full specs.
- Pin the **version** you align to (FIBO quarter, FHIR release) and record it in owl:versionIRI / dcterms:source.
- Never copy a whole standard into the canonical; take only what your use cases need (MIREOT-style).
- Version facts change: **re-check status** on the official sites before each release.

Key Takeaway: standards are the cheapest canonical you will ever get: someone else already argued about the definitions. Map to them at the boundaries.

Sources: acord.org (About; Data Standards; Reference Architecture) · spec.edmcouncil.org/fibo; github.com/edmcouncil/fibo releases · iso20022.org; hl7.org/fhir (history) · oagi.org (OAGIS & Score); cdisc.org/standards

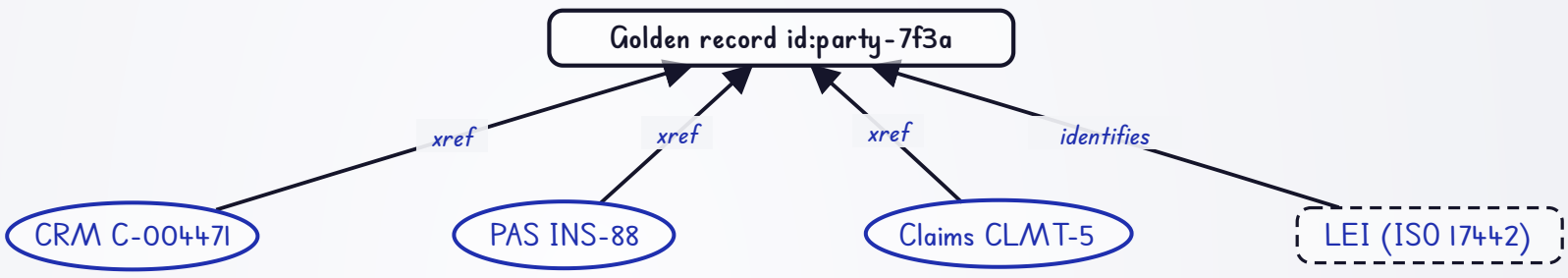
Rajesh Kumar Gupta

㉛ Identity, Keys & MDM

Part C · The Canonical Model

31.1 Three kinds of identifier

- **Enterprise IRI** (opaque, never re-used): *https://data.example.com/id/party-7f3a*. Identifies the *thing*.
- **Business / source keys**: policy number, claim number, CRM id. Identify a *record* in one system.
- **External identifiers**: LEI (ISO 17442) for legal entities, national IDs (handle as PII). Issued by an authority.



31.2 Master Data Management in one line

Ingest (source records) → Standardize (names, addresses) → Match (rules / ML) → Merge (survivorship) → Golden record (enterprise IRI) → Publish xref (to canonical)

MDM (DAMA-DMBOK2, ch. "Reference & Master Data") produces the **golden record** and the **cross-reference** that the canonical carries. The canonical never invents identity; it publishes what MDM decided.

31.3 Identifier policy (write it down)

1. Every canonical object carries the enterprise IRI **plus** the source key it came from.
2. IRIs are minted by one service (MDM / ID service), never by consumers.
3. Identifier values are data: model them (fnd:Identifier + issuing scheme) instead of 12 "xxxId" fields.
4. Merges and splits are events (PartyMerged) so consumers can re-point references.
5. Retired IRIs stay resolvable and point to the survivor.

31.4 owl:sameAs: handle with care

Halpin et al. (2010) showed sameAs is widely used in Linked Data for weaker relations than strict identity. In OWL it is symmetric, transitive and makes **all** statements about one IRI hold for the other, so one bad match merges two customers' claims.

- Use sameAs only for **confirmed** identity, ideally inside MDM.
- Otherwise: skos:closeMatch / exactMatch, or a reviewed crosswalk.

31.5 Turtle pattern (validated with rdflib)

```
@prefix fnd:  <https://ontology.example.com/fnd/> .
@prefix id:   <https://data.example.com/id/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
# one opaque enterprise IRI, many source keys (values illustrative)
id:party-7f3a  a fnd:Organization ;
   fnd:hasIdentifier [ a fnd:Identifier ;
      fnd:identifierValue "EXAMPLE0LEI000000099" ;
      fnd:issuedBy id:scheme-lei ] ,
    [ a fnd:Identifier ;
      fnd:identifierValue "C-004471" ;
      fnd:issuedBy id:scheme-crm ] .
# weaker link than owl:sameAs for an unconfirmed match
id:party-7f3a  skos:closeMatch  id:party-91bc .
```

Key Takeaway: identity is a service, not a field: one minting authority, explicit cross-references, and no casual owl:sameAs.

Sources: DAMA International, *DAMA-DMBOK* 2nd ed. (2017) · Halpin, Hayes, McCusker, McGuinness & Thompson, "When owl:sameAs Isn't the Same", ISWC 2010 (LNCS 6496) · ISO 17442 Legal Entity Identifier (GLEIF) · W3C SKOS Reference Recommendation (2009)

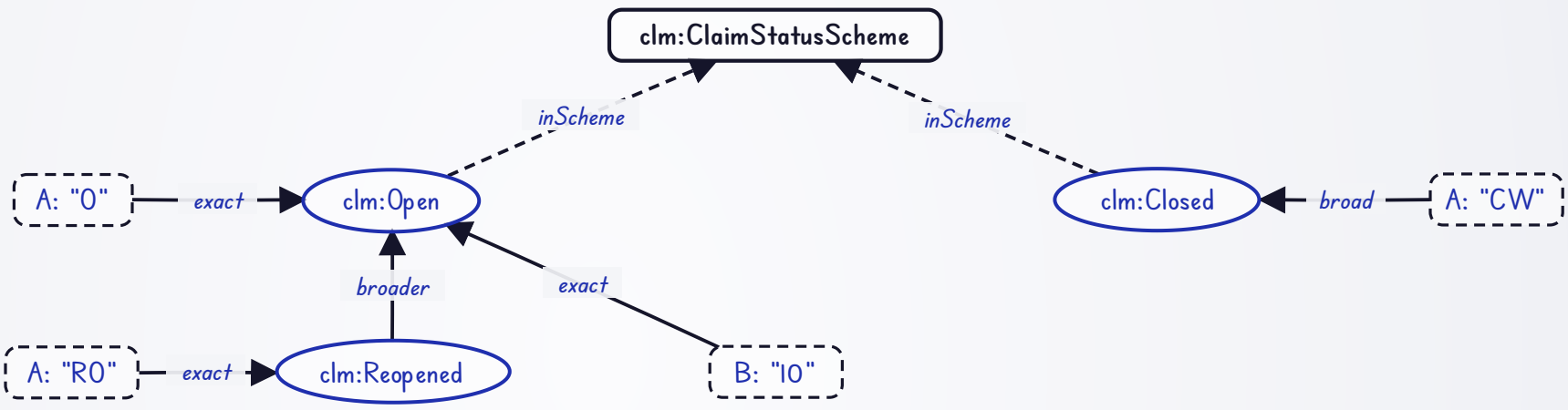
Rajesh Kumar Gupta

32 Reference Data & Code Lists

Part C · The Canonical Model

32.1 Code lists are the hidden half of the canonical

A canonical field such as *claimStatus* is useless if each system sends its own codes. Model every list as a **SKOS concept scheme**, publish it, and carry **code + scheme** in messages (p. 32).



32.2 Turtle (validated with rdflib)

```
@prefix clm:  <https://ontology.example.com/clm/> .
@prefix srcA:
<https://ontology.example.com/src/claimsA/status/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#>
.
clm:ClaimStatusScheme a skos:ConceptScheme ;
   skos:prefLabel "Claim Status"@en ;
   skos:hasTopConcept clm:Open, clm:Closed .
clm:Open a skos:Concept ; skos:inScheme
clm:ClaimStatusScheme ;
   skos:notation "OPEN" ; skos:prefLabel "Open"@en ;
   skos:definition "Claim accepted for handling; not yet
finally settled."@en .
clm:Reopened a skos:Concept ; skos:inScheme
clm:ClaimStatusScheme ;
   skos:broader clm:Open ; skos:notation "REOPENED" .
clm:Closed a skos:Concept ; skos:inScheme
clm:ClaimStatusScheme ;
   skos:notation "CLOSED" ; skos:prefLabel "Closed"@en .
# crosswalk: source system A codes
srcA:O  skos:exactMatch  clm:Open .
srcA:RO skos:exactMatch  clm:Reopened .
srcA:CW skos:broadMatch  clm:Closed .   # "closed
without payment"
```

32.3 Crosswalk table (steward-owned)

System A	System B	Canonical concept	SKOS relation	Note
0	10	clm:Open	exactMatch	
R0	–	clm:Reopened	exactMatch	B has no reopen state
CW	40	clm:Closed	broadMatch	source is narrower
C	41	clm:Closed	exactMatch	

32.4 Translate at the edge

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX srcA:
<https://ontology.example.com/src/claimsA/status/>
SELECT ?canon ?code WHERE {
  srcA:CW (skos:exactMatch|skos:broadMatch) ?canon .
  ?canon skos:notation ?code . }
```

The translator looks up the crosswalk; it does not hard-code if/else.

32.5 Rules of thumb

- **Reuse external lists** where they exist: ISO 3166 countries, ISO 4217 currencies; ACORD / FHIR code systems at those boundaries.
- Use the SKOS mapping properties honestly: exactMatch, closeMatch, broadMatch, narrowMatch, relatedMatch. A lossy mapping (broadMatch) must be visible to consumers.
- Each list has an **owner** (data steward), a version, and deprecation rules (owl:deprecated on retired concepts, never delete).
- Validate messages with SHACL *sh:in* (generated, p. 33) so unknown codes fail fast.

Key Takeaway: if you govern only the structure and not the code lists, you do not have a canonical model yet.

Sources: W3C SKOS Reference Recommendation (2009) · W3C SKOS Primer (Note, 2009) · W3C SHACL Recommendation (2017) · ISO 3166-1; ISO 4217:2015

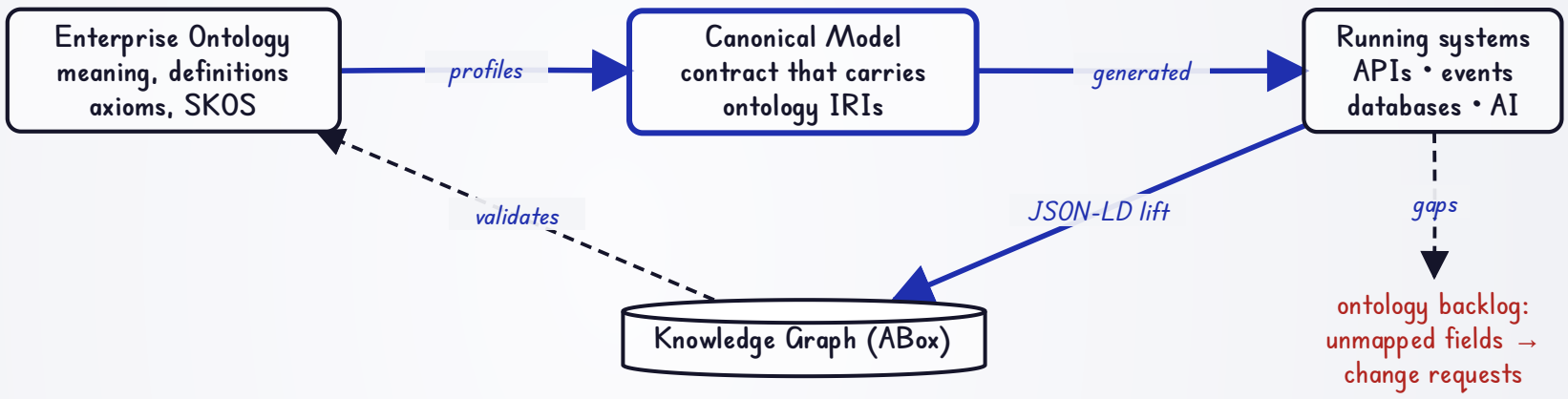
㉝ Making the Ontology Succeed

Part C · The Canonical Model

33.1 The gap every ontology program hits

An ontology alone lives in an editor and a triplestore. Payments, claims and billing systems exchange **JSON, Avro, XML and SQL rows**, not OWL. If meaning never reaches those formats, teams keep their own definitions and the ontology becomes shelf-ware. The canonical model is the delivery vehicle: it carries ontology meaning, as IRIs, into every message and table.

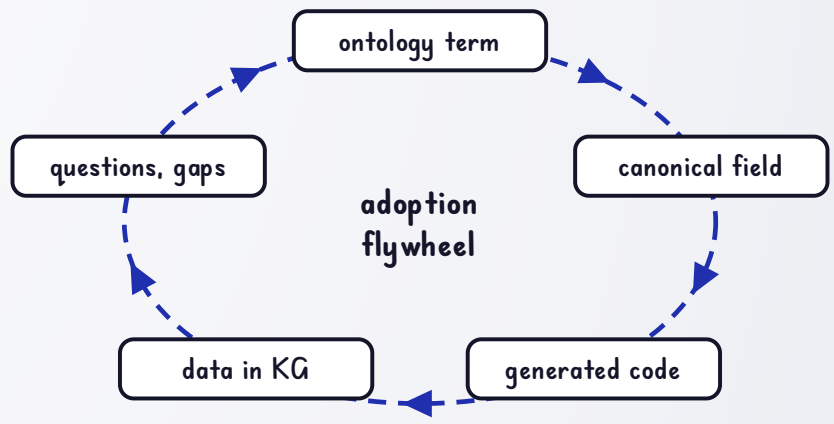
33.2 The bridge from meaning to running systems



33.3 Seven mechanisms that make it work

1. **Traceability** : every field has class_uri / slot_uri / @context to an ontology IRI.
2. **Enforcement** : generated schemas + SHACL checked at gateway and schema registry.
3. **Adoption** : generated types (Pydantic, Java) make the ontology the easy path.
4. **Free KG loading** : a JSON-LD @context turns canonical JSON into RDF: no extra mapping project.
5. **Feedback** : fields that cannot be mapped become ontology change requests.
6. **Impact analysis** : change clm:Claim → see every schema, topic, API using it.
7. **AI grounding** : LLM tools and GraphRAG see the same IRIs the APIs use.

33.4 The flywheel



33.5 Traceability matrix (generate it, never type it)

Ontology term	Canonical field	JSON path	DDL column
clm:Claim	Claim	$.data.claim	table Claim
clm:hasClaimStatus	claimStatus	$.data.claim.claimStatus.code	claimStatus
clm:lossDate	lossDate	$.data.claim.lossDate	lossDate

33.6 Monday plan

1. Pick **one** high-traffic event, e.g. ClaimOpened, and one consuming team.
2. Profile only the ontology terms it needs into the canonical master (p. 33); add slot_uri to every field.
3. Generate schema + @context; register the schema; turn validation on in "warn" mode, then "enforce".
4. Load events into the KG via the @context; run 3 competency questions to prove meaning survived. Publish the matrix, repeat; track share of fields with an IRI and open unmapped fields.

Key Takeaway: ontology gives meaning, canonical gives reach. Without the canonical the ontology is a document; with it, it is a running contract.

Sources: W3C JSON-LD 1.1 Recommendation (2020) · LinkML documentation, linkml.io (generators, accessed 2026) · Hohpe & Woolf, *Enterprise Integration Patterns* (2003) · W3C SHACL Recommendation (2017)

Rajesh Kumar Gupta

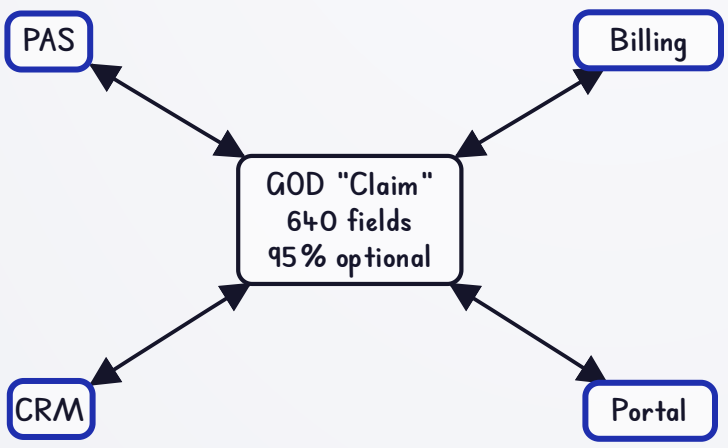
34 Canonical Anti-Patterns

Part C · The Canonical Model

34.1 The usual ways canonical programs fail

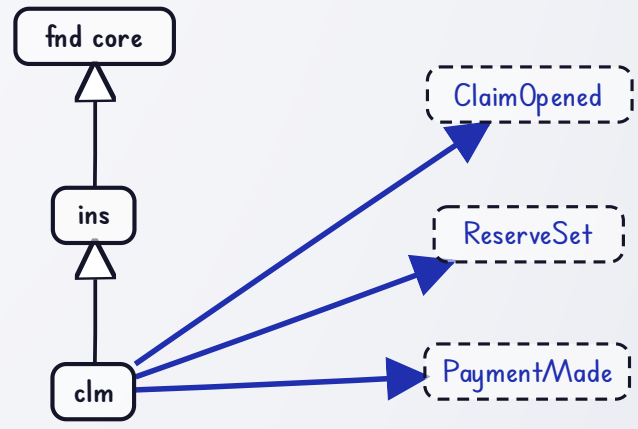
Anti-pattern	Symptom	Remedy
Big-bang canonical	long enterprise-wide modelling before the first message ships	grow event by event (p. 37); ship v1 for one flow
God model	one huge "Claim" with every field of every system, mostly optional	modules + profiles per message; only fields with an owner and a consumer
Lowest common denominator	only fields all systems share survive; rich data lost	optional extensions; separate events for specialised needs
Translation drift	adapters map the same field differently over time	generated validators, shared crosswalks, contract tests
Ungoverned extensions	"ext_1 … ext_40" or free-form maps carry real data	named extension namespace, review, promote to core
Canonical as database	apps forced to store canonical shape; tight coupling	canonical only at the boundary
Canonical inside a context	a team's internal code uses canonical types everywhere	keep internal models; translate at the edge (p. 39)
Versionless contract	breaking changes deployed in place	semver in type / schema URI; compatibility checks in CI
Meaning-free canonical	field names agreed, definitions not	slot_uri to ontology; definitions in one place

34.2 Bad: god model



(illustrative numbers) Every change touches everyone; nobody knows which fields are real.

34.3 Good: modular + profiles



Small message profiles built from shared modules; each has an owner and consumers.

34.4 Early-warning signs (check quarterly)

- Fields with no slot_uri / no definition are growing.
- Adapters contain hand-written mapping tables instead of the shared crosswalk.
- Consumers ask for a "raw" feed next to the canonical one, a sign the canonical lost information.

Key Takeaway: most failures are scope failures: too big, too early, or too generic. Keep it small, owned and versioned.

Sources: Hohpe & Woolf, *Enterprise Integration Patterns* (2003) · Evans, *Domain-Driven Design* (Addison-Wesley 2003) · Fowler, "BoundedContext", martinfowler.com bliki (2014)

Rajesh Kumar Gupta

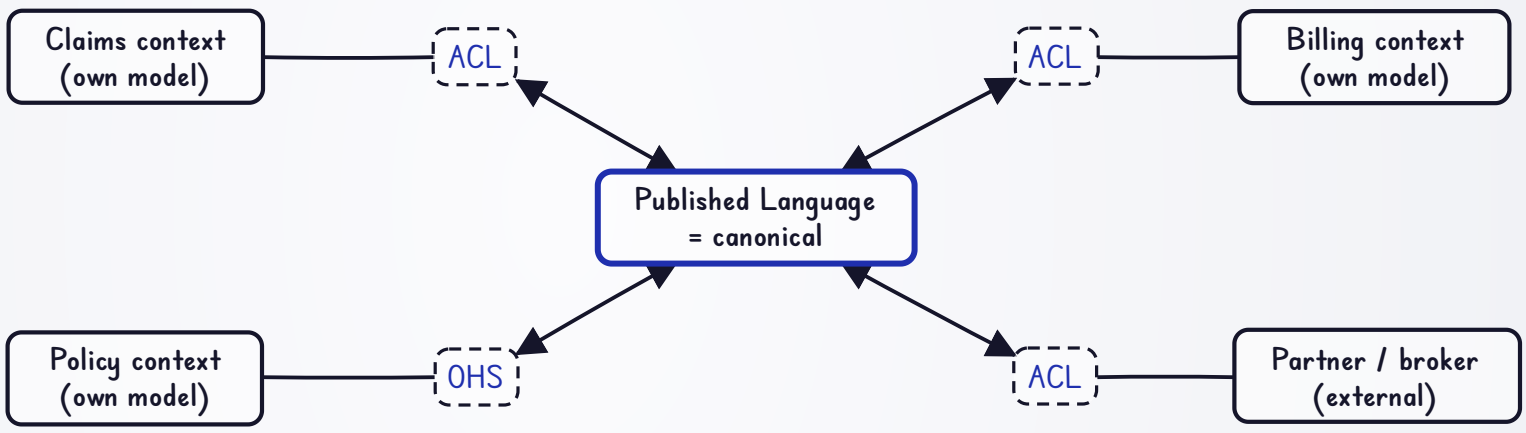
㉟ Bounded Contexts vs Canonical

Part C · The Canonical Model

35.1 The tension

DDD says one model for a large system is a trap: Evans writes that *"total unification of the domain model for a large system will not be feasible or cost-effective"* (quoted in Fowler 2014). A canonical model sounds like exactly that. It is not, **if you place it correctly**: at the boundaries between contexts, never inside them.

35.2 Context map with the canonical as Published Language

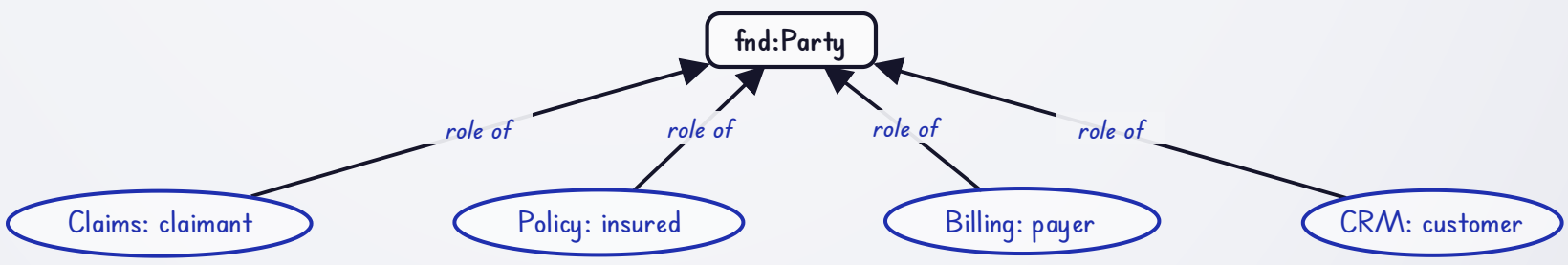


ACL = Anti-Corruption Layer, OHS = Open Host Service. Each context keeps its own language internally.

35.3 Mapping Evans's patterns to canonical design

Context-map pattern (Evans)	Meaning	Canonical role
Published Language	documented shared language for translation between contexts	= the canonical model
Open Host Service	a context offers a protocol/API for many consumers	expose it in canonical form
Anti-Corruption Layer	translation layer protecting a context's model	the Message Translator
Shared Kernel	small model subset shared by two teams	fnd core types (Money, Identifier)
Conformist	downstream adopts upstream model as is	e.g. ACORD/ISO 20022 at the edge
Separate Ways	no integration at all	no canonical needed

35.4 Same word, different contexts: the ontology reconciles



"Customer" means claimant, insured or payer depending on context. The **ontology** models these as roles of one fnd:Party; the canonical carries the role explicitly; contexts keep their local names.

Key Takeaway: bounded contexts own their internal models; the canonical is their Published Language; the ontology explains how the contexts' words relate.

Sources: Evans, *Domain-Driven Design* (Addison-Wesley 2003) · Fowler, "BoundedContext", martinfowler.com bliki (2014) · Hohpe & Woolf, *Enterprise Integration Patterns* (2003)

36 Part C Recap & Checklist

Part C · The Canonical Model

36.1 Part C on one page

Meaning: Enterprise Ontology
OWL classes + SKOS code lists (p. 31, 36)

Identity: MDM + ID service
enterprise IRIs, cross-references (p. 35)

Contract: Canonical master (LinkML)
profiles per message, slot_uri everywhere (p. 33)

Generated: schemas + @context + SHACL
JSON Schema, DDL, Pydantic; Avro/OpenAPI via converter

Boundary: translators / ACLs
one per system, contract-tested (p. 29, 39)

36.2 Should this be canonical?

Many systems share the object?
≥ 3 (p. 30)

↓

Partner standard exists?
adopt / adapt / align

↓

Inside one context?
then no canonical

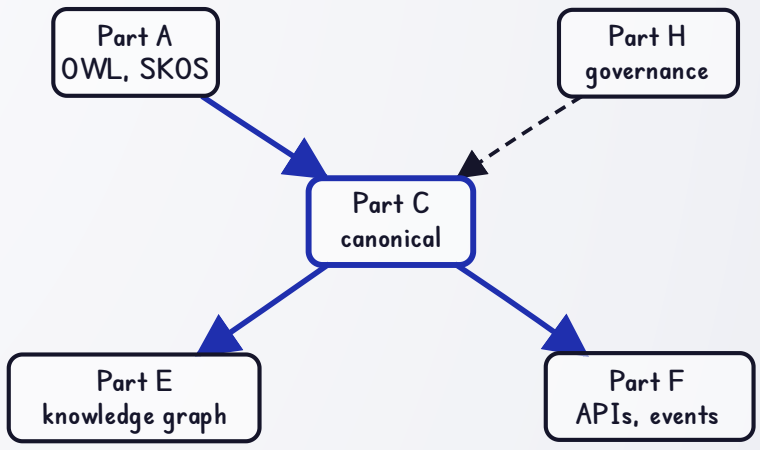
↓

Canonical profile
start with 1 event

36.3 Memory hooks

Page	One-line memory hook
29	one app-neutral format, a translator per app
30	n(n−1) vs 2n; break-even at 3
31	ontology = meaning, canonical = wire shape
32	standard envelope, ontology body
33	one master, generated schemas
34	adopt / adapt / align standards
35	identity is a service
36	code + scheme, SKOS crosswalks
37	canonical gives the ontology reach
38	small, owned, versioned
39	canonical = Published Language

36.5 Where Part C connects



36.4 Canonical design checklist (use in every design review)

- ☐ Scope: named events/APIs, owners and consumers listed
- ☐ Every field has slot_uri to an ontology IRI and a definition
- ☐ Envelope = CloudEvents (or other public standard); version in type / schema URI
- ☐ Money = decimal string + ISO 4217; time ISO 8601 with offset
- ☐ Identifiers: enterprise IRI + source key; no casual owl:sameAs
- ☐ Code lists = SKOS schemes; crosswalks owned by stewards
- ☐ Industry standard checked first (ACORD, FIBO, ISO 20022, FHIR ...)
- ☐ Schemas generated in CI; none hand-edited
- ☐ Validation at gateway / schema registry (warn → enforce)
- ☐ Compatibility check on every change; deprecation window
- ☐ Traceability matrix published; unmapped fields go to backlog
- ☐ Canonical used at boundaries only, never as storage

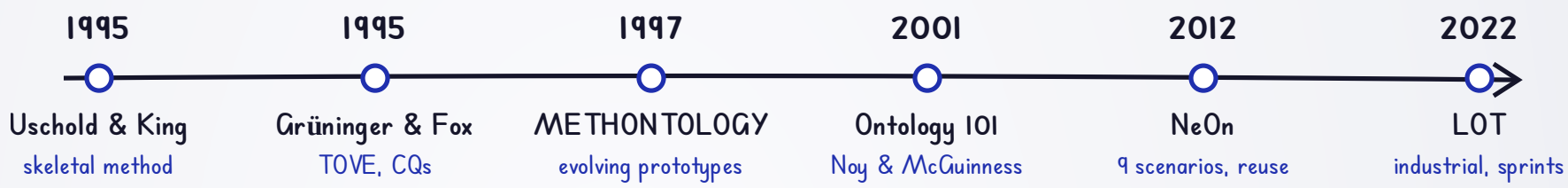
Key Takeaway: a good canonical model is small, generated, versioned and points every field at the ontology. That is how meaning reaches production.

Sources: Hohpe & Woolf, *Enterprise Integration Patterns* (2003) · Evans, *Domain-Driven Design* (Addison-Wesley 2003) · LinkML documentation, linkml.io (generators, accessed 2026) · CNCF CloudEvents v1.0.2

37 Ontology Methodologies

Part D · Implementing Enterprise Ontology in Industry

37.1 Six methods on one line

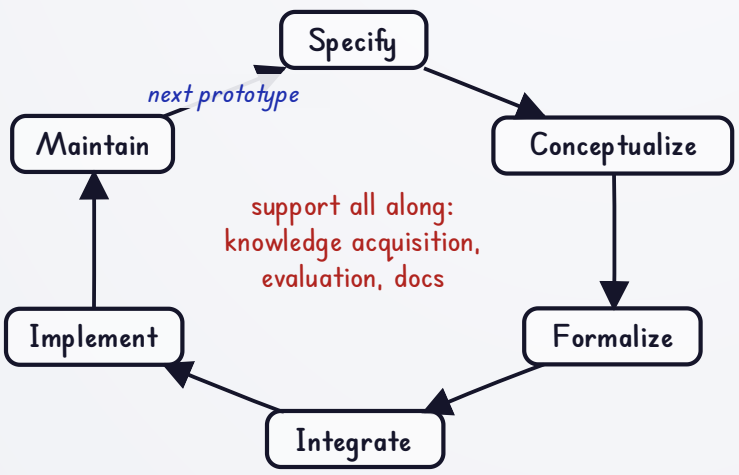


37.2 What each method contributes

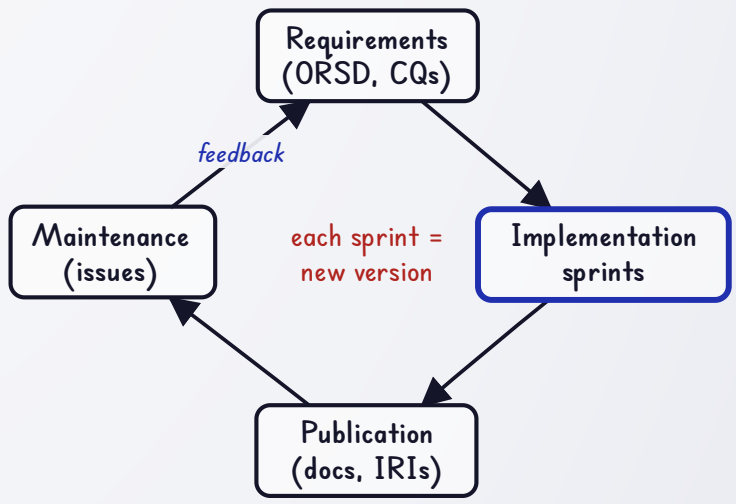
Method	Core phases / idea	Use it at work for
Uschold & King 1995	identify purpose → build (capture, code, integrate) → evaluate → document; *middle-out* term discovery	the minimum skeleton; start from "basic" terms such as Claim, then go up/down
Grüninger & Fox 1995	motivating scenarios → informal CQs → terminology → formal CQs → axioms	requirements and acceptance tests (next page)
METHONTOLOGY 1997	states: specification, conceptualization, formalization, integration, implementation, maintenance; life cycle of evolving prototypes	explicit intermediate conceptual models (glossary, tables) business can review
Ontology 101 2001	7 steps: scope, reuse, enumerate terms, classes & hierarchy, properties (slots), facets, instances	onboarding new modellers; a per-module checklist
NeOn 2012	9 scenarios (build from spec, reuse/re-engineer ontological & non-ontological resources, merge, ODPs, restructure, localize); ORSD	networks of modules and heavy reuse (FIBO, code lists, glossaries)
LOT 2022	requirements → implementation (conceptualize, reuse, encode, evaluate) → publication → maintenance, iterated in sprints	industrial delivery cadence: backlog, versions, issue tracker

37.3 Two life-cycle shapes

METHONTOLOGY: you may return to any earlier state when a definition is missing or wrong.



LOT: four activities looped in short iterations, each releasing a version.



37.4 The blend I would use on Monday

1. **Charter** : purpose, scope, users, ORSD-style requirement doc (NeOn)
2. **CQs** : written with the business; they become tests (G&F)
3. **Sprint** : per module: Ontology 101 steps as the checklist; reuse scenario chosen explicitly (NeOn)
4. **Publish** : versioned IRIs, docs, change log; issues drive the next sprint (LOT)

Key Takeaway: no method is "the" standard. Pick one life cycle (LOT-style sprints suit enterprises), and borrow requirement and reuse tools from the others.

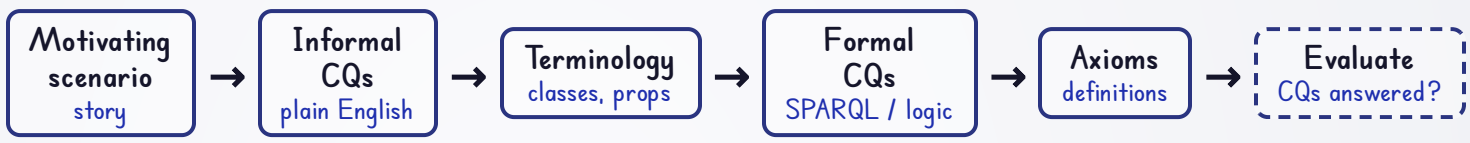
Sources: Uschold & King, "Towards a Methodology for Building Ontologies", IJCAI-95 Workshop (1995) · Grüninger & Fox, "Methodology for the Design and Evaluation of Ontologies", IJCAI-95 Workshop (1995) · Fernández, Gómez-Pérez & Juristo, "METHONTOLOGY", AAAI Tech. Report SS-97-06 (1997) · Noy & McGuinness, "Ontology Development 101", Stanford KSL-01-05 (2001) · Suárez-Figueroa, Gómez-Pérez & Fernández-López, "The NeOn Methodology", in Ontology Engineering in a Networked World, Springer (2012) · Poveda-Villalón, Fernández-Izquierdo, Fernández-López & García-Castro, "LOT", Eng. Appl. of AI 111 (2022)

38 Competency Questions

Part D · Implementing Enterprise Ontology in Industry

38.1 Where CQs come from (Grüninger & Fox)

Competency questions are the questions the ontology must be able to answer. They fix the **scope** (what is in and out) and later act as **acceptance tests**.



38.2 Writing good CQs

- Ask the **business** (claims handler, actuary), not the modeller; record who asked.
- One question, one need; use domain words ("reserve", not "amount_2").
- Prefer questions with a checkable answer (an entity, a number, yes/no).
- Include **negative** CQs ("must never…"): they become SHACL or ASK tests.
- Tag priority (must / should / could) and the source system that holds the data.

38.3 Requirement document (NeOn ORSD)

- purpose, scope, implementation language
- intended end users and intended uses
- requirements: non-functional + functional (CQs)
- a pre-glossary of terms pulled from the CQs (input to reuse search)

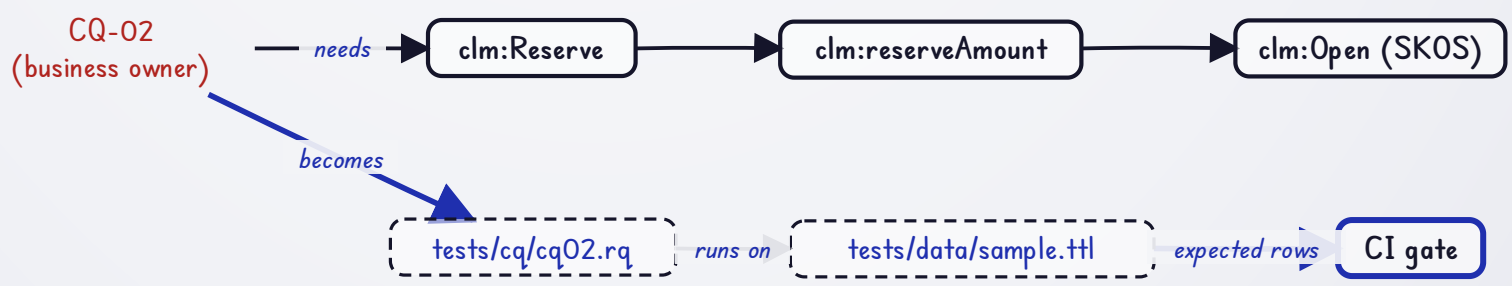
38.4 Claims CQ set (illustrative)

ID	Competency question (informal)	Terms it forces	Kind
CQ-01	Under which policy is claim C made?	Claim, Policy, onPolicy	lookup
CQ-02	What is the total reserve on each open claim?	Reserve, reserveAmount, ClaimStatus	aggregate
CQ-03	Which claims have a loss date outside the policy period?	LossEvent, lossDate, policy period	rule / quality
CQ-04	Who is the claimant on claim C, and are they also the insured?	Party, hasClaimant, role pattern	role
CQ-05	Which statuses count as "open"?	skos:broader in ClaimStatusScheme	code list
CQ-06	A claim must never have two policies.	cardinality → SHACL, not OWL	negative CQ

38.5 CQ → SPARQL → test

```
# CQ-02  "What is the total reserve on each claim that is still open?"
PREFIX clm:  <https://ontology.example.com/clm/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?claim (SUM(?amt) AS ?totalReserve) WHERE {
  ?claim a clm:Claim ;
         clm:hasClaimStatus ?st ;
         clm:hasClaimLine/clm:hasReserve/clm:reserveAmount ?amt .
  ?st skos:broader* clm:Open .          # Open or any narrower status
} GROUP BY ?claim
```

Real run (rdflib 7.x on the sample data of page 51): one row → **CLM-1001 totalReserve = 12500.00**. CLM-1002 is excluded because its status is not in the scheme.



Key Takeaway: a CQ with no owner, no data source and no test is a wish. Keep CQs in the repo next to the ontology, one .rq file each.

Sources: Grüninger & Fox, "Methodology for the Design and Evaluation of Ontologies", IJCAI-95 Workshop (1995) · Suárez-Figueroa, Gómez-Pérez & Fernández-López, "The NeOn Methodology", in Ontology Engineering in a Networked World, Springer (2012) · W3C SPARQL 1.1 Query Language (2013)

39 Scoping & Modular Architecture

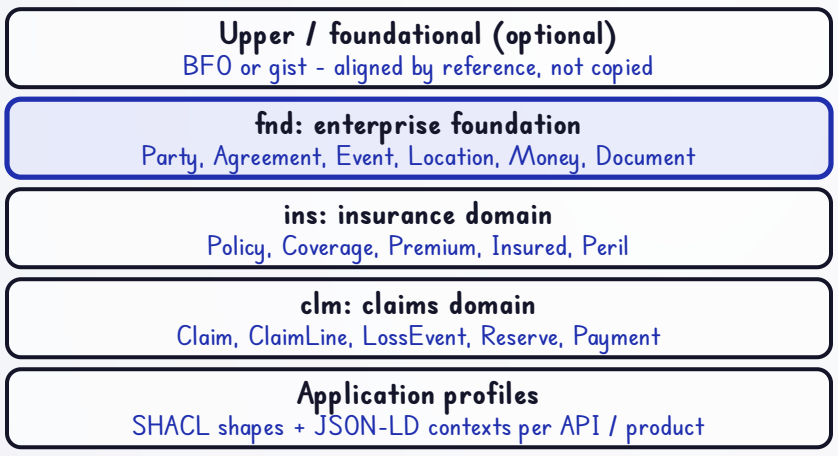
Part D · Implementing Enterprise Ontology in Industry

39.1 Scope before structure

Scope is decided by the CQs: a term enters only if a prioritised CQ needs it. Write the out-of-scope list down; it prevents the "model the whole company" trap.

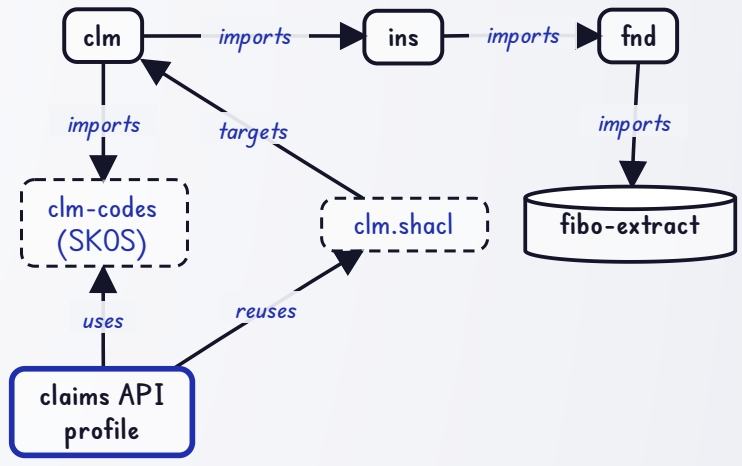
Scope decision (claims v1)	Items (illustrative)
In - answers a must-have CQ	Claim, ClaimLine, LossEvent, Reserve, Payment, claimant role, ClaimStatus
Deferred - later sprint, logged	subrogation, litigation, salvage, fraud indicators
Out - owned elsewhere	pricing/rating models (actuarial), GL postings (finance), UI fields

39.2 Layered modules



Lower layers may import upper ones, **never the reverse.** fnd stays small and stable; domains change faster.

39.3 Import graph (acyclic)



Shapes and code lists live in their own files so APIs can reuse them without pulling every axiom.

39.4 Module rules that survive real projects

- **One namespace per module**, one owner (steward) per module.
- A term is defined in exactly one module; others reference its IRI.
- Imports form a DAG; a cycle means a term is in the wrong layer.
- Keep each module **reasoner-friendly**: check the OWL 2 profile (EL/QL/RL) you need.
- Shapes (closed-world rules) separate from the OWL (open-world meaning).
- Use a catalog file so builds resolve imports locally and offline.

39.5 Repository layout

```
ontology/
  fnd/  fnd.ttl          # Party, Agreement, Event, Location, Money, Document
  ins/  ins.ttl          # Policy, Coverage, Premium, Insured, Peril
  clm/  clm.ttl          # Claim, ClaimLine, LossEvent, Reserve, Payment
        clm-codes.ttl    # SKOS: ClaimStatusScheme, ...
        clm.shacl.ttl    # shapes = data contract (separate file!)
  imports/ fibo-extract.ttl   # ROBOT extract of the FIBO terms we reuse
  mappings/ claims-db.r2rml.ttl
tests/ cq/*.rq   data/sample.ttl
catalog-v001.xml         # maps import IRIs to local files
```

Key Takeaway: modularity is an ownership decision first and a technical one second: draw the module boundary where the stewardship boundary is.

Sources: Suárez-Figueroa, Gómez-Pérez & Fernández-López, "The NeOn Methodology", in Ontology Engineering in a Networked World, Springer (2012) · W3C OWL 2 Structural Specification & Primer, 2nd ed. (2012) · Guarino, "Formal Ontology and Information Systems", FOIS (1998)

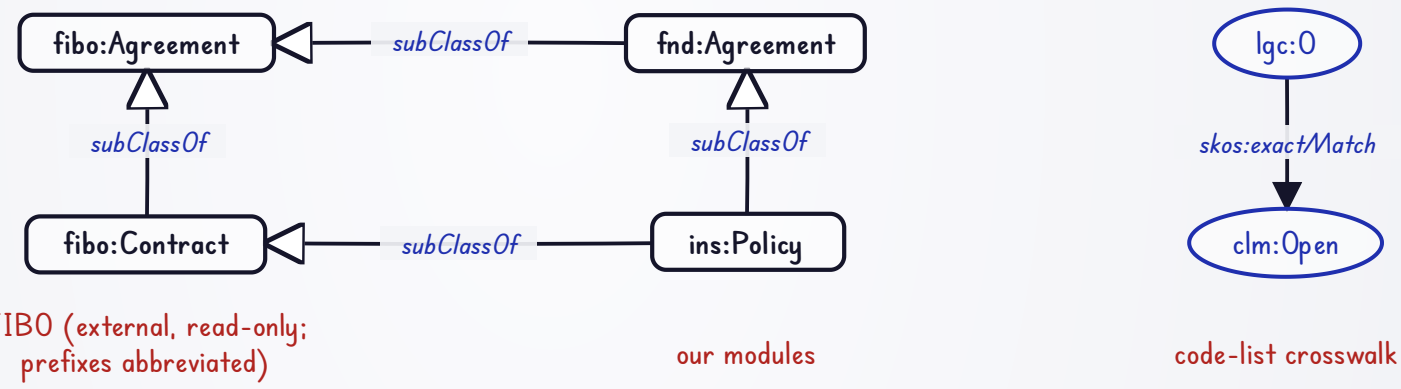
40 Reuse & Alignment

Part D · Implementing Enterprise Ontology in Industry

40.1 The reuse ladder (try top-down)

Reuse option	How	Choose when	Watch out
1 Import whole	*owl:imports* the published IRI	small, stable, fully relevant	pulls big closures; reasoner cost
2 Extract module	ROBOT *extract* BOT/STAR/TOP (syntactic locality)	large source, you need a logically safe subset	re-extract on each source release
3 MIREOT	copy minimal info per term: its IRI, source ontology, target superclass	a handful of terms, hierarchy only	axioms of the source are not carried
4 Align	mapping axioms in a bridge file	you keep your own terms	equivalence can over-commit
5 Build new	own term, document why	nothing fits after a search	record the search in the ADR

40.2 Alignment picture: bridge axioms, not copies



FIBO (external, read-only; prefixes abbreviated)

our modules

code-list crosswalk

40.3 Bridge file (validated Turtle)

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix fnd:  <https://ontology.example.com/fnd/> .
@prefix ins:  <https://ontology.example.com/ins/> .
@prefix clm:  <https://ontology.example.com/clm/> .
@prefix fibo-fnd-agr-agr: <https://spec.edmcouncil.org/fibo/ontology/FND/Agreements/Agreements/> .
@prefix fibo-fnd-agr-ctr: <https://spec.edmcouncil.org/fibo/ontology/FND/Agreements/Contracts/> .
@prefix lgc:  <https://codes.example.com/legacy-cms/status/> .

fnd:Agreement rdfs:subClassOf fibo-fnd-agr-agr:Agreement .  # weaker than equivalence
ins:Policy    rdfs:subClassOf fibo-fnd-agr-ctr:Contract .
lgc:0  skos:exactMatch clm:Open .       # legacy code -> canonical concept
lgc:CW skos:broadMatch clm:Closed .     # "closed without payment"
```

40.4 Choosing the mapping construct

- **rdfs:subClassOf** - safest class alignment; adds meaning upward only.
- **owl:equivalentClass** - only if definitions truly coincide; everything inferred flows both ways.
- **skos:exactMatch / closeMatch / broadMatch** - for code lists and glossaries; no OWL entailment on classes.
- **owl:sameAs** - individuals only, and rarely: it merges all statements (see page 35).

40.5 MIREOT / extract with ROBOT

```
robot extract --method BOT --input fibo-fnd.ttl \
  --term-file imports/fibo-terms.txt --output imports/fibo-extract.ttl
# or keep only hierarchy (MIREOT style):
robot extract --method MIREOT --input fibo-fnd.ttl \
  --lower-term
https://spec.edmcouncil.org/fibo/ontology/FND/Agreements/Contracts/Contract
--output imports/fibo-mireot.ttl
```

40.6 FIBO as a reuse source - facts to check before you plan

- Hosted by the **EDM Council**, open community process, standardised through the **OMG**; MIT licence; quarterly releases.
- Domains: FND, BE, FBC, SEC, DER, IND, LOAN, MD, CAE, BP (Foundations, Business Entities, Financial Business & Commerce, ...).
- Each ontology carries a maturity annotation (e.g. *Release*); reuse Release content first. FIBO is financial-services scoped - insurance-specific terms will mostly be yours.

Key Takeaway: reuse by reference (IRIs + extracts + bridge axioms) keeps you upgradeable. Copy-paste reuse forks the standard the day you do it.

Sources: Courtot et al., "MIREOT", Applied Ontology 6(1) (2011) · EDM Council, FIBO (spec.edmcouncil.org; GitHub edmcouncil/fibo, MIT licence) · ROBOT docs, robot.obolibrary.org (v1.9.x) (extract) · W3C SKOS Reference, Recommendation (2009)

㊶ Naming, IRI & Namespaces

Part D · Implementing Enterprise Ontology in Industry

41.1 Anatomy of a term IRI



41.2 Naming conventions (house style)

Element	Convention	Example
Class	UpperCamelCase, singular noun	clm:ClaimLine, ins:Coverage
Object property	lowerCamelCase verb phrase	clm:onPolicy, clm:hasClaimLine
Datatype property	lowerCamelCase noun	clm:claimNumber, clm:lossDate
SKOS scheme / concept	Scheme suffix; concept = UpperCamel	clm:ClaimStatusScheme, clm:Reopened
Shapes	own namespace, Shape suffix	clmsh:ClaimShape
Label	lower case natural language, @en	"claim line"@en
Data (ABox)	separate host / namespace from the TBox	https://data.example.com/CLM-1001

41.3 Readable or opaque?

	Readable IRI (clm:Claim)	Opaque IRI (clm:C000123)
Pros	self-explaining in code, SPARQL and JSON-LD	renaming a label never changes the IRI; language-neutral
Cons	a rename tempts an IRI change (breaking)	unreadable without labels; tooling must show rdfs:label
Used by	many enterprise and W3C vocabularies	OBO Foundry ontologies (prefix_digits)

Decision rule: **readable** for an enterprise vocabulary consumed by API teams; **opaque** when terms are numerous, multilingual or frequently relabelled. Either way: **an IRI is never reused for a different meaning.**

41.4 Cool-URI rules (W3C Note)

- **Simple, stable, manageable**: no .php/.owl, no tool names, no dates you will regret.
- Make IRIs dereferenceable: **hash** (#Claim) or **slash + 303** redirect to a document.
- Slash IRIs suit large modules (one resource per term).
- Keep data IRIs (ABox) on a different path or host than vocabulary IRIs.

41.5 Version IRIs

```
# prefixes omitted
<https://ontology.example.com/clm/> a owl:Ontology ;
   owl:versionIRI  <https://ontology.example.com/clm/1.2.0/> ;
   owl:priorVersion <https://ontology.example.com/clm/1.1.0/> ;
   owl:versionInfo "1.2.0" .
# term IRIs stay version-free:  https://ontology.example.com/clm/Claim
```

41.6 Policy checklist to publish internally

1. Base IRI and module paths fixed by the ontology board; registered in DNS/web server.
2. Term IRIs are permanent; changes = deprecate (owl:deprecated true) + replacement link.
3. Version IRI per release (semver); the ontology IRI always resolves to the latest.
4. Prefix registry (fnd, ins, clm, clmsh ...) shared with API and data teams.

Key Takeaway: the IRI is a contract. Spend an afternoon on this policy before the first term is minted; renaming later breaks every mapping, API and query.

Sources: W3C, "Cool URIs for the Semantic Web", Interest Group Note (2008) · OBO Foundry Principles (obofoundry.org) (identifier space, naming, versioning) · W3C OWL 2 Structural Specification & Primer, 2nd ed. (2012)

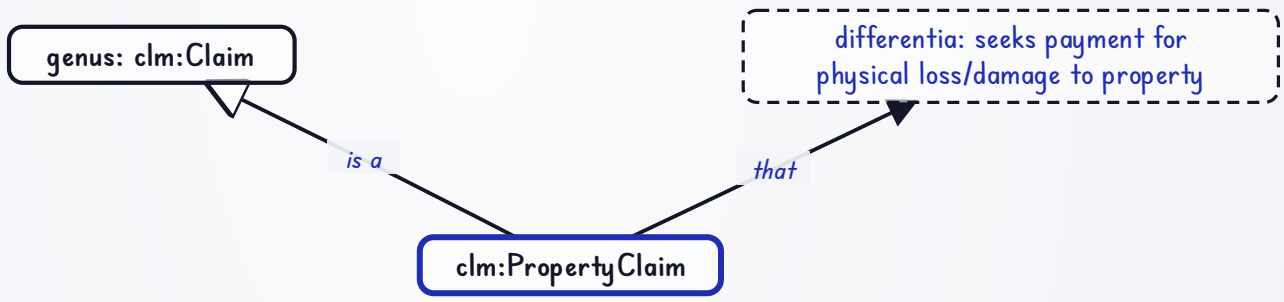
42 Annotations & Definitions

Part D · Implementing Enterprise Ontology in Industry

42.1 Minimum annotation profile per term

Annotation	Rule	Why
rdfs:label @en	**required**, exactly one per language	tools, UIs, ROBOT "missing label" = ERROR
skos:definition	**required**, exactly one, genus-differentia	shared meaning (see 42.5 for automated check)
skos:example / scopeNote	recommended for tricky terms	clarifies boundaries without bloating the definition
dcterms:source	recommended: glossary/regulation it came from	traceability for audit
rdfs:isDefinedBy	required: owning module IRI	who owns the term
owl:deprecated	true when retired (+ replacement link)	safe evolution, never delete

42.2 Aristotelian (genus-differentia) form



"A property claim is a claim that ..."

42.3 Spotting weak definitions

Guidelines from Seppälä, Ruttenberg & Smith (2017): exactly one genus (the parent class), necessary and ideally sufficient conditions, avoid circularity, negatives, lists, plurals and subjective words, and keep the text consistent with the logical axioms.

Weak definition	Problem (guideline)	Better
"Claims - see claim handling."	circular; plural genus	"A request to an insurer for payment ..."
"A claim that is not liability."	negative term	state what it *is* (property damage)
"Fire, flood or theft claims."	definition by extension / list	genus + differentia; list goes to skos:example
"Entry in CLM_TBL."	use/mention confusion (defines the record, not the thing)	define the real-world claim; map the table separately

42.4 A fully annotated class

```
clm:PropertyClaim a owl:Class ;                # prefixes omitted
   rdfs:subClassOf clm:Claim ;
   rdfs:label "property claim"@en ;
   skos:definition """A claim that seeks payment for physical loss of
      or damage to insured property."""@en ;
   skos:example "Kitchen fire damage under a homeowners policy."@en ;
   skos:scopeNote "Bodily injury to third parties is a liability claim."@en ;
   dcterms:source <https://glossary.example.com/claims#property-claim> ;
   rdfs:isDefinedBy <https://ontology.example.com/clm/> ;
   skos:editorialNote "Agreed at claims board 2026-05 (ADR-014)."@en .
```

42.5 Make it enforceable

- Ontology header: dcterms:title, dcterms:description, dcterms:license, dcterms:creator, owl:versionIRI (ROBOT report flags missing title/description/licence as ERROR).
- Run **robot report** in CI; fail the build on ERROR, review WARN weekly. Its built-in *missing definition* check looks for the OBO property IAO:0000115, so if you use skos:definition add your own query via **--profile**.
- Definitions are reviewed by the domain steward, not only by the ontologist.

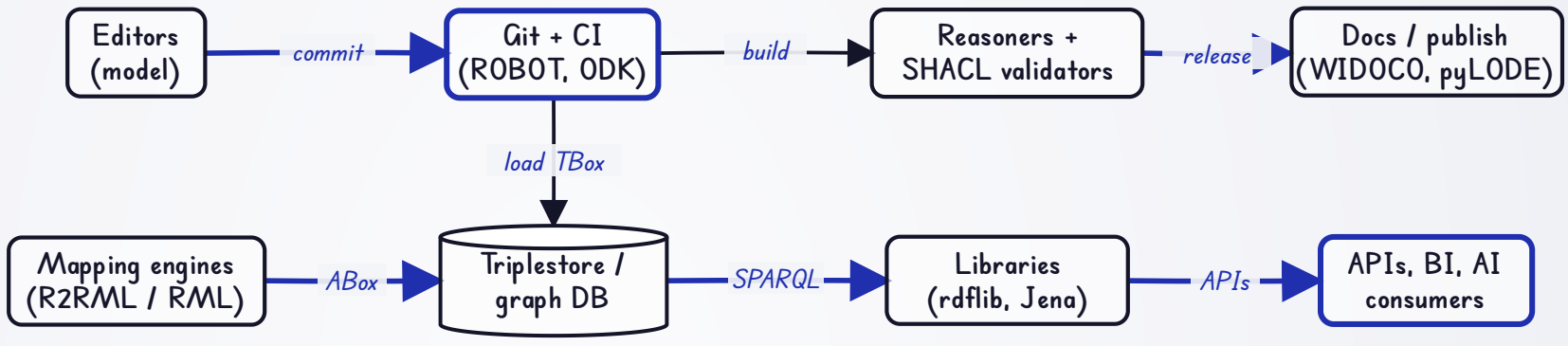
Key Takeaway: a class without a reviewed definition is just a name. Labels make terms findable; definitions make them shareable.

Sources: Seppälä, Ruttenberg & Smith, "Guidelines for writing definitions in ontologies", Ciência da Informação 46(1) (2017) · OBO Foundry Principles (obofoundry.org) (textual definitions) · W3C SKOS Reference, Recommendation (2009) · ROBOT docs, robot.obolibrary.org (v1.9.x) (report queries)

43 Tooling Landscape

Part D · Implementing Enterprise Ontology in Industry

43.1 Tool categories along the life cycle



43.2 Landscape by category

Category	Examples (alphabetical, not a ranking)	What to check
Ontology editors	Protégé / WebProtégé, TopBraid EDG, VocBench	OWL 2 + SKOS support, multi-user, Git round-trip
Pipeline / QC	ROBOT, Ontology Development Kit (ODK), OOPS! pitfall scanner	runs headless in CI
OWL reasoners	ELK (EL), HermiT, Konclude, Openllet	profile coverage vs speed
SHACL validators	Apache Jena SHACL, pySHACL, RDF4J ShaclSail, TopBraid SHACL	SHACL-SPARQL support, report format
RDF stores	Amazon Neptune, Apache Jena Fuseki/TDB2, Eclipse RDF4J, GraphDB, Oxigraph, Stardog, Virtuoso	SPARQL 1.1 conformance, named graphs, inference, scale-out, security
Property graphs	Neo4j (+ neosemantics plugin for RDF import)	how RDF/IRIs are kept
Mapping / OBDA	Morph-KGC, Ontop, RMLMapper, SDM-RDFizer	R2RML / RML version supported
Libraries	Apache Jena, OWL API, Owlready2, RDF4J, rdflib	language fit for your teams
Documentation	pyLODE, WIDOCO	HTML per term + content negotiation

43.3 Selection checklist

1. **Standards first**: W3C formats in and out (Turtle, JSON-LD, SPARQL, SHACL) - avoid lock-in.
2. Headless / CLI mode for CI.
3. Security model: SSO, named-graph or row-level access.
4. Licence and support model fit your procurement.
5. Run a **proof with your own CQs and data**, not vendor demos.

43.4 Minimum viable toolchain

Editor
desktop editor or plain Turtle in an IDE

Git + CI
ROBOT + pySHACL + SPARQL tests

Store
any SPARQL 1.1 triplestore

Publish
HTML docs + resolvable IRIs

Rule: tools change, standards persist. Keep the source of truth as text files in Git; every tool is a replaceable reader or writer of those files.

Sources: Jackson et al., "ROBOT: A Tool for Automating Ontology Workflows", BMC Bioinformatics 20:407 (2019) · Musen, "The Protégé Project: A Look Back and a Look Forward", AI Matters 1(4) (2015) · W3C SHACL, Recommendation (2017) · Calvanese et al., "Ontop: Answering SPARQL queries over relational databases", Semantic Web 8(3) (2017)

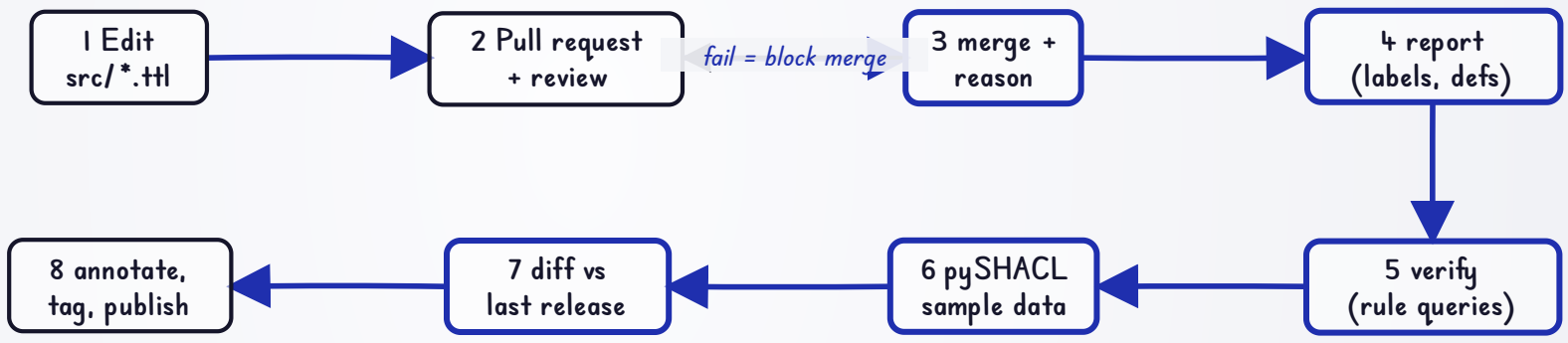
44 Ontology-as-Code CI/CD

Part D · Implementing Enterprise Ontology in Industry

44.1 Principles

- Source of truth = **text files in Git** (Turtle sorts and diffs well).
- Every change through a **pull request** reviewed by a steward.
- The build is **reproducible**: same commit, same release file.
- Tests live beside the model: CQs, rule queries, sample data.
- Releases are immutable, versioned, and published with a change log.
- Humans review meaning; machines review everything else.

44.2 Pipeline (one PR)



44.3 Build file (ROBOT commands checked against the docs)

```
# Makefile - real ROBOT / pySHACL commands (hermit: full OWL 2 DL; ELK: fast, EL only)
build/merged.owl: src/fnd.ttl src/ins.ttl src/clm.ttl
	robot merge --catalog catalog-v001.xml --input src/clm.ttl \
	  reason --reasoner hermit --equivalent-classes-allowed asserted-only \
	  --output build/merged.owl
report: build/merged.owl
	robot report --input build/merged.owl --fail-on ERROR --output build/report.tsv
cq: build/merged.owl     # each query returns rows only on a violation
	robot verify --input build/merged.owl --queries tests/verify/*.rq \
	  --output-dir build/verify
shacl:
	pyshacl -s src/clm.shacl.ttl -e build/merged.owl -i rdfs tests/data/sample.ttl
diff:
	robot diff --left release/clm-1.1.0.owl --right build/merged.owl \
	  --format markdown --output build/diff.md
release: report cq shacl diff
	robot annotate --input build/merged.owl \
	  --ontology-iri https://ontology.example.com/clm/ \
	  --version-iri https://ontology.example.com/clm/1.2.0/ --output release/clm.owl
```

Note: ROBOT **verify** treats each SELECT query as a rule (rows = violations). Positive CQs (page 42) run as **ASK**/SELECT tests with expected answers, e.g. via ROBOT **query** or a pytest + rdflib step.

44.4 Gates

Gate	Tool / command	Build fails when
Syntax & merge	robot merge (catalog)	Turtle error, unresolved import
Logic	robot reason (ELK or HermiT)	inconsistent ontology or unsatisfiable class
Hygiene	robot report --fail-on ERROR	missing label / title / licence, duplicate labels …
Rules	robot verify --queries	any rule query returns a row
Contracts	pyshacl on sample + edge-case data	conforms = false on "good" data, or true on "bad" data
Change review	robot diff --format markdown	never fails; diff is attached to the PR for humans

Key Takeaway: if a change cannot break the build, the build is not testing anything. Add one "bad data" file per shape so the SHACL gate is proven to bite.

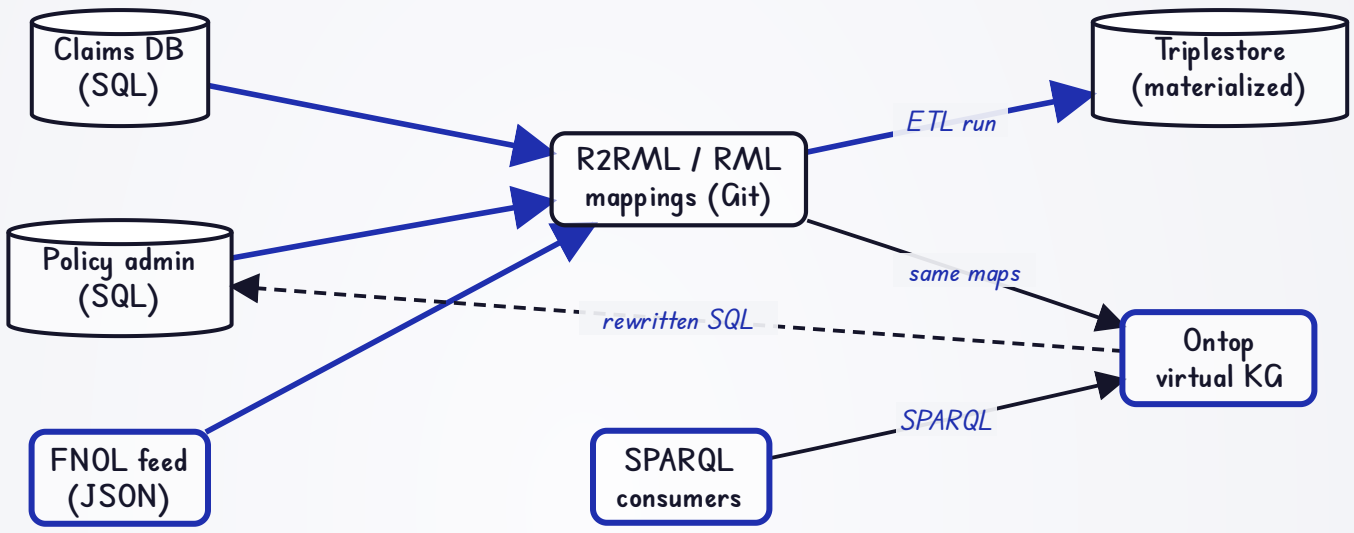
Sources: Jackson et al., "ROBOT: A Tool for Automating Ontology Workflows", BMC Bioinformatics 20:407 (2019) · ROBOT docs, robot.obolibrary.org (v1.9.x) (merge, reason, report, verify, diff, annotate) · W3C SHACL, Recommendation (2017) · pySHACL 0.40 / rdflib 7.x (RDFLib project, GitHub)

Rajesh Kumar Gupta

45 Mapping Source Data

Part D · Implementing Enterprise Ontology in Industry

45.1 Two ways to populate the knowledge graph



45.2 R2RML: relational → RDF

```
@prefix rr:  <http://www.w3.org/ns/r2rml#> .
@prefix clm: <https://ontology.example.com/clm/> .
<#ClaimMap> a rr:TriplesMap ;
 rr:logicalTable [ rr:tableName "CLAIM" ] ;
 rr:subjectMap [ rr:template
"https://data.example.com/{CLAIM_NO}" ;
              rr:class clm:Claim ] ;
 rr:predicateObjectMap [ rr:predicate clm:claimNumber ;
                    rr:objectMap [ rr:column "CLAIM_NO" ] ] ;
 rr:predicateObjectMap [ rr:predicate clm:onPolicy ;
    rr:objectMap [ rr:template
"https://data.example.com/{POLICY_NO}" ] ] .
```

R2RML is a W3C Recommendation (2012) for relational sources; Ontop reads R2RML mappings too.

45.3 RML: JSON / CSV / XML → RDF

```
@prefix rml: <http://w3id.org/rml/> .
@prefix clm: <https://ontology.example.com/clm/> .
<#FnolMap> a rml:TriplesMap ;
 rml:logicalSource [ rml:source [ a rml:RelativePathSource ;
      rml:root rml:MappingDirectory ; rml:path "fnol.json" ] ;
    rml:referenceFormulation rml:JSONPath ;
    rml:iterator "$.claims[*]" ] ;
 rml:subjectMap [ rml:template
"https://data.example.com/{$.claimNo}" ;
              rml:class clm:Claim ] ;
 rml:predicateObjectMap [ rml:predicate clm:claimNumber ;
    rml:objectMap [ rml:reference "$.claimNo" ] ] .
```

RML generalises R2RML to other formats. The current RML-Core text is a **Community Group draft** (not a W3C standard) - check rml.io / kg-construct for status.

45.4 Materialize or virtualize?

	Materialize (ETL to triplestore)	Virtualize (OBDA, e.g. Ontop)
Data freshness	as fresh as the last load	live: query rewritten to SQL at run time
Reasoning	any (materialize inferences)	limited to what rewriting supports (Ontop: OWL 2 QL / RDFS)
Integration of many sources	easy: everything in one store	needs federation or a DB that sees all sources
Load on source systems	batch extracts only	every query hits the source
Good first use	analytics, entity resolution, GraphRAG	operational lookups over a stable schema

45.5 Mapping sprint - steps that work

1. **Profile** : source tables/fields, keys, code values, null rates
2. **Map** : one triples map per business entity; IRI templates from the identifier policy (page 45)
3. **Crosswalk codes** : source codes → SKOS concepts with skos:exactMatch / broadMatch
4. **Validate** : run SHACL shapes on the output; track mapping coverage per CQ
5. **Own it** : mappings are versioned in Git and reviewed like the ontology

Key Takeaway: the mapping is where the ontology meets reality. Keep mappings declarative (R2RML/RML), not buried in ETL code, so they can be tested and reused.

Sources: W3C R2RML, Recommendation (2012) · W3C KG-Construct Community Group, RML-Core, Draft CG Report (2026) · Dimou et al., "RML: A Generic Language for Integrated RDF Mappings", LDOW (2014) · Calvanese et al., "Ontop: Answering SPARQL queries over relational databases", Semantic Web 8(3) (2017)

Rajesh Kumar Gupta

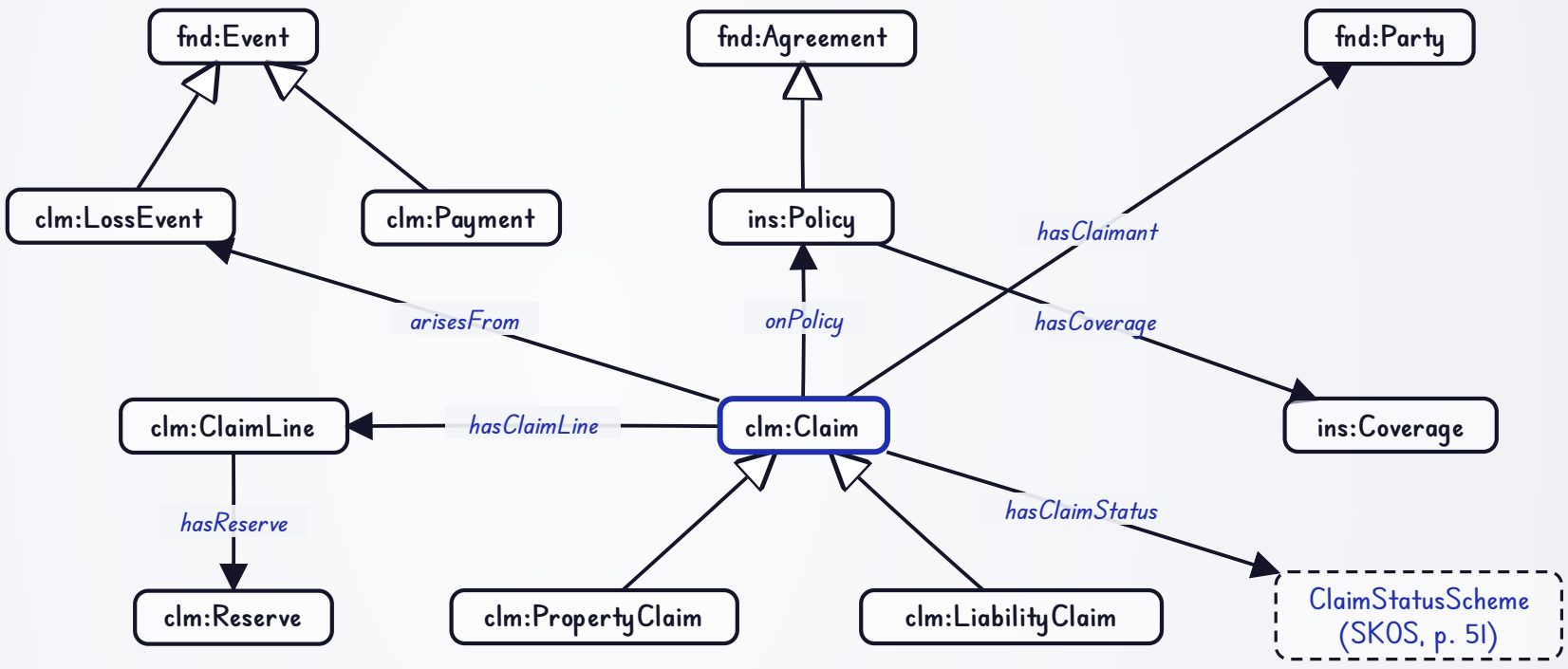
46 Worked Example I: Claims Classes

Part D · Implementing Enterprise Ontology in Industry

46.1 Goal of the worked example

Model the claims module (clm) on top of fnd and ins so that CQ-01 to CQ-06 (page 42) can be answered and tested. Everything below was parsed with rdflib 7.x; the OWL RL closure (owlrl) found no clash on the sample data.

46.2 Class hierarchy and main relations



hollow triangle = rdfs:subClassOf · plain arrow = object property (domain → range). Not drawn: clm:Claim ⊑ fnd:Event; fnd:Person / fnd:Organization under Party; claimsUnder (line → coverage) and hasPayment (line → payment), listed on page 51.

46.3 Turtle: header + classes (clm.ttl, part 1)

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  @prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .  @prefix fnd: <https://ontology.example.com/fnd/> .
@prefix ins: <https://ontology.example.com/ins/> .  @prefix clm: <https://ontology.example.com/clm/> .

<https://ontology.example.com/clm/> a owl:Ontology ;
   owl:versionIRI <https://ontology.example.com/clm/1.2.0/> ;
   owl:imports <https://ontology.example.com/fnd/> , <https://ontology.example.com/ins/> ;
   dcterms:title "Claims Ontology"@en ;
   dcterms:license <https://creativecommons.org/licenses/by/4.0/> .

clm:Claim a owl:Class ; rdfs:label "claim"@en ; rdfs:subClassOf fnd:Event ;
   skos:definition "A request to an insurer for payment or service under a policy, following a loss event."@en ;
   rdfs:subClassOf [ a owl:Restriction ; owl:onProperty clm:onPolicy ;
      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ; owl:onClass ins:Policy ] .
clm:PropertyClaim  a owl:Class ; rdfs:subClassOf clm:Claim ; rdfs:label "property claim"@en .
clm:LiabilityClaim a owl:Class ; rdfs:subClassOf clm:Claim ; rdfs:label "liability claim"@en .
clm:ClaimLine    a owl:Class ; rdfs:label "claim line"@en .
clm:Reserve      a owl:Class ; rdfs:label "reserve"@en .
clm:LossEvent    a owl:Class ; rdfs:subClassOf fnd:Event ; rdfs:label "loss event"@en .
clm:Payment      a owl:Class ; rdfs:subClassOf fnd:Event ; rdfs:label "payment"@en .
[] a owl:AllDisjointClasses ; owl:members ( clm:Claim clm:LossEvent clm:ClaimLine clm:Reserve ) .
```

46.4 Design decisions worth an ADR

- **Claimant is a role, not a class**: a person is a claimant only relative to a claim (anti-rigid), so model it as clm:hasClaimant, not "Claimant ⊆ Person".
- **Claim and LossEvent are both fnd:Event, but disjoint**: the loss happens once; the claim is the long-lived handling of it, with lines, reserves and status.
- **Exactly one policy** is stated in OWL for meaning, but under the open world a missing policy is not an error - SHACL enforces it (page 51).
- **Disjointness** makes errors visible: a node typed both Claim and Reserve becomes inconsistent.

Key Takeaway: start the module from the CQs and the shared foundation; every clm class either specialises an fnd/ins class or is justified in writing.

Sources: W3C OWL 2 Structural Specification & Primer, 2nd ed. (2012) · Noy & McGuinness, "Ontology Development 101", Stanford KSL-01-05 (2001) · Guarino & Welty, "Evaluating ontological decisions with OntoClean", CACM 45(2) (2002)

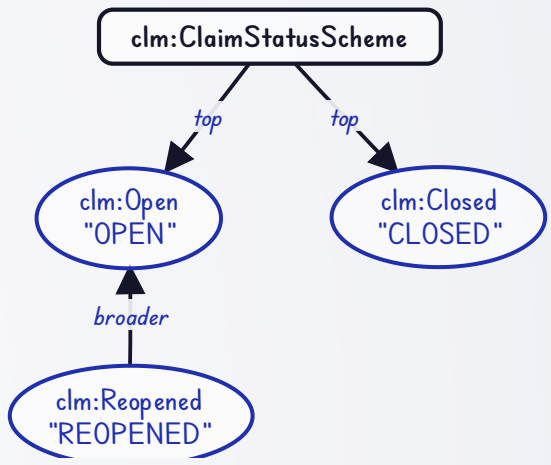
47 Worked Example II: Axioms & Shapes

Part D · Implementing Enterprise Ontology in Industry

47.1 Properties, axioms and the SKOS status list

```
# clm.ttl part 2 (prefixes as on page 50)
clm:onPolicy     a owl:ObjectProperty ; rdfs:domain clm:Claim ; rdfs:range ins:Policy .
clm:arisesFrom   a owl:ObjectProperty ; rdfs:domain clm:Claim ; rdfs:range clm:LossEvent .
clm:hasClaimant  a owl:ObjectProperty ; rdfs:domain clm:Claim ; rdfs:range fnd:Party .
clm:hasClaimLine a owl:ObjectProperty ; rdfs:domain clm:Claim ; rdfs:range clm:ClaimLine ;
                 owl:inverseOf clm:partOfClaim .
clm:hasReserve   a owl:ObjectProperty ; rdfs:domain clm:ClaimLine ; rdfs:range clm:Reserve .
clm:hasPayment   a owl:ObjectProperty ; rdfs:domain clm:ClaimLine ; rdfs:range clm:Payment .
clm:claimsUnder  a owl:ObjectProperty ; rdfs:domain clm:ClaimLine ; rdfs:range ins:Coverage .
clm:hasClaimStatus a owl:ObjectProperty ; rdfs:domain clm:Claim ; rdfs:range skos:Concept .
clm:claimNumber  a owl:DatatypeProperty ; rdfs:domain clm:Claim ; rdfs:range xsd:string .
clm:reportedOn   a owl:DatatypeProperty ; rdfs:domain clm:Claim ; rdfs:range xsd:date .
clm:reserveAmount a owl:DatatypeProperty ; rdfs:domain clm:Reserve ; rdfs:range xsd:decimal .

clm:ClaimStatusScheme a skos:ConceptScheme ; skos:hasTopConcept clm:Open , clm:Closed .
clm:Open     a skos:Concept ; skos:prefLabel "Open"@en ; skos:notation "OPEN" ;
             skos:inScheme clm:ClaimStatusScheme ; skos:topConceptOf clm:ClaimStatusScheme .
clm:Reopened a skos:Concept ; skos:prefLabel "Reopened"@en ; skos:notation "REOPENED" ;
             skos:broader clm:Open ; skos:inScheme clm:ClaimStatusScheme .
clm:Closed   a skos:Concept ; skos:prefLabel "Closed"@en ; skos:notation "CLOSED" ;
             skos:inScheme clm:ClaimStatusScheme ; skos:topConceptOf clm:ClaimStatusScheme .
```



Status is a **SKOS concept**, not a string: Reopened is narrower than Open, so CQ-02 counts it via skos:broader*.

47.2 SHACL shapes (clm.shacl.ttl): the data contract

```
@prefix sh:  <http://www.w3.org/ns/shacl#> .      # + xsd, fnd, ins, clm
@prefix clmsh: <https://ontology.example.com/clm/shapes/> .
clmsh:ClaimShape a sh:NodeShape ; sh:targetClass clm:Claim ;
  sh:property [ sh:path clm:claimNumber ; sh:datatype xsd:string ;
     sh:minCount 1 ; sh:maxCount 1 ; sh:pattern "^CLM-[0-9]{4,}$" ] ;
  sh:property [ sh:path clm:onPolicy ; sh:class ins:Policy ;
     sh:minCount 1 ; sh:maxCount 1 ;
     sh:message "Every claim must reference exactly one policy."@en ] ;
  sh:property [ sh:path clm:hasClaimStatus ; sh:minCount 1 ; sh:maxCount 1 ;
     sh:in ( clm:Open clm:Reopened clm:Closed ) ] ;
  sh:property [ sh:path clm:hasClaimant ; sh:class fnd:Party ; sh:minCount 1 ] ;
  sh:property [ sh:path clm:reportedOn ; sh:datatype xsd:date ;
     sh:minCount 1 ; sh:severity sh:Warning ] .
clmsh:ReserveShape a sh:NodeShape ; sh:targetClass clm:Reserve ;
  sh:property [ sh:path clm:reserveAmount ; sh:datatype xsd:decimal ;
     sh:minCount 1 ; sh:minInclusive 0 ] .
```

47.3 Sample data: one good, one bad claim

```
# sample.ttl (d: = https://data.example.com/)
d:CLM-1001 a clm:PropertyClaim ; clm:claimNumber "CLM-1001" ;
  clm:onPolicy d:POL-77 ; clm:hasClaimant d:P-12 ;
  clm:hasClaimStatus clm:Open ;
  clm:reportedOn "2026-03-14"^^xsd:date ;
  clm:hasClaimLine d:CL-1001-1 .          # line -> reserve 12500.00
d:CLM-1002 a clm:Claim ; clm:claimNumber "1002" ;   # no policy, no date
  clm:hasClaimant d:P-12 ; clm:hasClaimStatus clm:Pending ;
  clm:hasClaimLine d:CL-1002-1 .          # line -> reserve -500.00
```

47.4 Run it

```
pyshacl -s clm.shacl.ttl -e merged.ttl -i rdfs sample.ttl
# merged.ttl = fnd + ins + clm   (exit code 1 = not conforming)
```

Actual result (pySHACL 0.40.2): **Conforms: False**, 5 results; CLM-1001 passes (Person accepted as Party via -e + rdfs).

47.5 Validation report (real output, summarised)

Focus node	Path	Constraint	Severity
d:CLM-1002	clm:onPolicy	minCount 1 ("Every claim must reference exactly one policy.")	Violation
d:CLM-1002	clm:claimNumber	pattern ^CLM-[0-9]{4,}$ ("1002")	Violation
d:CLM-1002	clm:hasClaimStatus	sh:in (clm:Pending not in list)	Violation
d:R-2	clm:reserveAmount	minInclusive 0 (-500.00)	Violation
d:CLM-1002	clm:reportedOn	minCount 1	Warning

Key Takeaway: OWL said "every claim has exactly one policy" and stayed silent about CLM-1002 (open world). SHACL turned the same business rule into a failing test. Use both: OWL for meaning, SHACL for data quality.

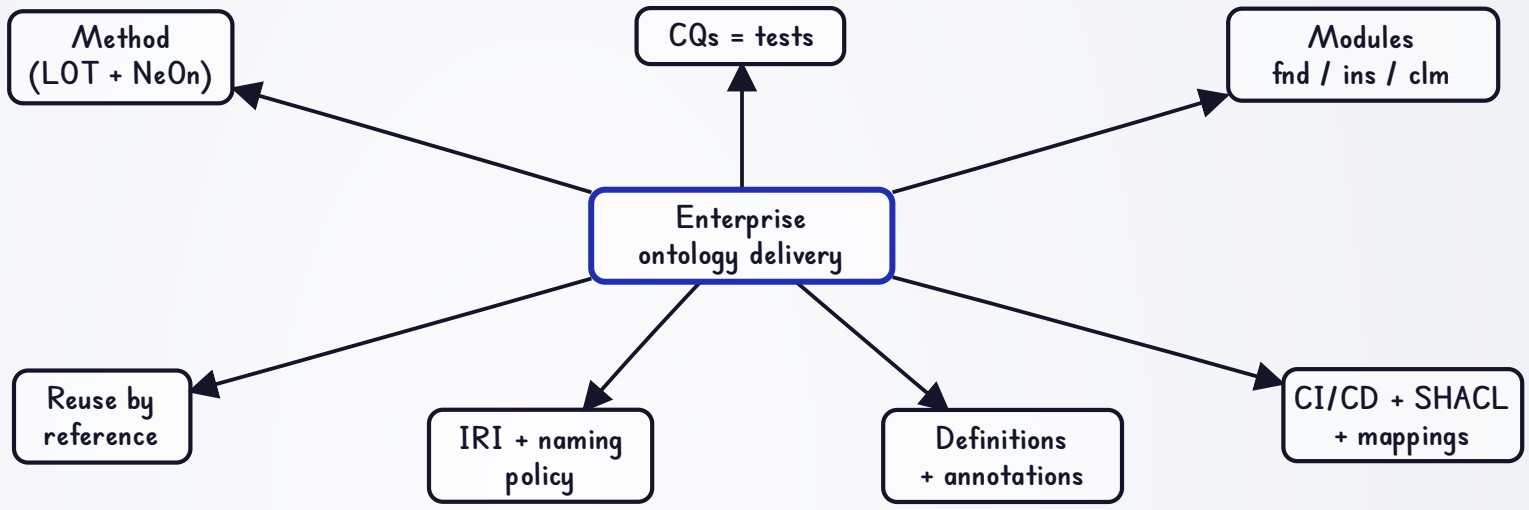
Sources: W3C SHACL, Recommendation (2017) · W3C SKOS Reference, Recommendation (2009) · W3C OWL 2 Structural Specification & Primer, 2nd ed. (2012) · pySHACL 0.40 / rdflib 7.x (RDFLib project, GitHub)

Rajesh Kumar Gupta

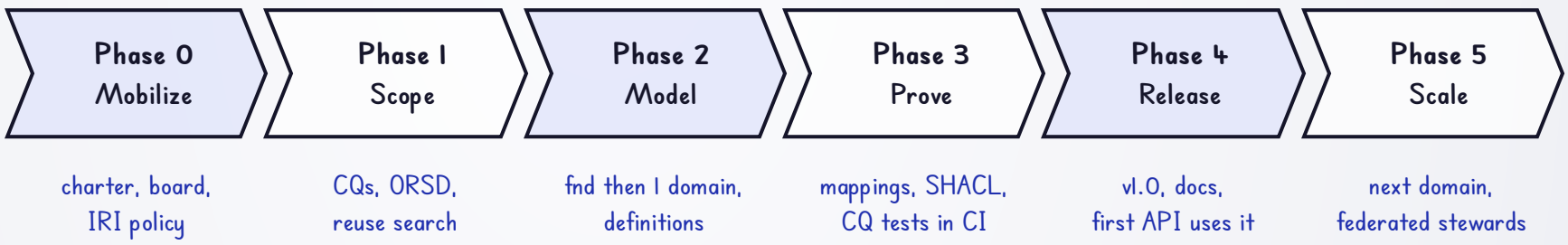
48 Part D Recap & Roadmap

Part D · Implementing Enterprise Ontology in Industry

48.1 Part D on one page



48.2 Implementation roadmap



phases overlap and repeat per module; durations depend on scope (no fixed timeline implied)

48.3 Phase exit criteria

Phase	Exit criterion (evidence, not opinion)
0 Mobilize	board charter signed; base IRI and prefix registry published
1 Scope	prioritised CQs with owners; in/out list agreed; reuse decisions logged
2 Model	all must-have CQs have the terms they need; every term labelled and defined
3 Prove	CI green: reason, report, verify, SHACL; CQs return expected answers on real mapped data
4 Release	versioned release + docs; at least one consumer (API, report or KG load) in production
5 Scale	second domain onboarded by its own steward using the same pipeline

48.4 Recap checklist

- Pick one life cycle; borrow requirement and reuse tools from others.
- Every term traces to a CQ; every CQ has an owner and a test.
- Layered, acyclic modules; one namespace and steward each.
- Reuse by IRI, extract or bridge axioms - never by copy-paste.
- IRI policy before the first term; versions via owl:versionIRI.
- Label + genus-differentia definition for every term.
- Ontology as code: PR review, ROBOT gates, SHACL on sample data.
- Declarative mappings (R2RML/RML), versioned and validated.

Key Takeaway: deliver one thin, tested vertical slice (fnd + claims + one consuming API) before widening. Value shown early buys the time to do the rest properly.

Sources: Poveda-Villalón, Fernández-Izquierdo, Fernández-López & García-Castro, "LOT", Eng. Appl. of AI 111 (2022) · Suárez-Figueroa, Gómez-Pérez & Fernández-López, "The NeOn Methodology", in Ontology Engineering in a Networked World, Springer (2012) · Jackson et al., "ROBOT: A Tool for Automating Ontology Workflows", BMC Bioinformatics 20:407 (2019) · Grüninger & Fox, "Methodology for the Design and Evaluation of Ontologies", IJCAI-95 Workshop (1995)

Rajesh Kumar Gupta

49 What is a KG?

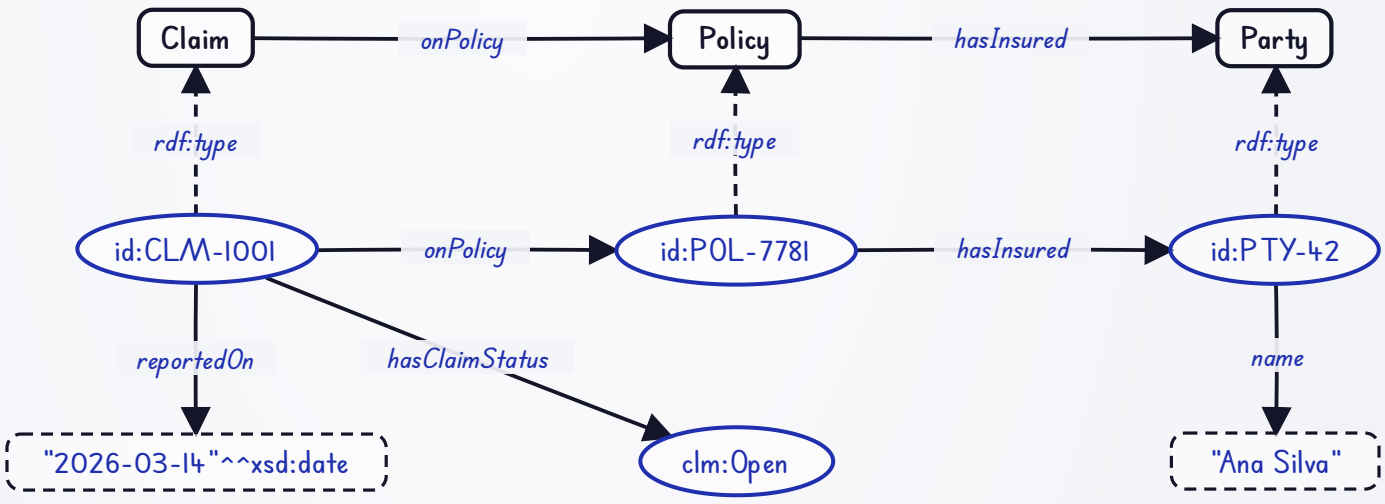
Part E · Knowledge Graphs

49.1 Working definition

Hogan et al. (2021) adopt an inclusive definition: a knowledge graph is "a graph of data intended to accumulate and convey knowledge of the real world", whose nodes are entities of interest and whose edges are relations between them. Google popularised the idea in 2012 as **"things, not strings"** (Singhal).

- **Open KGs** are published on the Web (DBpedia, Wikidata, YAGO); **enterprise KGs** are internal and serve commercial use-cases (Hogan et al. 2021).
- Practitioner reading: **KG = instance data (ABox) + schema/ontology (TBox) + identity + context** (provenance, time, source).

49.2 A tiny claims KG: schema row + instance rows



Top row = **classes** from the ontology. Ellipses = **entities** with enterprise IRIs. Dashed boxes = **literals**. Dashed arrows = *rdf:type*. The same edge name (*onPolicy*) is declared once in the TBox and used millions of times in the ABox.

49.3 Short history (verified milestones)



49.4 A KG is NOT …

- **just a graph database** – the DB is storage; the KG is the curated content + meaning.
- **just an ontology** – ontology is the schema; KG adds the facts.
- **a vector store** – embeddings find similar text; a KG states explicit, queryable relations.

49.5 Monday-morning test

1. Can you name the **entities** and **relations** that matter (competency questions)?
2. Does every entity have a **stable IRI** and a **type** from the ontology?
3. Can you say **where each fact came from** and when?

Key Takeaway: A knowledge graph is the **instance level** of your enterprise ontology: identified things, typed by shared classes, linked by shared relations, with provenance.

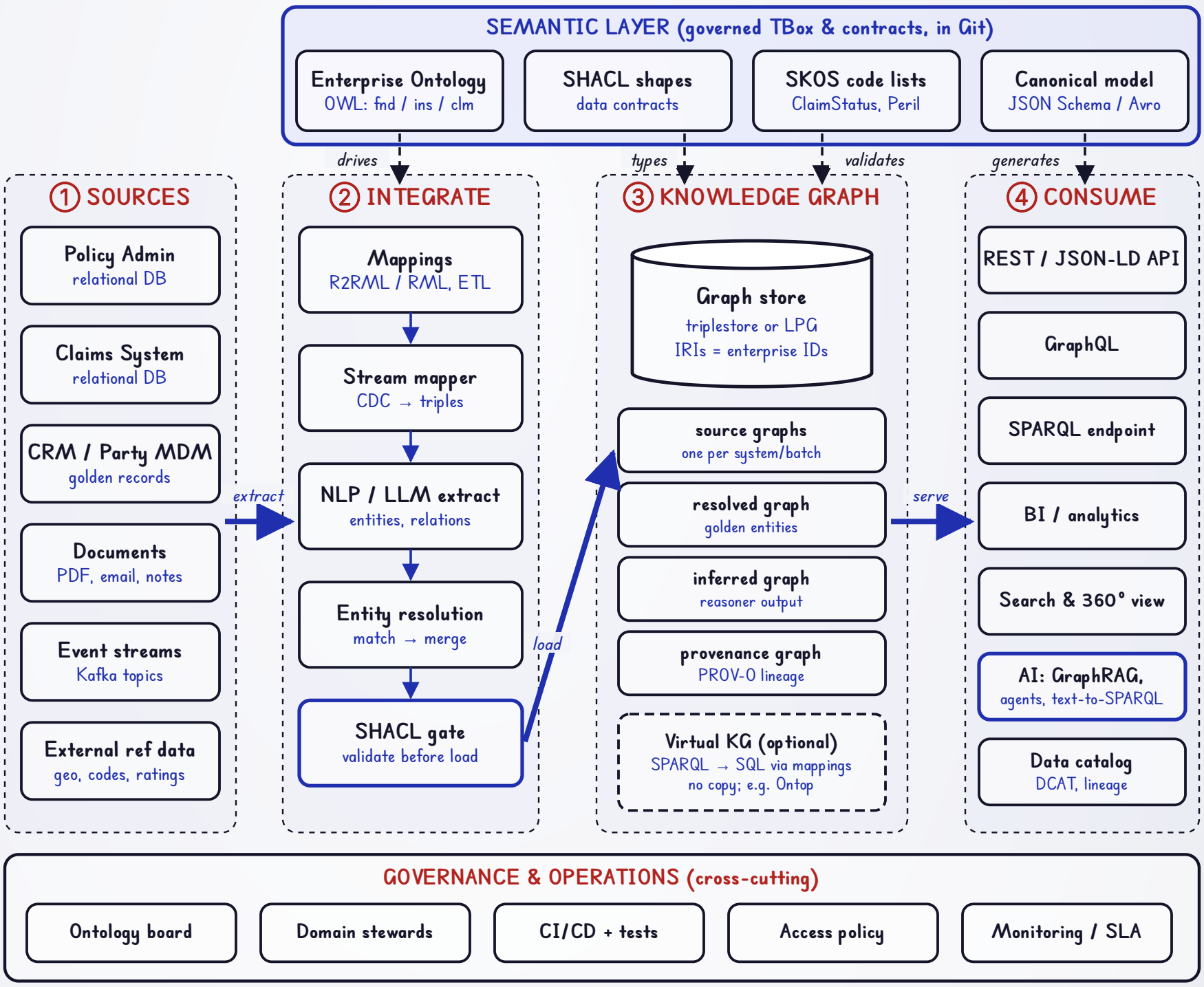
Sources: Hogan et al., "Knowledge Graphs", ACM Computing Surveys 54(4) (2021) · Singhal, "Introducing the Knowledge Graph: things, not strings", Google Blog (2012) · Berners-Lee, Hendler & Lassila, "The Semantic Web", Scientific American (2001)

Rajesh Kumar Gupta

50 The KG in Enterprise Architecture

Part E · Knowledge Graphs

50.1 Reference architecture (insurance example)



50.2 How to read it

1. **Semantic layer** : the governed meaning. It never stores instance data; it **drives** mappings, **types** entities, **validates** loads and **generates** API schemas.
2. **Integrate** : map sources to ontology terms, resolve identities, and stop bad data at the SHACL gate.
3. **KG** : separate named graphs by source, resolution, inference and provenance so each can be reloaded alone.
4. **Consume** : every channel reuses the same IRIs and terms, so a "Claim" means one thing everywhere.

50.3 Placement decisions

Question	Usual answer
System of record?	**No.** KG integrates; sources stay authoritative
Copy or virtualize?	Copy for joins & AI; virtual for live, low-volume
Who owns the TBox?	Ontology team + domain stewards
Who owns the ABox?	Source domains, via mappings

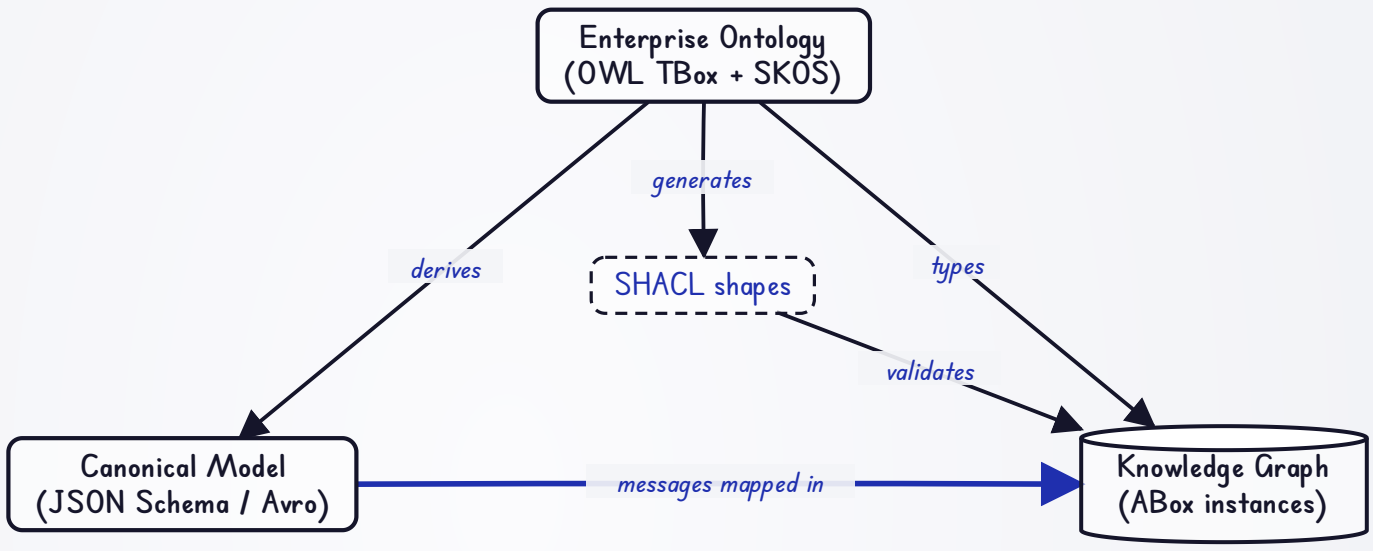
Sources: Hogan et al., Knowledge Graphs, ACM CSUR (2021) · W3C R2RML (2012) · Calvanese et al., "Ontop", Semantic Web Journal (2017) · W3C SHACL (2017)

Rajesh Kumar Gupta

⑤① Ontology + Canonical + KG

Part E · Knowledge Graphs

51.1 Three artifacts, one meaning



The ontology says **what things are**; the canonical model says **how they travel** between systems; the KG holds **what is true** about specific things. SHACL sits between the TBox and the data as the executable contract.

51.2 The same term in all three

```
# TBox - meaning (ontology)   # prefixes omitted
clm:Claim a owl:Class ;
  skos:definition
    "A demand for payment under a policy."@en .
clm:onPolicy a owl:ObjectProperty ;
  rdfs:domain clm:Claim ;
  rdfs:range  ins:Policy .
```

```
# ABox - facts (knowledge graph)
id:CLM-1001 a clm:Claim ;
  clm:onPolicy id:POL-7781 .
```

```
// Canonical - exchange shape (JSON Schema)
"ClaimMessage": {
  "type": "object",
  "x-ontology": "clm:Claim",
  "properties": {
    "policyId": {"type": "string",
                 "x-ontology": "clm:onPolicy"} } }
```

x-ontology is a house convention: JSON Schema 2020-12 treats unknown keywords as annotations, so validators ignore it while generators can read it.

51.3 Who governs what

Artifact	Level	Owner	Changed by	Typical consumers
Ontology (OWL, SKOS)	TBox	Ontology team + stewards	Change request, board review	Modelers, KG, AI
SHACL shapes	contract	Ontology team	Generated / reviewed in CI	Pipelines, APIs
Canonical schemas	exchange	Integration / API team	Semver release from ontology	Apps, events
KG instance data	ABox	Source domains	Mappings + loads	Analytics, search, AI

51.4 Decision rules

- A new **business concept** goes into the ontology first; schemas and shapes follow from it, never the reverse.
- A field that exists only for transport (message id, retry count) belongs in the canonical **envelope**, not the ontology.
- If KG data needs a class the ontology lacks, raise a change request; do not mint local classes in the ABox.

Key Takeaway: Ontology = meaning, canonical = shape, KG = facts. Keep one source of truth for meaning and generate the rest.

Sources: W3C OWL 2 Primer (2012) · W3C SHACL (2017) · JSON Schema 2020-12 Core (annotations) · Hohpe & Woolf, Enterprise Integration Patterns (2003)

52 KG Construction Pipeline

Part E · Knowledge Graphs

52.1 The seven-step pipeline

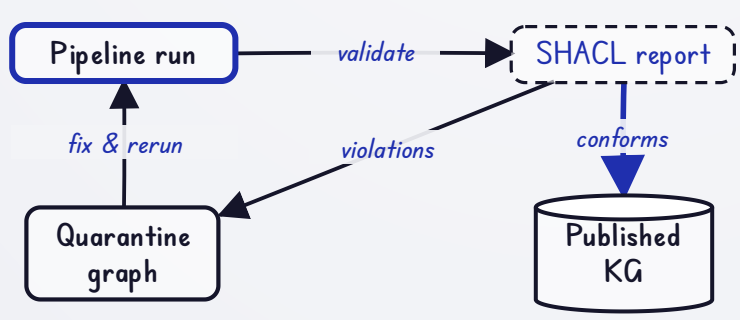


Step	What happens	Standards / tools	Exit gate
Ingest	Pull extracts, CDC or files; keep raw copy	ETL/ELT, Kafka, object store	Row counts reconcile
Map	Rows/JSON → triples with ontology terms	W3C R2RML, RML, SPARQL Anything	Every column mapped or waived
Resolve	Find records of the same real thing	Blocking + matching (page 57)	Match review sample
Link	Connect claims, policies, parties across sources	IRIs, foreign keys, lookups	No dangling references
Enrich	Infer types, extract from text, add codes	OWL RL, SHACL rules, NLP/LLM	Inferred graph kept separate
Validate	Check data against shapes	SHACL engine	No sh:Violation
Publish	Swap named graph, update catalog	Graph store, DCAT, PROV-O	Smoke-test CQs pass

52.2 Mapping example (W3C R2RML)

```
@prefix rr:  <http://www.w3.org/ns/r2rml#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix clm: <https://ontology.example.com/clm/> .
<#ClaimMap> a rr:TriplesMap ;
 rr:logicalTable [ rr:tableName "CLAIM" ] ;      # source table
 rr:subjectMap [ rr:class clm:Claim ;
  rr:template "https://data.example.com/id/claim/{CLAIM_ID}" ] ;
 rr:predicateObjectMap [ rr:predicate clm:reportedOn ;
  rr:objectMap [ rr:column "REPORTED_DT" ; rr:datatype xsd:date ] ] .
```

52.3 Validation loop



52.4 Batch or incremental?

- **Batch**: rebuild a source graph and swap it atomically.
- **Incremental**: apply deltas from CDC events; needs stable IRIs and delete handling.
- Keep mappings in Git next to the ontology; test them in CI.

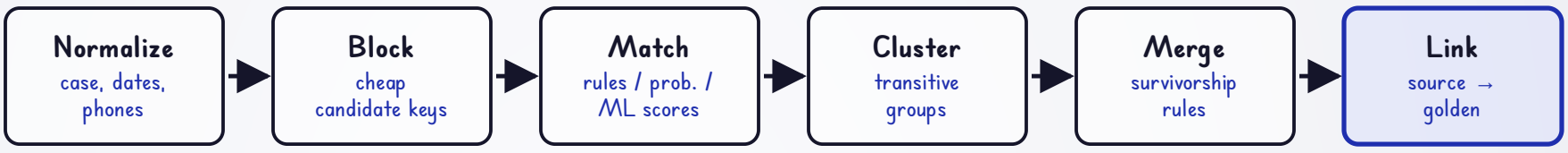
Key Takeaway: Make every step repeatable and every output a separate named graph, so you can rerun one step without rebuilding the KG.

Sources: W3C R2RML: RDB to RDF Mapping Language (2012) · RML specification, rml.io · W3C SHACL (2017) · Hogan et al., Knowledge Graphs, ACM CSUR (2021)

53 Entity Resolution

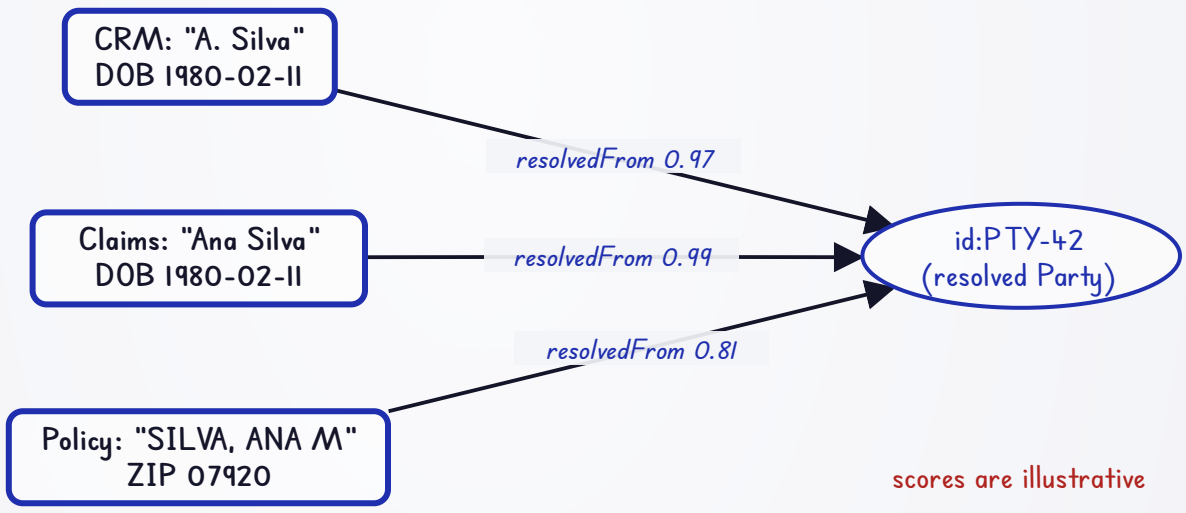
Part E · Knowledge Graphs

53.1 The entity resolution (ER) workflow



Christophides et al. (2020) describe end-to-end ER as **blocking** (to avoid comparing all pairs), **matching**, and **clustering**; the classic probabilistic model is Fellegi & Sunter (1969).

53.2 Three source records, one real person



53.3 Recommended identity pattern

```
@prefix fnd:  <https://ontology.example.com/fnd/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix id:   <https://data.example.com/id/> .
id:PTY-42 a fnd:Person ;
  fnd:resolvedFrom     id:crm-A991 , id:clm-55107 ;
  prov:wasDerivedFrom id:crm-A991 , id:clm-55107 .
id:crm-A991 fnd:sourceSystem "CRM" ;
  fnd:sourceKey "A991" .
```

fnd:resolvedFrom, *sourceSystem*, *sourceKey* are terms of our example ontology; *prov:wasDerivedFrom* is W3C PROV-O.

53.4 Which link to use?

Link	Use when	Caution
owl:sameAs	truly the same individual	merges ALL statements; errors spread
skos:exactMatch	mapping SKOS concepts / codes	not for people or claims
own link property	source record → golden entity	needs a definition in fnd:

- Keep source records as nodes; never overwrite them.
- Store match score, rule version and date on the link (reification or RDF 1.2 triple terms).
- Route low-confidence matches to a steward queue.

Rule of thumb: Mint one enterprise IRI per real-world thing and link source records to it; reserve owl:sameAs for cases you would defend in an audit.

Sources: Christophides et al., "An Overview of End-to-End Entity Resolution for Big Data", ACM CSUR (2020) · Fellegi & Sunter, "A Theory for Record Linkage", JASA (1969) · Halpin et al., "When owl:sameAs isn't the Same", ISWC (2010) · W3C PROV-O (2013)

54 Storage, Named Graphs & Provenance

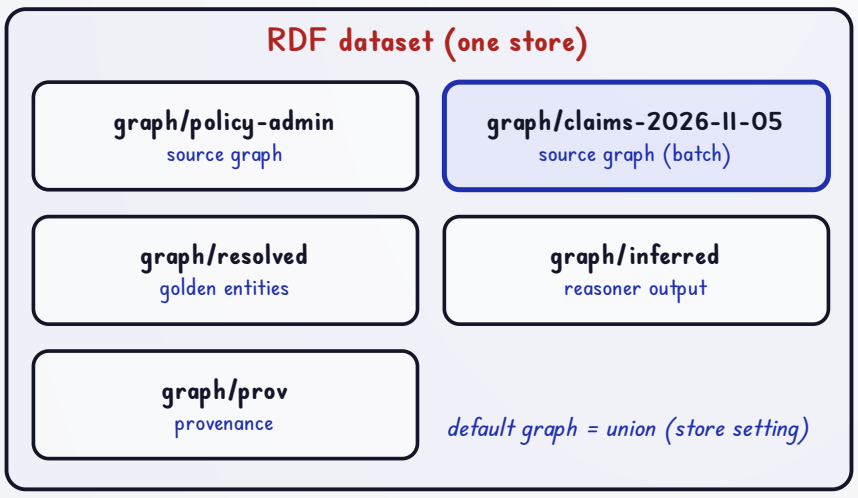
Part E · Knowledge Graphs

54.1 Triplestore vs property-graph database

Aspect	RDF triplestore	Labeled property graph (LPG)
Data model	triples / quads; IRIs everywhere	nodes + edges with key-value properties
Schema & meaning	RDFS/OWL, SHACL, SKOS are data too	labels + constraints; ontology kept outside
Query	SPARQL 1.1 (1.2 in progress)	Cypher, ISO GQL (ISO/IEC 39075:2024)
Global identity	native (IRIs, linked data)	internal ids; add an IRI property
Edge attributes	reification / RDF 1.2 triple terms	native edge properties
Choose when	integration, standards, reasoning	path analytics, app-centric traversal

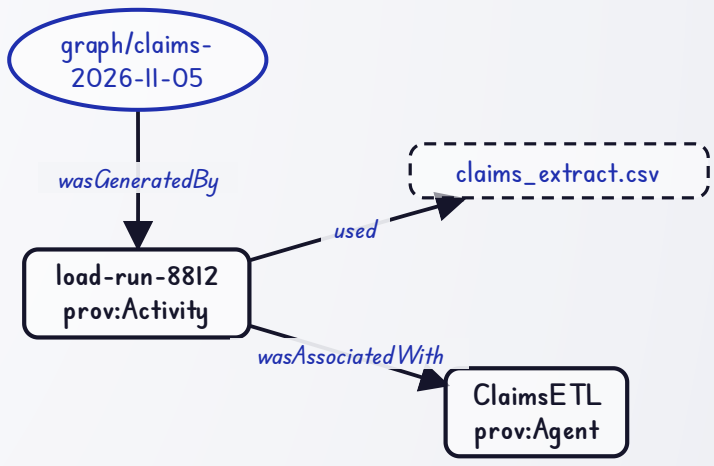
Both can hold an enterprise KG. Many teams keep the ontology in RDF and publish a projection to an LPG for graph analytics.

54.2 Named graphs = context



- RDF 1.1 **datasets**: one default graph + named graphs (quads).
- Use one graph per source and per batch: easy reload, delete and access control.
- Whether the default graph is the union is a store setting – check yours.

54.3 Provenance with PROV-O



PROV-O core: **Entity**, **Activity**, **Agent**; relations such as *wasGeneratedBy*, *used*, *wasAssociatedWith*, *wasDerivedFrom*.

54.4 TriG: data and its provenance in one file

```
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix clm:  <https://ontology.example.com/clm/> .
@prefix id:   <https://data.example.com/id/> .
@prefix g:    <https://ontology.example.com/graph/> .
g:claims-2026-11-05 {
  id:CLM-1001 a clm:Claim ; clm:onPolicy id:POL-7781 . }
g:prov {
  g:claims-2026-11-05 a prov:Entity ;
    prov:wasGeneratedBy id:load-run-8812 .
  id:load-run-8812 a prov:Activity ;
    prov:startedAtTime "2026-11-05T02:00:00Z"^^xsd:dateTime ;
    prov:wasAssociatedWith id:ClaimsETL . }
```

Key Takeaway: Store **facts** in source named graphs and **facts about those graphs** in a provenance graph. Lineage questions then become ordinary SPARQL queries.

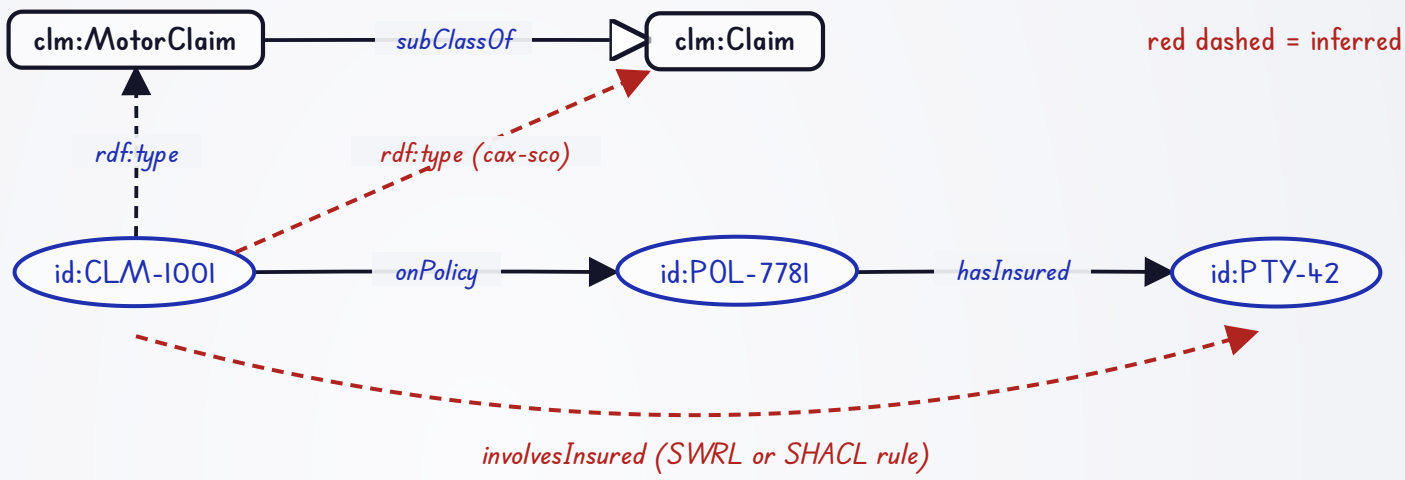
Sources: W3C RDF 1.1 Concepts (2014) & TriG (2014) · W3C PROV-O, Recommendation (2013) · Carroll et al., "Named Graphs, Provenance and Trust", WWW (2005) · ISO/IEC 39075:2024 GQL

Rajesh Kumar Gupta

55 Reasoning & Inference

Part E · Knowledge Graphs

55.1 Asserted vs inferred



55.2 Rule families you will actually use

Family	Example	Standard	Use for
RDFS entailment	rdfs9: subclass ⇒ type; rdfs2/3: domain, range	RDF 1.1 Semantics	type hierarchies
OWL 2 RL rules	cax-sco, prp-invl, prp-trp, prp-symp	OWL 2 Profiles (2012)	inverse, transitive, sameAs
SWRL	Horn rules over OWL classes/properties	W3C Member Submission (2004)	derived relations
SHACL-AF rules	sh:TripleRule, sh:SPARQLRule	W3C WG Note (2017)	rules next to shapes
SPARQL CONSTRUCT	hand-written derivation queries	SPARQL 1.1 (2013)	anything else, explicit

```
# SWRL (human-readable syntax)
clm:Claim(?c) ^ clm:onPolicy(?c, ?p) ^
  ins:hasInsured(?p, ?i)
    -> clm:involvesInsured(?c, ?i)
```

SWRL is a **Member Submission**, not a W3C Recommendation; tool support varies.

```
# SHACL-AF triple rule  # prefixes omitted
clmsh:ClaimRuleShape a sh:NodeShape ;
  sh:targetClass clm:Claim ;
  sh:rule [ a sh:TripleRule ;
    sh:subject sh:this ;
    sh:predicate clm:involvesInsured ;
    sh:object [ sh:path
      ( clm:onPolicy ins:hasInsured ) ] ] .
```

SHACL 1.2 Inference Rules is a 2026 Working Draft – check current status at w3.org.

55.3 Materialize or reason at query time?

	Materialize (forward chain)	Query-time (backward / rewrite)
Read speed	fast, plain lookups	slower, rules expand queries
Freshness	must re-run after each load	always current
Storage	extra triples (keep in graph/inferred)	none
Good for	stable TBox, heavy read APIs, AI retrieval	volatile data, virtual KGs (OWL 2 QL)

Key Takeaway: Start with RDFS + a few OWL RL rules, write inferences to a separate named graph, and test each rule with a competency question.

Sources: W3C OWL 2 Profiles, OWL 2 RL rules (2nd ed. 2012) · Horrocks et al., SWRL, W3C Member Submission (2004) · W3C SHACL Advanced Features, WG Note (2017) · W3C RDF 1.1 Semantics (2014)

56 Querying, Federation, Virtual KGs

Part E · Knowledge Graphs

56.1 Three ways to answer a cross-source question

Pattern	How	Pros	Cons
Materialized KG	copy & map into one store	fast joins, reasoning, AI	latency of loads; storage
Federated SPARQL	SERVICE calls to other endpoints	no copy; owner keeps data	network cost; remote SLA
Virtual KG (OBDA)	SPARQL rewritten to SQL via mappings	live data, one model	limited reasoning (OWL 2 QL)

56.2 SPARQL 1.1 federation

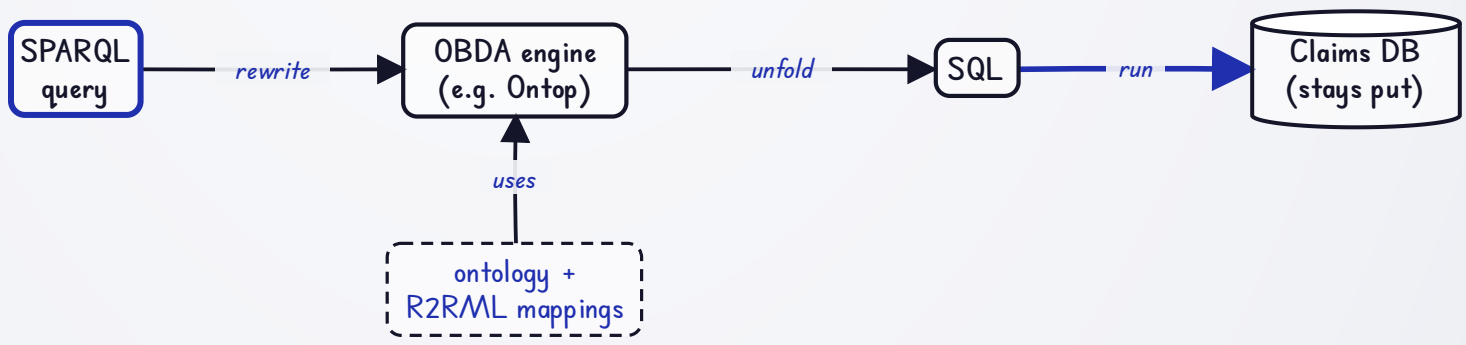
```
PREFIX clm: <https://ontology.example.com/clm/>
PREFIX ins: <https://ontology.example.com/ins/>
PREFIX geo: <https://ontology.example.com/geo/>
SELECT ?claim ?policy ?region WHERE {
  ?claim a clm:Claim ;
         clm:hasClaimStatus clm:Open ;
         clm:onPolicy ?policy .
  SERVICE <https://geo.example.com/sparql> {
    ?policy ins:riskLocation ?loc .
    ?loc geo:inRegion ?region . } }
```

56.3 Cypher / GQL

```
// Same question in Cypher (LPG projection)
MATCH (c:Claim {status: 'open'})
      -[:ON_POLICY]->(p:Policy)
      -[:RISK_LOCATION]->(l:Location)
RETURN c.claimNumber, p.policyNumber, l.region
```

- ISO GQL (ISO/IEC 39075:2024) is the ISO standard property-graph query language.
- Keep LPG labels traceable to ontology IRIs (e.g. a mapping table *:Claim* ↔ *clm:Claim*).

56.4 Virtual KG: query in, SQL out



Ontop translates SPARQL over an ontology into SQL using R2RML mappings, so the relational source remains the system of record (Calvanese et al. 2017).

56.5 Practical rules

1. Default to **materialize** the core (Party, Policy, Claim); **federate** reference data owned elsewhere.
2. Put **LIMIT**, timeouts and selective patterns first inside SERVICE blocks; remote endpoints are not free.
3. Use **SERVICE SILENT** only when a missing remote answer is acceptable.
4. Publish one **logical schema** (the ontology) whatever the physical pattern, so consumers never see the difference.

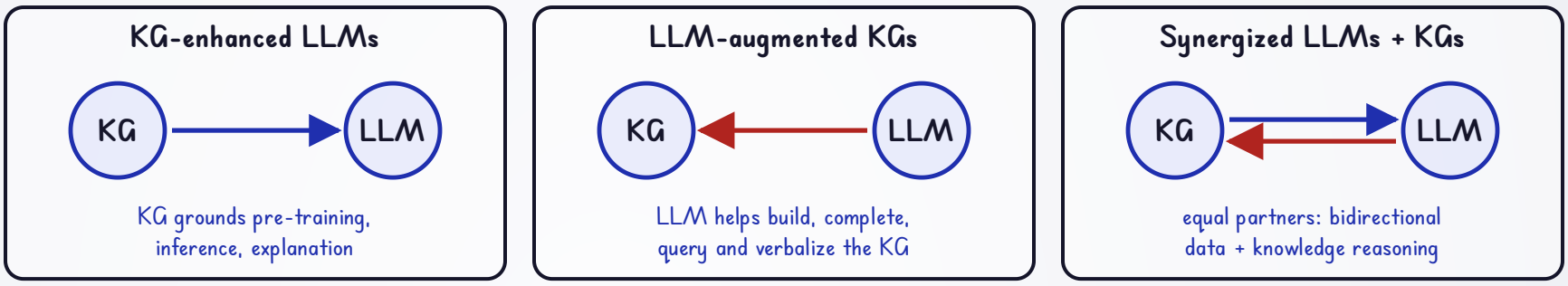
Key Takeaway: Physical choice (copy, federate, virtualize) is an operational decision; the semantic contract stays the same.

Sources: W3C SPARQL 1.1 Query & Federated Query (2013) · Calvanese et al., "Ontop", SWJ (2017) · ISO/IEC 39075:2024 GQL · W3C R2RML (2012)

57 KG + LLMs: GraphRAG

Part E · Knowledge Graphs

57.1 Three ways to combine LLMs and KGs (Pan et al. 2024)

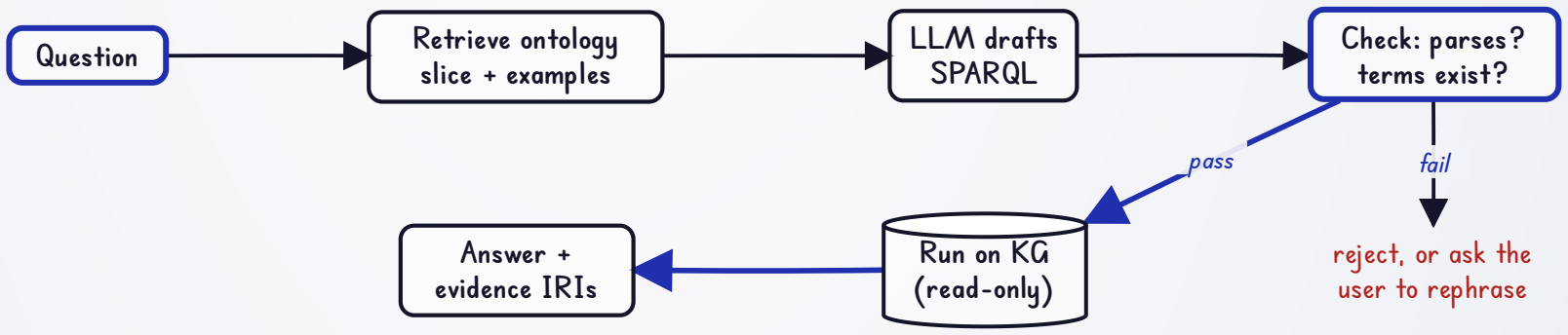


57.2 GraphRAG (Edge et al. 2024)



Edge et al. (arXiv:2404.16130) build an entity graph from text with an LLM, partition it with the **Leiden** algorithm into hierarchical communities, pre-summarize each community, and answer **global sensemaking** questions by map-reduce over those summaries. Vector RAG remains suited to specific fact lookups. Open source: microsoft/graphrag.

57.3 Ontology-grounded text-to-SPARQL



Sequeda, Allemang & Jacob (2023) report, on their insurance-domain benchmark, GPT-4 zero-shot accuracy of 16% on SQL vs 54% over a KG representation of the same data – evidence that explicit semantics help, not a guarantee for your data.

57.4 Guardrails for production

- Give the LLM **only the relevant ontology slice** (labels, definitions, domains/ranges).
- Reject queries using terms **not in the ontology**; parse before running.
- Run with a **read-only** account, timeouts and row limits.
- Return **evidence IRIs** and source graphs with every answer.
- Keep a CQ → expected-result test set; re-run on each model change.

57.5 Which pattern when?

Question type	Pattern
"What does clause 4.2 say?"	vector RAG over text
"Main themes across 10k notes?"	GraphRAG (global)
"Open claims on policy X?"	text-to-SPARQL on KG
"Why was this claim flagged?"	KG paths + provenance

Key Takeaway: For enterprise AI, the KG supplies identity and explicit relations; the ontology supplies the vocabulary that keeps the LLM's queries and answers on-model.

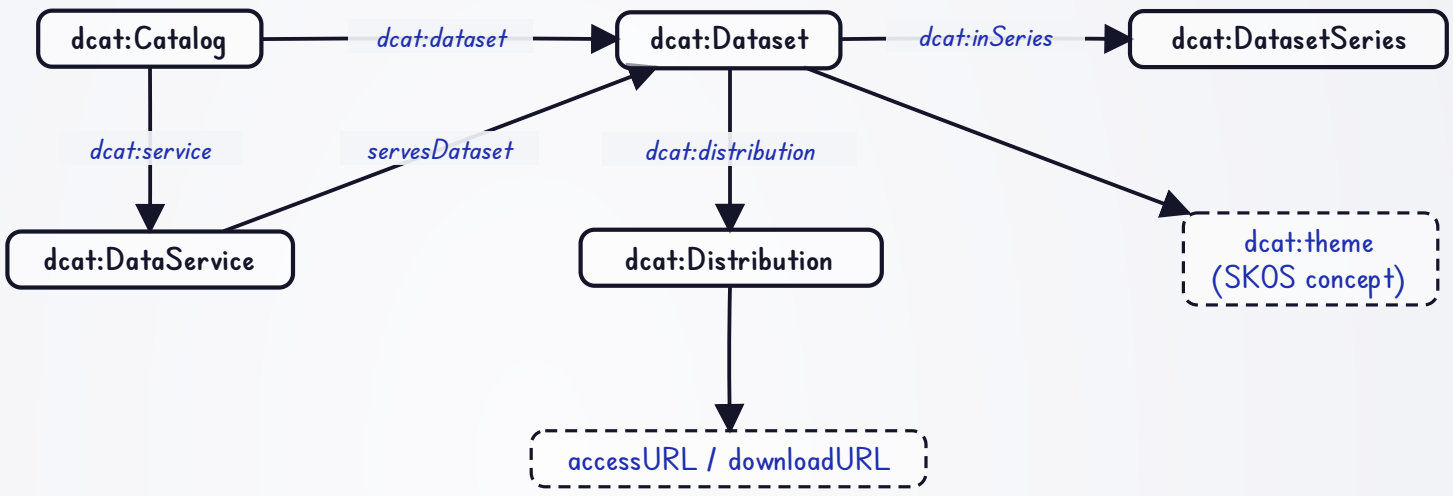
Sources: Edge et al., "From Local to Global: A Graph RAG Approach to Query-Focused Summarization", arXiv:2404.16130 (2024) · Pan et al., "Unifying LLMs and Knowledge Graphs: A Roadmap", IEEE TKDE (2024) · Sequeda, Allemang & Jacob, arXiv:2311.07509 (2023)

Rajesh Kumar Gupta

58 Catalogs & Data Products

Part E · Knowledge Graphs

58.1 DCAT 3 in one picture



DCAT version 3 became a W3C Recommendation on 22 August 2024; it added **dcat:DatasetSeries** and versioning terms and keeps DCAT 2 compatibility. Namespace: *http://www.w3.org/ns/dcat#*.

58.2 Catalog entry for the claims KG

```
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix dct:  <http://purl.org/dc/terms/> .
@prefix id:   <https://data.example.com/id/> .
@prefix g:    <https://ontology.example.com/graph/> .
id:catalog a dcat:Catalog ; dct:title "Enterprise KG catalog" ;
  dcat:dataset id:ds-claims .
id:ds-claims a dcat:Dataset ;
  dct:title "Claims knowledge graph"@en ;
  dct:publisher id:claims-domain ;
  dct:conformsTo <https://ontology.example.com/clm/> ;
  dcat:distribution [ a dcat:Distribution ;
    dcat:accessURL <https://kg.example.com/sparql> ;
    dcat:mediaType <https://www.iana.org/assignments/media-types/application/sparql-results+json> ] ;
  dct:source g:claims-2026-11-05 .
```

58.3 What to record per KG dataset

- Title, description, **publisher** (owning domain) and contact point.
- **dct:conformsTo** the ontology module and SHACL shapes it obeys.
- Distributions: SPARQL endpoint, API, dumps (media type).
- Themes from a SKOS scheme; keywords for search.
- Lineage: source graphs and PROV-O activities.

58.4 From dataset to data product

Data product needs	Where it lives
Discoverable	DCAT catalog entry
Understandable	ontology terms + definitions
Trustworthy	SHACL report, PROV-O lineage
Addressable	IRIs, endpoint URLs
Owned	dct:publisher = domain team

Key Takeaway: Describe the KG with the same technology it is built with: a DCAT catalog is itself RDF, so it can be queried, validated and linked to the ontology.

Sources: W3C Data Catalog Vocabulary (DCAT) Version 3, Recommendation (22 Aug 2024) · DCMI Metadata Terms · W3C PROV-O (2013) · Dehghani, Data Mesh, O'Reilly (2022)

59 Scale & Operations

Part E · Knowledge Graphs

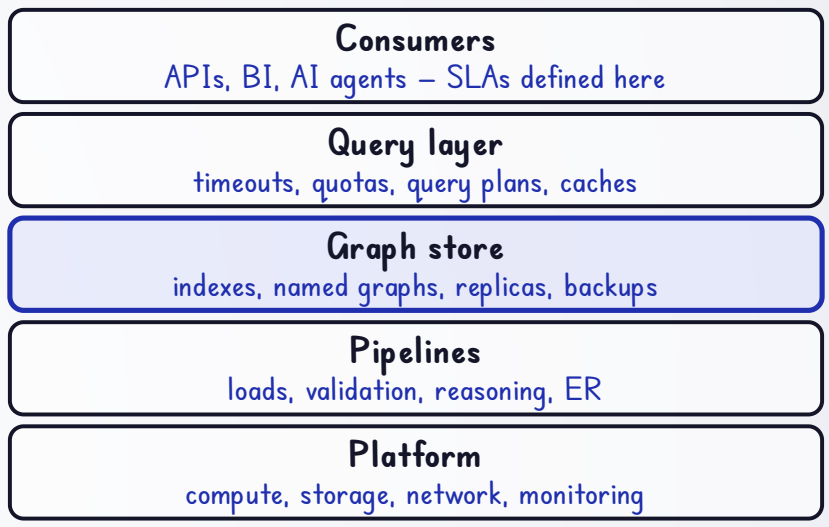
59.1 Where performance comes from

triple index permutations (RDF-3X keeps all six)

SPO | SOP | PSO | POS | OSP | OPS

many stores keep a subset, e.g. SPO / POS / OSP, plus G for quads

- **Indexes**: stores index triple/quad patterns so any bound position is a lookup (Neumann & Weikum 2008).
- **Selective patterns first**: bind IRIs or rare types early; avoid leading *?s ?p ?o*.
- **Property paths** (+, *) can explode; cap depth or precompute closures.



59.2 Cost levers and trade-offs

Lever	Helps	Costs
Materialized inference	read latency	load time, storage, re-run on change
Named-graph per batch	atomic swap, deletes, ACL	more graphs to manage
Partition / shard	scale-out writes	cross-partition joins
Read replicas	read throughput, HA	replication lag
Result caches	repeated API calls	staleness; invalidation

59.3 Operational SLOs to define

- Data **freshness**: source change → visible in KG.
- Query **latency** per consumer class (API vs analytics).
- **Availability** of endpoint and APIs.
- **Quality**: SHACL violations per load must be zero for blocking shapes.
- **Recovery**: backup restore time, tested.

59.4 Runbook checklist

1. Benchmark **your** queries on **your** data; vendor numbers do not transfer.
2. Log slow queries; review plans weekly.
3. Separate analytics from API traffic (replica or quota).
4. Load in a staging graph, validate, then swap.
5. Watch growth of inferred and provenance graphs.

Key Takeaway: Set SLAs per consumer, measure them on your own workload, and choose materialization, partitioning and caching to meet them; there is no universal benchmark answer.

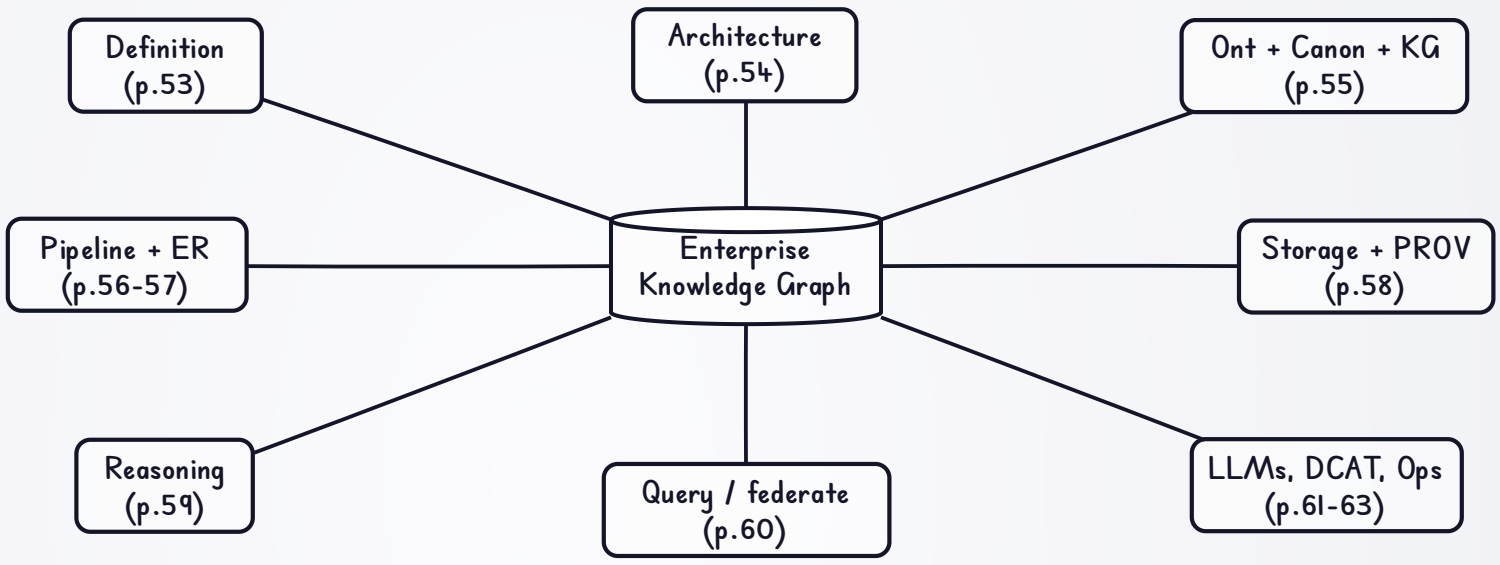
Sources: Neumann & Weikum, "RDF-3X: a RISC-style Engine for RDF", PVLDB (2008) · W3C SPARQL 1.1 Query (2013) · Kleppmann, Designing Data-Intensive Applications, O'Reilly (2017)

Rajesh Kumar Gupta

60 Part E Recap

Part E · Knowledge Graphs

60.1 Part E on one page



60.2 Standards cheat-sheet (status checked Nov 2026)

Standard	Role in the KG	Status
RDF 1.1 / SPARQL 1.1	data model, query, federation	W3C Rec (2014 / 2013)
RDF 1.2 / SPARQL 1.2	triple terms, new features	RDF 1.2 Concepts: Candidate Rec (2026); check w3.org
R2RML / RML	source → RDF mappings	R2RML W3C Rec (2012); RML community spec
OWL 2 RL, RDFS	rule-based inference	W3C Rec (2012 / 2014)
SHACL / SHACL-AF	validation / rules	Rec (2017) / WG Note (2017); 1.2 drafts in 2026
PROV-O	lineage & provenance	W3C Rec (2013)
DCAT 3	catalog of KG datasets	W3C Rec (Aug 2024)
ISO GQL	property-graph query	ISO/IEC 39075:2024

60.3 Architect's KG checklist

1. Ontology terms on every node and edge; no local classes.
2. Enterprise IRIs + resolved identities with source links.
3. Named graphs per source/batch; PROV-O for each load.
4. SHACL gate before publish; reports kept.
5. Inference in its own graph; rules tested by CQs.

60.4 Checklist, continued

1. Physical pattern (copy / federate / virtual) chosen per source.
2. DCAT entry with owner, conformsTo, distributions.
3. LLM access only through grounded, validated queries.
4. SLOs for freshness, latency, quality, recovery.
5. Everything (mappings, shapes, rules) in Git + CI.

Key Takeaway: Ontology gives meaning, canonical model gives shape, the KG gives connected, traceable facts. Part F shows how departments and APIs consume them.

Sources: Hogan et al., Knowledge Graphs, ACM CSUR (2021) · W3C Standards & drafts index, w3.org/TR (checked Nov 2026) · ISO/IEC 39075:2024

Rajesh Kumar Gupta

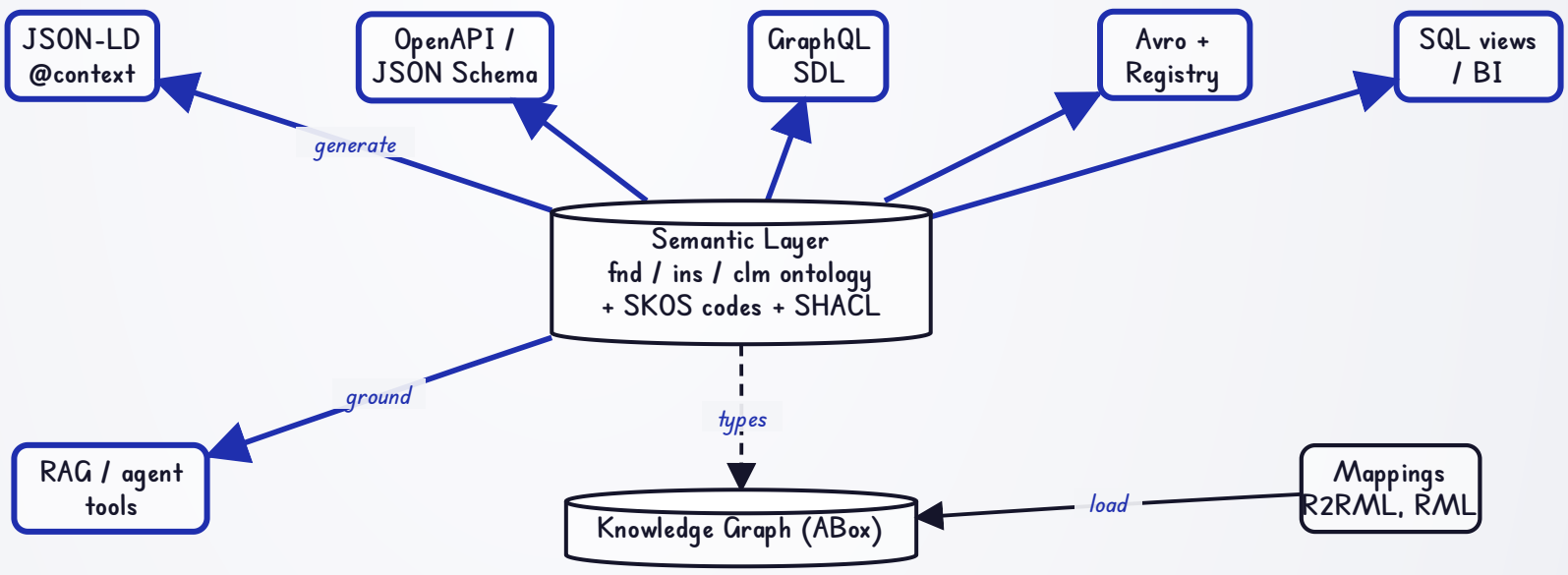
61 The Semantic Layer Concept

Part F · Consuming the Ontology: APIs & Departments

61.1 The problem it solves

Without it, every department publishes its own "claim" JSON: *status* means one thing in Claims, another in Finance, and the mapping lives in someone's head. A **semantic layer** (working definition used in this notebook) is the governed set of shared meanings — ontology classes and properties, SKOS code lists, SHACL shapes and source mappings — exposed through many delivery formats, each one traceable to the same IRIs.

61.2 One meaning, many delivery formats



61.3 The layer cake

Consumers
department apps • partners • analysts • AI agents

Delivery formats
REST • GraphQL • events • SQL • files

Semantic layer
one IRI per concept • code lists • shapes • mappings

Data
knowledge graph • source systems • lake / warehouse

61.4 Rules that make it work

1. **One IRI, one meaning** : every field in every format points back to a single ontology term.
2. **Generate, don't copy** : schemas are build outputs of the model, never hand-retyped.
3. **Meaning travels** : IRIs ride inside payloads (JSON-LD, x- tags, Avro attributes).
4. **Format is the consumer's choice** : REST, events or SQL; the meaning is the same.

61.5 Monday morning

- Pick one high-traffic concept (e.g. **clm:Claim**) and list every API, topic and table that carries it.
- Publish its IRI, definition and code list; add the IRI to each existing schema as an annotation — no breaking change needed.
- Wire one generator (Topic 63) so the next schema change starts from the model, not from a copy.

Key Takeaway: the semantic layer is not a product you buy; it is the discipline of delivering one model in the format each consumer already uses.

Sources: W3C JSON-LD 1.1 (Rec. 2020) · Hogan et al., Knowledge Graphs, ACM Computing Surveys (2021) · W3C R2RML (Rec. 2012)

Rajesh Kumar Gupta

⑥② JSON-LD: Ontology Terms in API JSON

Part F · Consuming the Ontology: APIs & Departments

62.1 Why JSON-LD for APIs

JSON-LD 1.1 (W3C Recommendation, 16 July 2020) lets an API keep ordinary-looking JSON while a **@context** binds each key to an ontology IRI. Developers see *"status": "OPEN"*; the graph sees *clm:hasClaimStatus clm:Open*. The example below was expanded with pyld to check it.

62.2 Context + payload

```
{ "@context": {
    "@version": 1.1, "@protected": true,
    "clm": "https://ontology.example.com/clm/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "id": "@id", "type": "@type",
    "Claim": "clm:Claim",
    "claimNumber": "clm:claimNumber",
    "onPolicy": {"@id": "clm:onPolicy",
                 "@type": "@id"},
    "status": {"@id": "clm:hasClaimStatus",
               "@type": "@vocab"},
    "OPEN": "clm:Open",
    "reportedOn": {"@id": "clm:reportedOn",
                   "@type": "xsd:date"} },
  "id": "https://data.example.com/id/claim/CLM-1001",
  "type": "Claim",
  "claimNumber": "CLM-1001",
  "onPolicy": "https://data.example.com/id/policy/POL-77",
  "status": "OPEN",
  "reportedOn": "2026-03-14" }
```

62.3 What a machine sees

```
# expanded with pyld -> N-Quads (prefixes
# shortened here for space)
<.../claim/CLM-1001> rdf:type clm:Claim .
<.../claim/CLM-1001> clm:claimNumber
     "CLM-1001" .
<.../claim/CLM-1001> clm:hasClaimStatus
     clm:Open .
<.../claim/CLM-1001> clm:onPolicy
     <.../policy/POL-77> .
<.../claim/CLM-1001> clm:reportedOn
     "2026-03-14"^^xsd:date .
```





Plain JSON clients ignore **@context**; RDF-aware clients get triples for free.

62.4 Keywords you will use

Keyword	What it does in an API
@context	maps JSON keys to ontology IRIs
@id / @type	node IRI / class (aliased to id, type)
"@type": "@id"	value is an IRI link, not a string
"@type": "@vocab"	value is a term (code list member)
@protected	later contexts cannot redefine terms
@vocab / @base	default prefix for keys / relative IRIs

62.5 Delivery rules

- Publish the context at a stable, **versioned** URL; generate it (LinkML *gen-jsonld-context*).
- For plain *application/json* responses, point to it with an HTTP Link header, rel *http://www.w3.org/ns/json-ld#context*.
- Clients should cache contexts — never fetch them on every call.
- Type dates (*xsd:date*) and link fields (*@id*) or they arrive as untyped strings.

Key Takeaway: a JSON-LD context is the cheapest bridge from ontology to API: zero change for JSON clients, full meaning for graph clients.

Sources: W3C JSON-LD 1.1 (Rec. 16 Jul 2020) · W3C JSON-LD 1.1 Processing Algorithms and API (2020) · pyld library (JSON-LD processor for Python) · LinkML docs: JSON-LD context generator

Rajesh Kumar Gupta

63 Generating OpenAPI & JSON Schema

Part F · Consuming the Ontology: APIs & Departments

63.1 The pipeline (one source, many schemas)



The ontology says what things *mean*; a canonical/schema layer (here LinkML) says what an API must *carry*. Generators turn that into JSON Schema, then OpenAPI wraps it. The snippets below are real outputs from our mini claims schema.

63.2 Canonical source

```yaml
# LinkML (excerpt of clm_api.yaml)
classes:
  Claim:
    class_uri: clm:Claim
    attributes:
      claimNumber:
        slot_uri: clm:claimNumber
        required: true
      status:
        slot_uri: clm:hasClaimStatus
        range: ClaimStatus
        required: true
```

63.3 Generated schema

```json
# gen-json-schema output (excerpt)
"Claim": {
 "type": "object",
 "additionalProperties": false,
 "properties": {
  "claimNumber": {"type": "string"},
  "reportedOn": {"format": "date",
    "type": ["string", "null"]},
  "status": {"$ref": "#/$defs/ClaimStatus"} },
 "required": ["id", "claimNumber",
            "onPolicy", "status"] }
```

63.4 OpenAPI with ontology tags

```yaml
openapi: 3.1.0
info: {title: Claims API, version: 2.1.0}
components:
  schemas:
    Claim:
      type: object
      x-ontology-class: https://ontology.example.com/clm/Claim
      required: [claimNumber, status]
      properties:
        claimNumber:
          type: string
          x-ontology-property: https://ontology.example.com/clm/claimNumber
        status:
          type: string
          enum: [OPEN, CLOSED]
          x-ontology-property: https://ontology.example.com/clm/hasClaimStatus
# paths omitted here; full file passed openapi-spec-validator
```

63.5 What maps, what is lost

Model construct	JSON Schema / OpenAPI
sh:minCount 1	*required*
sh:maxCount 1 / none	scalar / *array*
SKOS scheme, sh:in	*enum* (keep IRIs in x- tag)
xsd:date, xsd:decimal	*format: date*, string/number
sh:pattern	*pattern*
object property	IRI string or *$ref*
subClassOf, disjoint	**lost**: keep the IRI

63.6 Gotchas

- OpenAPI 3.1 aligns its Schema Object with JSON Schema 2020-12; OAS 3.2.0 (Sept 2025) keeps that. Pin the dialect (*jsonSchemaDialect* or *$schema*) — our LinkML run emitted draft 2019-09.
- Extensions must start with **x-**; tools ignore unknown ones, so they are safe to add today.
- Generated files are read-only: fix the model, re-generate, never hand-edit.
- Run a breaking-change diff against the last published spec in CI (Topic 70).

Key Takeaway: OpenAPI carries structure, not meaning: the x-ontology tags are what let a reviewer, a linter or an LLM trace each field back to the ontology.

Sources: OpenAPI Initiative, OpenAPI Specification 3.1.0 (2021) & 3.2.0 (2025) · JSON Schema 2020-12 · LinkML docs: generators (gen-json-schema)

64 GraphQL over the Knowledge Graph

Part F · Consuming the Ontology: APIs & Departments

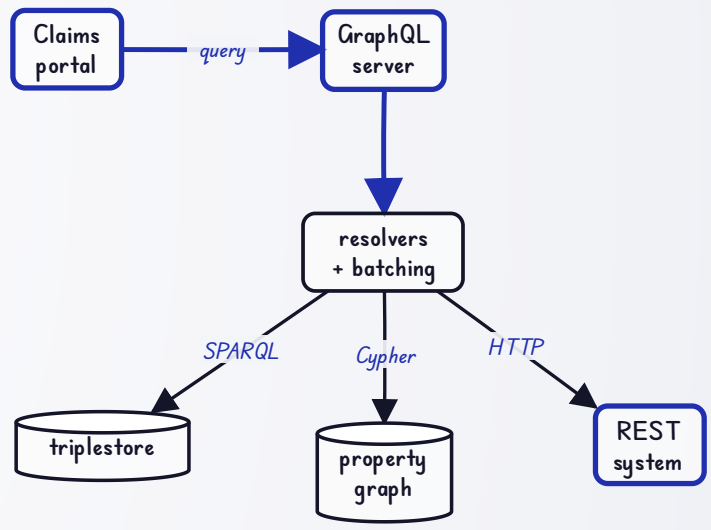
64.1 Why GraphQL fits a KG

A GraphQL client asks for exactly the fields it needs, against a typed schema. Knowledge graphs are already graph-shaped, so the ontology is a natural source for that schema. Current spec: the **September 2025 edition** (first since October 2021; adds OneOf input objects and schema coordinates).

64.2 Generated SDL (validated)

```
"clm:Claim - a request for payment under a policy"
type Claim {
  id: ID!                # the claim IRI
  claimNumber: String!
  status: ClaimStatus!    # SKOS scheme -> enum
  reportedOn: String
  policy: Policy!         # object property
  lines(first: Int = 20): [ClaimLine!]!
}
enum ClaimStatus { OPEN CLOSED }
type Policy { id: ID!  policyNumber: String! }
type ClaimLine { id: ID!  amount: Float }
type Query {
  claim(id: ID!): Claim
  claimsByPolicy(policyId: ID!): [Claim!]!
}
```

64.3 Resolver architecture




```
# resolver for claimsByPolicy -> SPARQL
PREFIX clm: <https://ontology.example.com/clm/>
PREFIX pol: <https://data.example.com/id/policy/>
SELECT ?claim ?number ?status WHERE {
  VALUES ?policy { pol:POL-77 }
  ?claim a clm:Claim ;
         clm:onPolicy ?policy ;
         clm:claimNumber ?number ;
         clm:hasClaimStatus ?status .
} LIMIT 20
```

64.4 Mapping rules

Ontology	GraphQL SDL
owl:Class	object *type*
datatype property	scalar field
object property	field typed by another type
SKOS concept scheme	*enum*
superclass shared by many	*interface*
individual IRI	*ID*
owl:deprecated	*@deprecated(reason:)*

64.5 Tooling (check versions)

- **LinkML gen-graphql** emits SDL from the canonical model.
- **Neo4j GraphQL Library** generates resolvers over Neo4j.
- **HyperGraphQL** (open source) serves GraphQL over SPARQL endpoints.
- Several triplestores ship GraphQL endpoints — see vendor docs.

64.6 Production cautions

- Set query **depth / complexity limits**; one nested query can explode into thousands of graph lookups.
- Batch per-field lookups (the DataLoader pattern) to avoid the N+1 problem.
- Keep the IRI in the description or an *id* field so GraphQL types stay traceable to the ontology.

Key Takeaway: GraphQL is a projection of the ontology for app developers — generate the SDL, keep inference and governance in the graph.

Sources: GraphQL Foundation, GraphQL Specification, Sept 2025 edition · W3C SPARQL 1.1 Query Language (2013) · LinkML docs: GraphQL generator · Neo4j GraphQL Library docs; HyperGraphQL (GitHub)

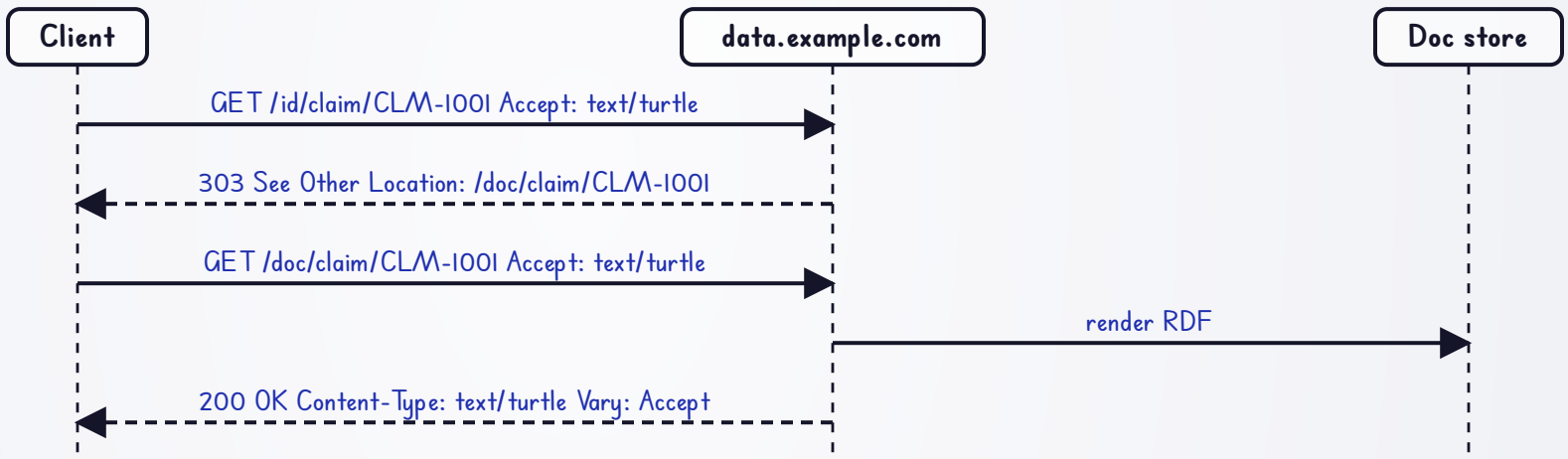
65 Linked-Data REST & Content Negotiation

Part F · Consuming the Ontology: APIs & Departments

65.1 Thing IRI vs document URL

Cool URIs for the Semantic Web (W3C Interest Group Note, 2008) separates the **thing** (a claim) from the **document** that describes it. Two patterns: **hash IRIs** (*.../clm#Claim*, good for small vocabularies) and **303 redirects** (good for large data sets, one doc per thing).

65.2 Dereferencing a claim (303 pattern)

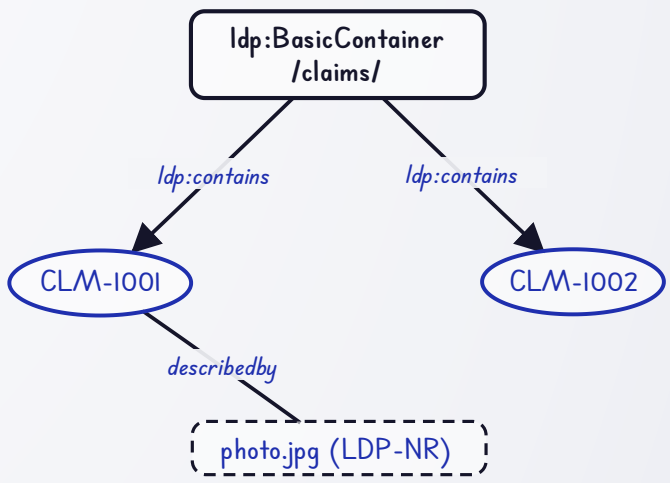


65.3 Content negotiation

Accept header	Returned representation
text/html	human page (steward view)
text/turtle	RDF triples (LDP default)
application/ld+json	JSON-LD with @context
application/json	plain JSON + Link to context

Use **Vary: Accept** so caches keep variants apart, and *Content-Location* to name the variant returned.

65.4 W3C LDP 1.0 in one picture



LDP (Rec. 2015): RDF Sources and Non-RDF Sources; Basic, Direct and Indirect containers. Servers must offer *text/turtle* and *application/ld+json*, send ETags and honour *If-Match*.

65.5 Design rules for linked-data REST

1. **Split namespaces** : *ontology.example.com* for terms, *data.example.com/id/* for things, */doc/* for documents.
2. **No technology in IRIs** : no *.jsp*, no system names, no version in a thing IRI — it must outlive the platform.
3. **Every IRI resolves** : even deprecated ones return a page saying what replaced them.
4. **Links, not foreign keys** : return *"onPolicy": "https://data.example.com/id/policy/POL-77"* instead of a bare number.

Key Takeaway: if an IRI cannot be looked up, it is just a long string; dereferenceable IRIs turn the ontology into a browsable API.

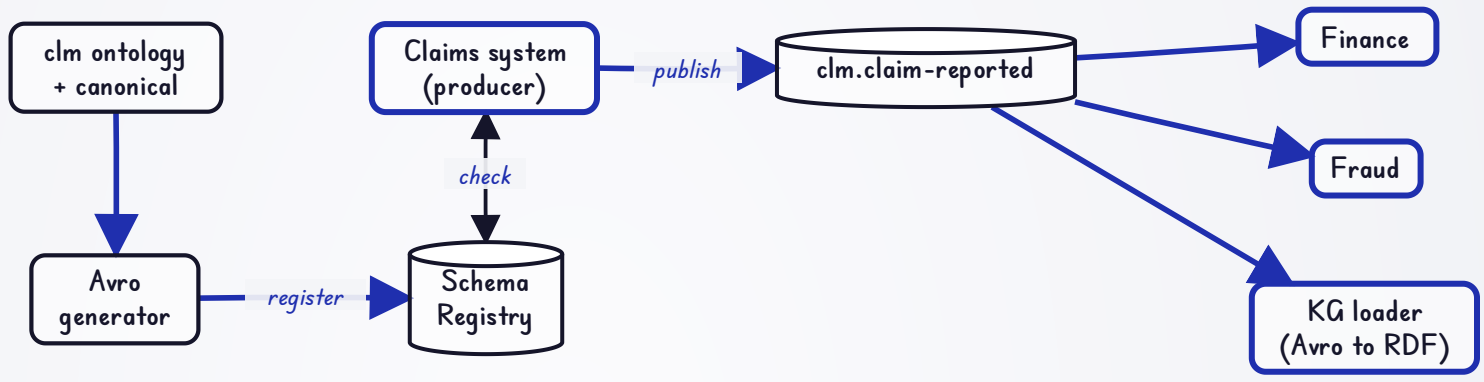
Sources: W3C Cool URIs for the Semantic Web, IG Note (2008) · W3C Linked Data Platform 1.0 (Rec. 2015) · IETF RFC 9110 HTTP Semantics (2022)

Rajesh Kumar Gupta

66 Event-Driven Consumption

Part F · Consuming the Ontology: APIs & Departments

66.1 Events carry canonical meaning



66.2 Avro generated from canonical

```
{"type": "record", "name": "ClaimReported",
 "namespace": "com.example.clm.events",
 "doc": "Canonical event: clm:Claim was reported",
 "iri": "https://ontology.example.com/clm/Claim",
 "fields": [
  {"name": "claimNumber", "type": "string",
   "iri": "https://ontology.example.com/clm/claimNumber"},
  {"name": "status", "type": {"type": "enum",
     "name": "ClaimStatus", "symbols": ["OPEN",
"CLOSED"]}},
  {"name": "reportedOn",
   "type": {"type": "int", "logicalType": "date"}},
  {"name": "channel", "type": ["null", "string"],
   "default": null} ]}
```

"iri" is a custom attribute: the Avro spec permits unknown attributes as metadata as long as they do not change the encoding. Parsed with fastavro to check.

66.3 Safe changes (Avro)

- Add a field **with a default** → readers on the new schema fill it in for old events.
- Rename via *aliases*, never by delete + add.
- Code list grows (new *ClaimStatus*)? Adding enum symbols can break old readers – give the enum a *default* symbol or ship as a string with the SKOS IRI.

66.4 Schema Registry compatibility modes

Mode	Guarantee	Checked vs	Upgrade first
BACKWARD **(default)**	new readers read data of X-1	latest	consumers
BACKWARD_TRANSITIVE	new readers read all older	all	consumers
FORWARD	old readers read new data	latest	producers
FORWARD_TRANSITIVE	all old readers read new	all	producers
FULL	both ways with X-1	latest	any order
FULL_TRANSITIVE	both ways with all	all	any order
NONE	no checks	-	coordinate!

66.5 Working rules

- Name events in business terms from the ontology (*ClaimReported*), not table names.
- Subjects default to **TopicNameStrategy** (*<topic>-value*); use RecordNameStrategy or TopicRecordNameStrategy when one topic carries several event types.
- Long-retention or replayable topics: prefer a **_TRANSITIVE** mode so any old event stays readable.
- Turn off producer auto-registration in production; register only from the CI pipeline.

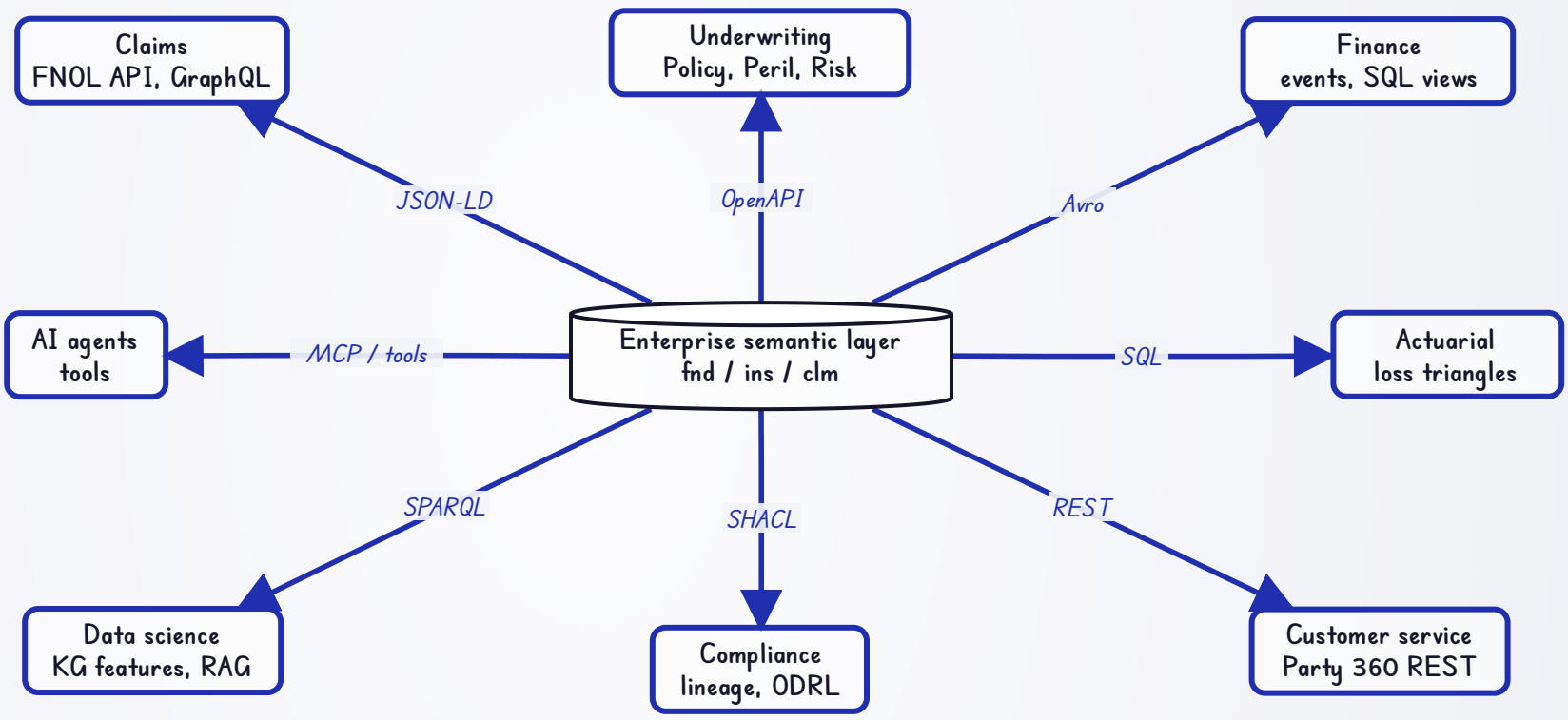
Key Takeaway: the Schema Registry enforces structure; the iri attributes and the ontology version in the event header keep its meaning.

Sources: Confluent docs: Schema Evolution and Compatibility; Schema subjects · Apache Avro Specification 1.12.0 · Hohpe & Woolf, Enterprise Integration Patterns (2003)

67 How Each Department Consumes the Ontology

Part F · Consuming the Ontology: APIs & Departments

67.1 One layer, many consumers (illustrative)



Edge labels show the **typical** delivery format each team prefers; any team can use any channel. The point is that every channel is generated from, and traceable to, the same model.

67.2 Who consumes what

Department	Concepts it reuses	Usual delivery	What it gains
Claims	Claim, ClaimLine, LossEvent, ClaimStatus	REST + JSON-LD, events	one status list across systems
Underwriting	Policy, Coverage, Peril, Insured	OpenAPI, GraphQL	consistent coverage terms
Finance	Payment, Reserve, Money	Avro events, SQL	reconcilable amounts & currency
Actuarial	LossEvent, Reserve, Peril	SQL views / files	stable definitions over years
Customer service	Party, Agreement, Claim	REST (Party 360)	one customer view
Compliance	Document, PII tags, provenance	SPARQL, reports	traceable lineage & policies
Data science / AI	whole graph + definitions	SPARQL, embeddings, tools	grounded features & answers

67.3 Onboarding a department

1. List its top questions → competency questions.
2. Map its fields to existing terms first; propose new ones only for gaps.
3. Choose its channel; generate the schema; add it to CI.

67.4 Signals of real reuse

- Count of schemas carrying *x-ontology* / *iri* tags (measurable in CI).
- Number of local code lists retired in favour of SKOS schemes.
- Cross-department joins done on IRIs, not spreadsheets.

Key Takeaway: departments do not consume "the ontology"; they consume generated, format-specific slices of it — design the slices per team.

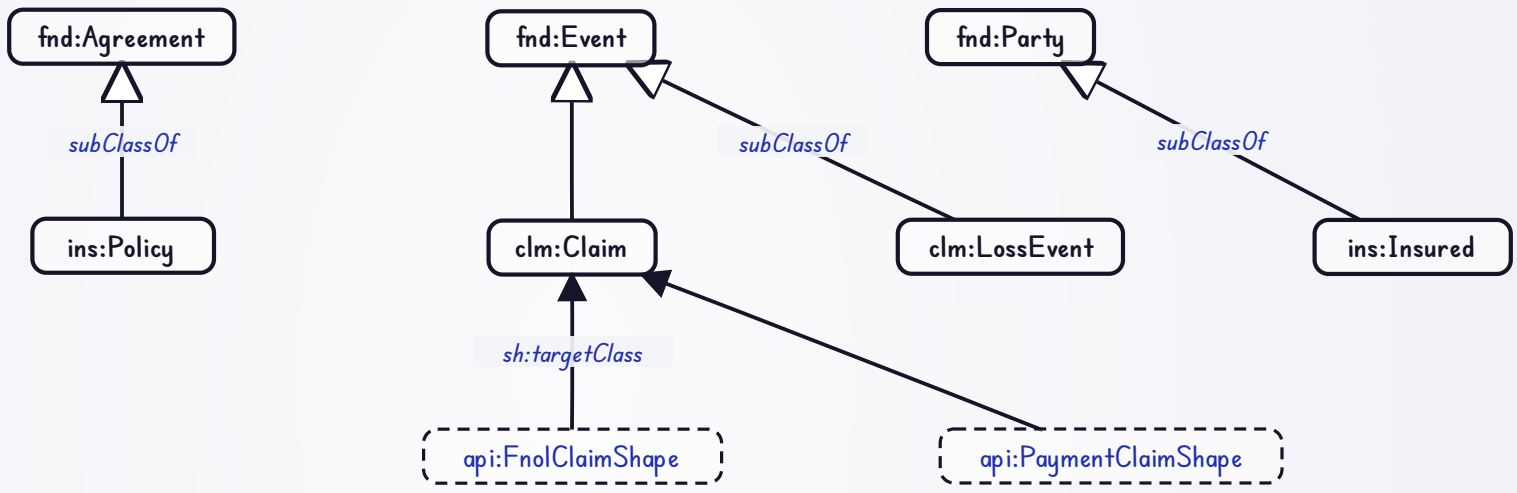
Sources: Hohpe & Woolf, Enterprise Integration Patterns (2003) – Canonical Data Model · W3C SKOS Reference (Rec. 2009) · Dehghani, Data Mesh (O'Reilly, 2022)

Rajesh Kumar Gupta

⑥⑧ Reuse: Enterprise → Domain → API

Part F · Consuming the Ontology: APIs & Departments

68.1 Three levels, one lineage



Domain classes **specialise** enterprise classes (*rdfs:subClassOf*); API shapes **narrow** domain classes for one use case. Nothing is copied, so a query for *fnd:Event* still finds every claim and loss event.

68.2 SHACL profile per API (tested with pySHACL)

```
@prefix sh:  <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix clm: <https://ontology.example.com/clm/> .
@prefix api: <https://ontology.example.com/api/fnol/> .
# FNOL profile: narrower than the enterprise shape
api:FnolClaimShape a sh:NodeShape ;
  sh:targetClass clm:Claim ;
  sh:property [ sh:path clm:claimNumber ;
    sh:minCount 1 ; sh:maxCount 1 ;
    sh:pattern "^CLM-[0-9]{4,}$" ] ;
  sh:property [ sh:path clm:hasClaimStatus ;
    sh:minCount 1 ; sh:in ( clm:Open ) ] ;
  sh:property [ sh:path clm:reportedOn ;
    sh:datatype xsd:date ; sh:minCount 1 ] .
```

A claim numbered "X9" with status *closed* and no date produced 3 violations: pattern, sh:in, minCount.

68.3 Declare the profile (W3C PROF)

```
@prefix prof: <http://www.w3.org/ns/dx/prof/> .
@prefix role: <http://www.w3.org/ns/dx/prof/role/> .
@prefix api: <https://ontology.example.com/api/fnol/> .
api:profile a prof:Profile ;
  prof:isProfileOf
    <https://ontology.example.com/clm/> ;
  prof:hasResource [
    a prof:ResourceDescriptor ;
    prof:hasRole role:validation ;
    prof:hasArtifact
      <https://ontology.example.com/api/fnol/shapes.ttl> ] .
```

PROF (W3C WG Note, 2019) records *what profiles what* and where its shapes live.

68.4 Reuse decision rules

1. **Same meaning?** : reuse the IRI as-is.
2. **Narrower meaning?** : subclass / subproperty in the domain module.
3. **Same meaning, stricter data?** : SHACL profile, not a new class.
4. **Different meaning?** : new term + skos:closeMatch note, never overload.

68.5 Field-level semantic tags

- JSON Schema / OpenAPI: *x-ontology-property*.
- Avro: custom *iri* attribute.
- GraphQL: IRI in the type / field description.
- SQL: column comment holding the IRI.
- JSON-LD: the @context itself.

Key Takeaway: a profile may only tighten the enterprise contract (stricter cardinality, smaller code list) — never redefine a term.

Sources: W3C SHACL (Rec. 2017) · W3C The Profiles Vocabulary (PROF), WG Note (2019) · W3C RDF Schema 1.1 (2014)

69 Versioning & Backward Compatibility

Part F · Consuming the Ontology: APIs & Departments

69.1 Semantic Versioning applied to meaning

SemVer 2.0.0: MAJOR for incompatible changes, MINOR for backward-compatible additions, PATCH for fixes. For an ontology the "API" is its terms, their meaning and the shapes data must satisfy.

Change to the ontology / canonical	Bump	Why
fix label, definition typo, add skos:example	PATCH	no data or schema effect
add class, optional property, new code	MINOR	additive; old data still valid*
deprecate a term (keep it working)	MINOR	signal only, nothing removed
remove / rename term, change domain or range	MAJOR	existing data or queries break
tighten a shape (optional → required)	MAJOR	old payloads now fail validation
change the meaning of a definition	MAJOR	silent breakage: worst kind

** "additive" is only safe if consumers ignore unknown fields/codes — state this in the contract (Topic 70).*

69.2 Deprecation lifecycle



69.3 OWL 2 version metadata (parsed with rdflib)

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix clm: <https://ontology.example.com/clm/> .
<https://ontology.example.com/clm/> a owl:Ontology ;
  owl:versionIRI <https://ontology.example.com/clm/2.1.0/> ;
  owl:priorVersion <https://ontology.example.com/clm/2.0.0/> ;
  owl:backwardCompatibleWith
      <https://ontology.example.com/clm/2.0.0/> ;
  owl:versionInfo "2.1.0" .
clm:lossDate a owl:DatatypeProperty ;
  owl:deprecated true ;
  dcterms:isReplacedBy clm:lossOccurredOn ;
  skos:editorialNote "Deprecated 2.1.0; removal planned 3.0.0." .
```

69.4 Same version, every format

Artifact	Where the version shows
Ontology	*owl:versionIRI, owl:versionInfo*
JSON-LD	versioned context URL
OpenAPI	*info.version; deprecated: true*
GraphQL	*@deprecated(reason: ...)*
Avro	registry subject version

The ontology IRI means "latest"; the *versionIRI* pins one release.

```
# notify consumers: what is deprecated?
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dcterms: <http://purl.org/dc/terms/>
SELECT ?old ?new WHERE {
  ?old owl:deprecated true .
  OPTIONAL { ?old dcterms:isReplacedBy ?new } }
```

- **Never reuse** a retired IRI for a new meaning.
- Deprecated IRIs keep resolving and point to the replacement.

Key Takeaway: version the meaning, not just the file: a changed definition with an unchanged IRI is a breaking change nobody sees.

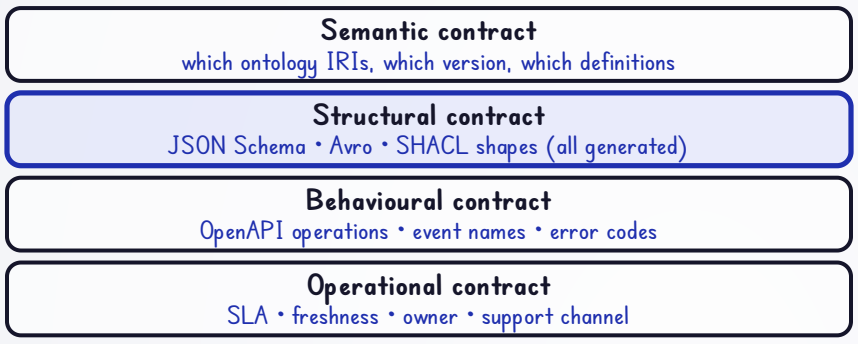
Sources: Preston-Werner, Semantic Versioning 2.0.0 (semver.org) · W3C OWL 2 Structural Specification (2nd ed., 2012) · DCMI Metadata Terms (dcterms:isReplacedBy) · GraphQL Specification, Sept 2025 edition

Rajesh Kumar Gupta

70 Contracts & API Governance

Part F · Consuming the Ontology: APIs & Departments

70.1 Four layers of an API contract



An API contract is more than a schema. The **semantic** layer is the part most API programs skip — and the one the ontology supplies. The structural layer should be **generated** from it so the two cannot drift.

- SHACL validates data entering the KG.
- JSON Schema / Avro validate data at the API or topic edge.
- Same source model ⇒ same rules at both ends.

70.2 Consumer-driven contracts

Robinson (2006) describes **consumer-driven contracts**: each consumer states the parts of a provider's contract it actually relies on, and the provider runs all those expectations as tests before releasing. Pact is a widely used open-source tool for this style.

Who writes the test	Approach	Best for
Provider	provider contract (OpenAPI + SHACL)	many unknown consumers
Each consumer	consumer-driven contracts (e.g. Pact)	few, known consumers
Platform	linting + registry gates	enterprise-wide rules

70.3 Lint for semantics

```
# .spectral.yaml (Spectral ruleset)
extends: spectral:oas
rules:
  property-has-ontology-iri:
    description: Every schema property
      cites an ontology IRI
    given: $.components.schemas[*].properties[*]
    severity: error
    then:
      field: x-ontology-property
      function: truthy
```

Spectral rules use a JSONPath *given* and a *then* function; *truthy* fails when the field is missing.

70.4 Governance checklist

- Every property cites an IRI (lint).
- Enums come from SKOS schemes, not hand lists.
- Breaking diff ⇒ MAJOR bump + design-authority sign-off.
- Owner and support channel in *info.contact*.
- Deprecations announced with replacement + date.
- Unknown fields tolerated by consumers (stated).

70.5 The CI gate



Key Takeaway: make the contract executable: if a rule is not checked in CI, it is a guideline, not governance.

Sources: Robinson, Consumer-Driven Contracts: A Service Evolution Pattern, martinfowler.com (2006) · Stoplight Spectral docs: rulesets · W3C SHACL (Rec. 2017) · OpenAPI Specification 3.1.0

Rajesh Kumar Gupta

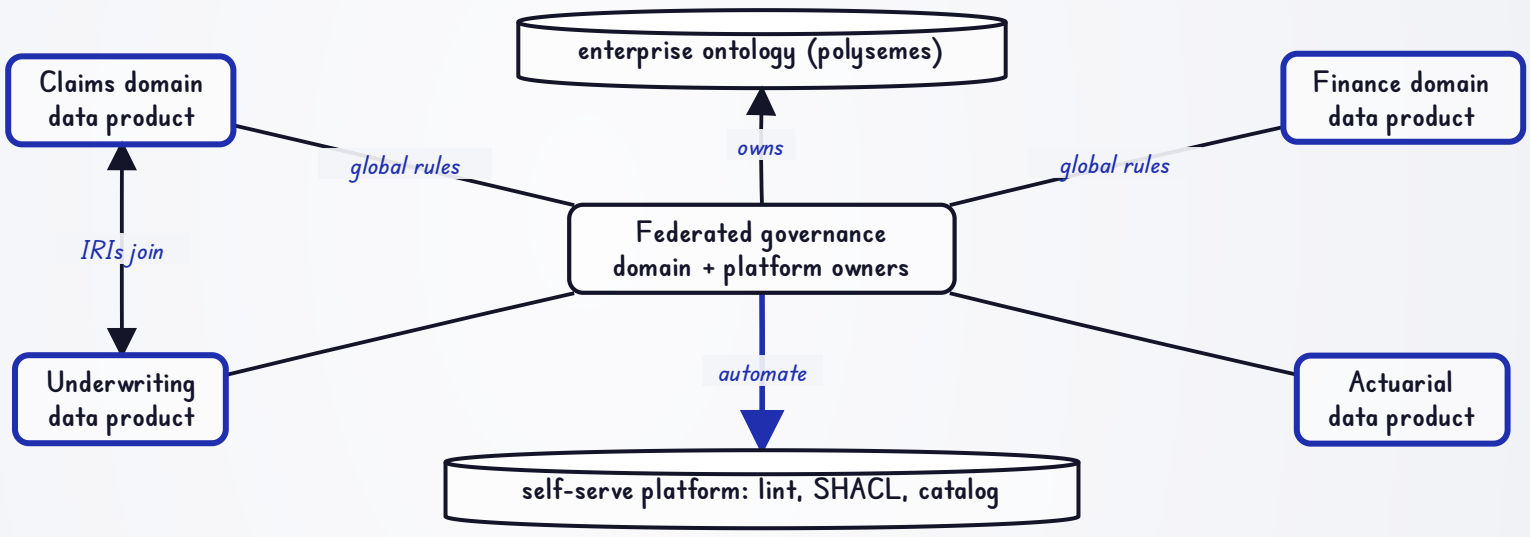
⑦① Data Mesh & Federated Governance

Part F · Consuming the Ontology: APIs & Departments

71.1 Data mesh in four principles

Dehghani (2020 article; 2022 book) names four principles: **domain-oriented ownership**, **data as a product**, **self-serve data platform** and **federated computational governance**. The last one is where the enterprise ontology lives.

71.2 Where the ontology sits



In her 2020 article, governance is a federation of domain and platform owners: local autonomy inside a small set of **global rules** for interoperability, executed **automatically by the platform**. She lists modelling **polysemes** – data elements that cross domain boundaries (her example: a podcast listener) – as a governance concern. That is exactly the job of the *fnd* layer.

71.3 What is global vs local

Global (federated) decision	Local (domain) decision
IRIs & meaning of shared concepts (Party, Policy, Claim)	domain-only classes and properties
identifier policy, code-list governance	internal storage & pipeline tech
metadata a product must publish (DCAT, owner)	release cadence of its product
security / PII classification scheme	which consumers it onboards first

71.4 A data product declares its semantics

```
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ex: <https://data.example.com/product/> .
ex:open-claims a dcat:Dataset ;
  dcterms:title "Open claims (daily)" ;
  dcterms:publisher <https://data.example.com/team/claims> ;
  dcterms:conformsTo
<https://ontology.example.com/clm/2.1.0/> ,
       <https://ontology.example.com/api/fnol/profile> ;
  dcat:distribution [ a dcat:Distribution ;
    dcat:accessURL <https://data.example.com/sparql> ;
    dcat:mediaType
<https://www.iana.org/assignments/media-types/text/turtle>
] .
```

71.5 Making it "computational"

- Platform CI rejects a product whose schema lacks IRIs or whose data fails the shared SHACL shapes.
- Catalog entries (DCAT) cite the ontology version via *dcterms:conformsTo*.
- Keep the global set small: too many global rules recreate the central bottleneck mesh tries to remove.

Key Takeaway: data mesh decentralises ownership, not meaning: the shared ontology is the "global standard" that lets independent products join.

Sources: Dehghani, Data Mesh: Delivering Data-Driven Value at Scale (O'Reilly, 2022) · Dehghani, Data Mesh Principles and Logical Architecture, martinfowler.com (2020) · W3C DCAT v3 (Rec. 2024)

Rajesh Kumar Gupta

72 Security & Access

Part F · Consuming the Ontology: APIs & Departments

72.1 Tag sensitivity in the ontology, once

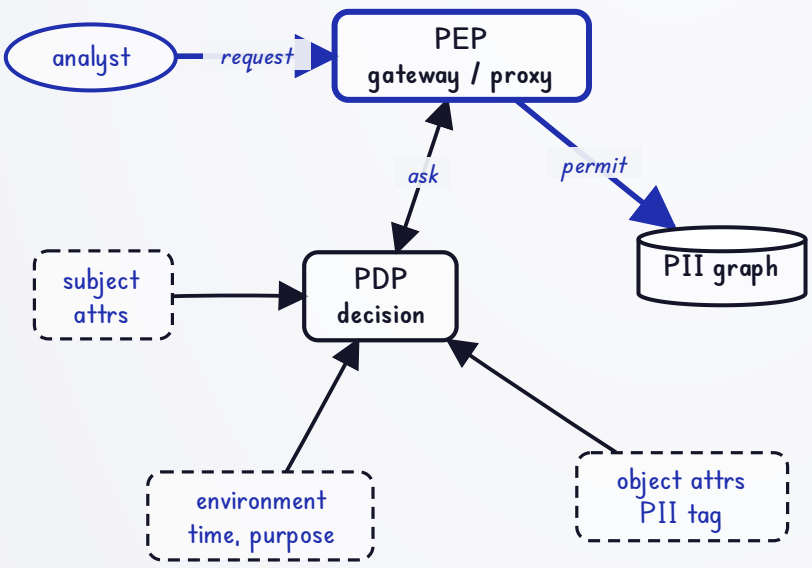
```
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dpv:  <https://w3id.org/dpv#> .
@prefix fnd:  <https://ontology.example.com/fnd/> .
fnd:sensitivity a owl:AnnotationProperty ;
  rdfs:comment "Classification of a property" .
fnd:dateOfBirth fnd:sensitivity fnd:PII ;
  rdfs:seeAlso dpv:PersonalData .
fnd:PII a skos:Concept ;
  skos:prefLabel "Personal data"@en ;
  skos:inScheme fnd:SensitivityScheme .
```

Parsed with rdflib. Tag **properties** (schema level), so every generated API, topic and view inherits the label. W3C DPVCG's Data Privacy Vocabulary (Community Group report) offers shared terms such as *dpv:PersonalData*.

72.2 Where to enforce

Enforcement point	Mechanism
Triplestore	named graph per sensitivity; graph-level ACLs
SPARQL proxy	rewrite / restrict *GRAPH* & *FROM*
API gateway	drop or mask fields tagged PII
Kafka	topic ACLs; PII-free event variant
Generated schemas	profile omits PII fields entirely

72.3 ABAC around the knowledge graph



NIST SP 800-162 (2014) defines attribute-based access control: a decision from **subject**, **object** and **environment** attributes evaluated against policy. The ontology supplies object attributes (PII tag, domain, named graph); HR / IAM supplies subject attributes.

72.4 ODRL 2.2 policy (W3C Rec. 2018)

```
{ "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Set",
  "uid": "https://ontology.example.com/policy/claims-pii-01",
  "permission": [{
    "target": "https://data.example.com/graph/claims-pii",
    "assignee": "https://data.example.com/team/fraud-analytics",
    "action": "read",
    "constraint": [{ "leftOperand": "purpose", "operator": "eq",
      "rightOperand": {"@id":
        "https://ontology.example.com/purpose/FraudInvestigation"}
}],
    "duty": [{ "action": "anonymize" }] }],
  "prohibition": [{
    "target": "https://data.example.com/graph/claims-pii",
    "assignee": "https://data.example.com/team/fraud-analytics",
    "action": "distribute" }] }
```

Read as: fraud analytics may **read** the PII claims graph only for the purpose *FraudInvestigation*, with a duty to **anonymize**, and may not **distribute** it. Expanded to RDF with pyld using the ODRL context terms.

Caution: ODRL *expresses* policy; it does not enforce it. A PDP / PEP must evaluate it at each enforcement point.

Key Takeaway: classify meaning once in the ontology, enforce it everywhere the data flows — graph, API, topic and generated schema.

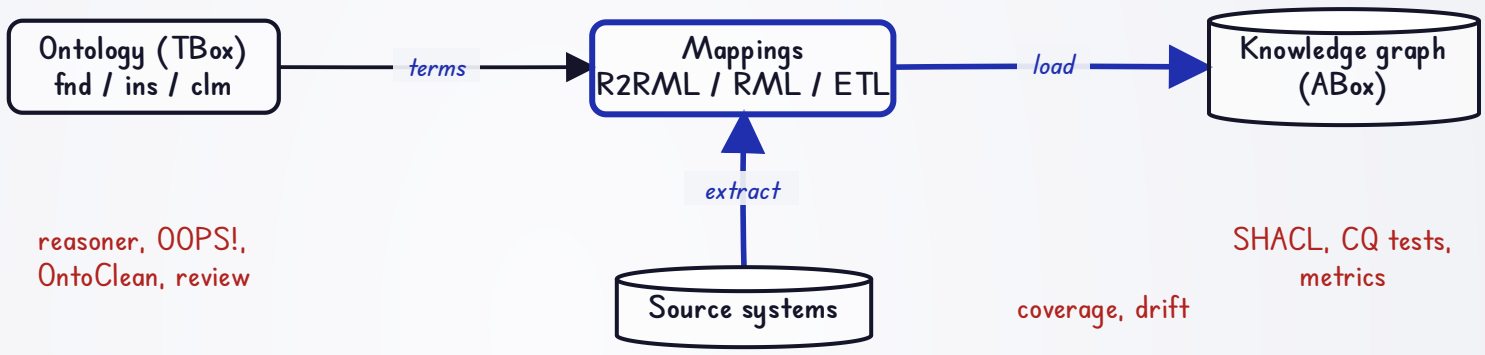
Sources: W3C ODRL Information Model 2.2 & Vocabulary (Rec. 2018) · Hu et al., NIST SP 800-162, Guide to ABAC (2014) · W3C DPVCG, Data Privacy Vocabulary (CG report, 2024) · W3C RDF 1.1 Concepts (2014): datasets & named graphs

73 Quality Dimensions

Part G · Accuracy, Consistency & Quality

73.1 Quality lives in three places

An enterprise KG can be wrong in the **model** (TBox), in the **mappings** that load it, or in the **data** (ABox). Each needs its own checks, owner and gate:



73.2 ISO/IEC 25012:2008 data quality model

ISO/IEC 25012 (SQuaRE series) defines **15 characteristics** seen from two points of view: *inherent* (the values themselves) and *system-dependent* (how a system keeps and serves them). Use it for the **ABox / data product** side and in data contracts.

Inherent (data itself)	Inherent + system-dependent	System-dependent
Accuracy, Completeness, Consistency, Credibility, Currentness	Accessibility, Compliance, Confidentiality, Efficiency, Precision, Traceability, Understandability	Availability, Portability, Recoverability

73.3 Linked Data quality

Zaveri et al. (2016) surveyed Linked Data quality work and organized **18 dimensions** (with 69 metrics) into four groups:

Group	Dimensions (Zaveri et al.)
Accessibility	availability, licensing, interlinking, security, performance
Intrinsic	syntactic validity, semantic accuracy, consistency, conciseness, completeness
Contextual	relevancy, trustworthiness, understandability, timeliness
Representational	representational conciseness, interoperability, interpretability, versatility

73.4 Ontology-specific criteria

Vrandečić (2009) collects eight criteria for the **ontology** itself. Six of them, each mapped to a test:

Criterion	How we check it	Page
Consistency / coherence	reasoner, justifications	78-79
Accuracy	expert review, OntoClean	80, 85
Clarity	definitions, OOPS! P08/P22	81, 86
Completeness / competency	CQ tests, coverage	83, 86
Conciseness	duplicate labels, P02	81
Organizational fitness	conformance to sources	84

Key Takeaway: Pick 4-6 dimensions that matter for *your* consumers, attach one measurable check to each, and publish the scores. A dimension without a check is only a slogan.

Sources: ISO/IEC 25012:2008 Data quality model · Zaveri et al., Quality assessment for Linked Data: A survey, SWJ 7(1) 2016 · Vrandečíc, Ontology Evaluation, Handbook on Ontologies (2009)

74 Logical Consistency with Reasoners

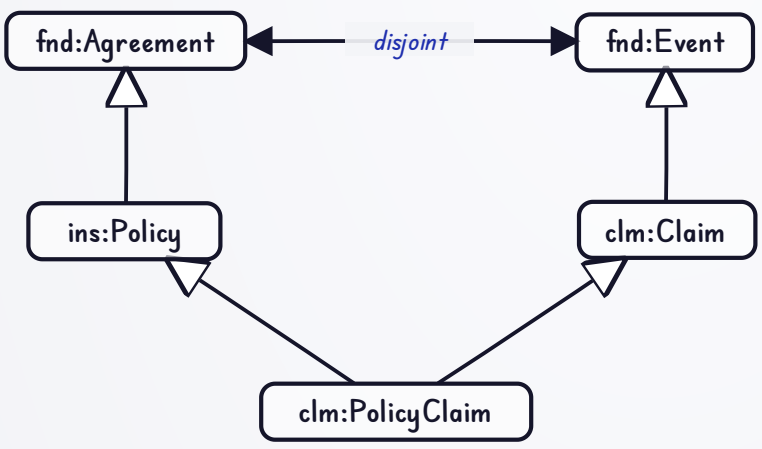
Part G · Accuracy, Consistency & Quality

74.1 Three questions a DL reasoner answers

1. **Consistent?** : is there any model at all? One contradiction (often in the ABox) makes everything entailed, so stop the build.
2. **Coherent?** : is every named class satisfiable, i.e. not equivalent to owl:Nothing?
3. **Classified?** : the inferred hierarchy: check that no surprise subclass links appear.

74.2 The classic slip

A modeller puts **PolicyClaim** under both Policy and Claim. Because Agreement and Event are disjoint, no individual can ever be a PolicyClaim:



= owl:Nothing (unsatisfiable)

74.3 Run it (tiny real test)

```
# prefixes omitted: owl rdfs fnd ins clm
fnd:Agreement owl:disjointWith fnd:Event .
ins:Policy    rdfs:subClassOf fnd:Agreement .
clm:Claim     rdfs:subClassOf fnd:Event .
# modelling slip: one class under both
clm:PolicyClaim rdfs:subClassOf ins:Policy , clm:Claim
.
```

```
# real output: owlready2 0.51 + bundled HermiT
HermiT unsat:
['http://www.w3.org/2002/07/owl#Nothing',
 'https://ontology.example.com/clm/PolicyClaim']
# ABox test: ex:POL-778 a ins:Policy, clm:LossEvent
ERROR: OwlReadyInconsistentOntologyError
```

Note: the TBox stays *consistent*; only the class is unsatisfiable. Typing one individual as both Policy and LossEvent makes the whole ontology inconsistent.

74.4 Choosing a reasoner

Reasoner	Covers	Use it for
ELK	OWL 2 EL profile	fast classification of large TBoxes; ROBOT default
HermiT	full OWL 2 DL (hypertableau)	release check: complete DL reasoning
Pellet	OWL 2 DL (tableau)	second DL reasoner to cross-check results

74.5 Put it in CI

```
robot reason --reasoner ELK --input clm-edit.owl \
     -D results/unsat-debug.owl --output clm-reasoned.owl
# exits non-zero if the ontology is incoherent; -D dumps a debug module
```

- Run ELK on every commit (cheap); run HermiT nightly / before release if you use axioms outside EL.
- Always reason with the **full import closure**: imported axioms can make your own classes unsatisfiable.
- Reasoners work under the open-world assumption: a missing value is not an error. Use SHACL (page 82) for that.

Sources: Glimm et al., HermiT: An OWL 2 Reasoner, J. Autom. Reasoning 53 (2014) · Kazakov, Krötzsch & Simančík, The Incredible ELK, J. Autom. Reasoning 53 (2014) · Sirin et al., Pellet: A practical OWL-DL reasoner, J. Web Semantics 5(2) 2007 · ROBOT docs: reason (robot.obolibrary.org)

Rajesh Kumar Gupta

75 Explaining Inconsistency

Part G · Accuracy, Consistency & Quality

75.1 What is a justification?

A **justification** for an entailment is a **minimal** subset of the ontology that still entails it: remove any one axiom and the entailment disappears (Horridge 2011; Kalyanpur et al. 2007). It is the "why" behind "PolicyClaim ⊑ owl:Nothing". One entailment can have **several** justifications; fixing one may not be enough.

75.2 Computing one: the black-box shrink loop



75.3 Real result for page 78

We ran the loop above with HermiT as the oracle. The LossEvent axiom was dropped as irrelevant; these 5 axioms remain:

```
# real output of our shrink loop (HermiT as oracle)
clm:Claim rdfs:subClassOf fnd:Event
ins:Policy rdfs:subClassOf fnd:Agreement
clm:PolicyClaim rdfs:subClassOf ins:Policy
clm:PolicyClaim rdfs:subClassOf clm:Claim
fnd:Agreement owl:disjointWith fnd:Event
```

Read it as a chain: PolicyClaim → Policy → Agreement, PolicyClaim → Claim → Event, and Agreement disjoint with Event. The **wrong** axiom is "PolicyClaim ⊑ Policy".

75.4 Debugging workflow



Root unsatisfiable classes cause the others. Fix roots first; many "derived" ones disappear.

75.5 Fix patterns

Fix pattern	When
Delete axiom	it was simply wrong
Weaken	subClassOf → a relation (Claim *onPolicy* Policy)
Split the class	two meanings in one name (OOPS! P07)
Move disjointness	disjoint at wrong level
Fix the import	imported axiom conflicts: raise upstream

```
robot explain --input clm-edit.owl --reasoner ELK -M unsatisfiability \
    --unsatisfiable root --explanation results/unsat.md
robot explain --input clm-edit.owl --reasoner ELK -M inconsistency \
    --explanation results/inconsistent.md   # markdown for the PR / ticket
```

Tip: Paste the justification into the change request. A reviewer who sees the 5 axioms can discuss the fix directly; a reviewer who sees only "unsatisfiable" has to start debugging from scratch.

Sources: Horridge, Justification Based Explanation in Ontologies, PhD thesis, Univ. of Manchester (2011) · Kalyanpur et al., Finding All Justifications of OWL DL Entailments, ISWC 2007 · ROBOT docs: explain (robot.obolibrary.org)

Rajesh Kumar Gupta

76 OntoClean

Part G · Accuracy, Consistency & Quality

76.1 What OntoClean is

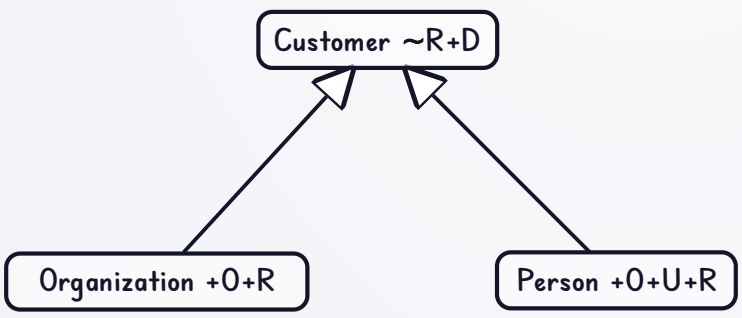
Guarino & Welty (2002) proposed OntoClean: tag each class with philosophical **meta-properties**, then check every subClassOf link against a few rules. It catches taxonomies that are logically consistent but **conceptually wrong**, which no reasoner can see.

Meta-property	Tags	Test question (paraphrased)
Rigidity	+R rigid, -R non-rigid, ~R anti-rigid	Can an instance stop being one and still exist? Person: no (+R). Customer: yes (~R).
Identity	+I carries an identity criterion, +O supplies its own, -I none	How do I tell two instances apart / the same? Policy: by contract identity.
Unity	+U whole, -U no common unity, ~U anti-unity	Is each instance a whole with clear parts? Vehicle +U; amount of money ~U.
Dependence	+D depends on another entity, -D not	Must something else exist? Insured needs a Policy (+D).

76.2 The subsumption rules (super ⊒ sub)

1. An **anti-rigid** class cannot subsume a **rigid** one (~R ⊒ +R forbidden).
2. If the super class carries identity (+I), the sub class must carry it too.
3. If the super carries a unity criterion (+U), the sub must carry the same one; an **anti-unity** (~U) super cannot subsume a +U sub.
4. A dependent super (+D) cannot subsume an independent sub (-D).

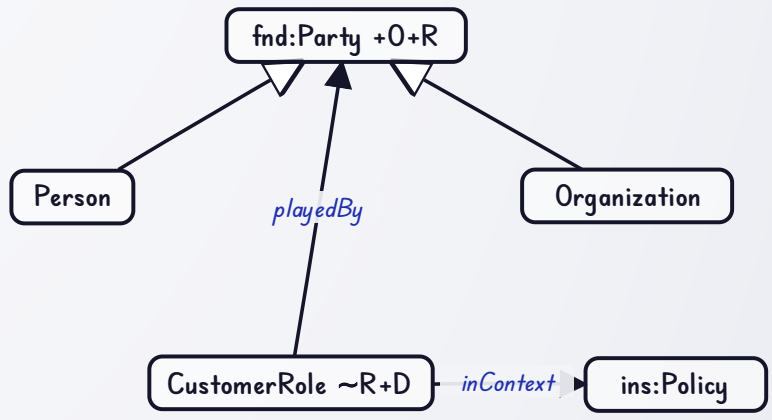
76.3 Insurance violation



both links violate rule 1 (~R ⊒ +R)

"Every Organization (or Person) is a Customer" fails rule 1: a party does not stop existing when its last policy lapses. Tags shown are illustrative.

76.4 Clean fix: roles



Rigid **types** (Party, Person) form the backbone; anti-rigid **roles** (Customer, Insured, Claimant, Policyholder) are separate classes linked to a Party and a context.

76.5 Running an OntoClean session on Monday

- Take the top 2-3 levels only (30-60 classes): that is where the damage is.
- Tag with SMEs using the test questions; record tags in an internal annotation property.
- A short SPARQL query then lists subClassOf pairs that break rules 1-4; review each one.

Key Takeaway: Watch for **roles modelled as types**. Ask "can it stop being one and still exist?" for every class with a business-relationship name (Customer, Broker, Insured, Payee).

Sources: Guarino & Welty, Evaluating Ontological Decisions with OntoClean, CACM 45(2) 2002 · Guarino & Welty, An Overview of OntoClean, Handbook on Ontologies (2004 / 2009)

77 Common Modeling Pitfalls

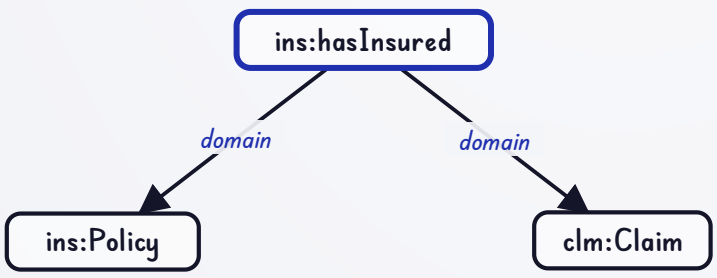
Part G · Accuracy, Consistency & Quality

77.1 OOPS! in one paragraph

OOPS! (OntOlogy Pitfall Scanner!) by Poveda-Villalón, Gómez-Pérez & Suárez-Figueroa (2014) is a web tool that scans an OWL file against a pitfall catalogue (the online catalogue currently lists **41**, P01-P41), each graded Critical, Important or Minor. Codes and names below are copied from the official catalogue page; examples are ours.

Code	Pitfall (official name)	Level	Insurance example / fix
P04	Creating unconnected ontology elements	Minor	clm:Reserve has no links: add *forClaim*
P05	Defining wrong inverse relationships	Critical	hasClaim inverse of onPolicy? check direction
P06	Including cycles in a class hierarchy	Critical	Claim ⊑ ClaimLine ⊑ Item ⊑ Claim: break it
P07	Merging different concepts in the same class	Minor	"PolicyOrQuote": split into two
P08	Missing annotations	Minor	no rdfs:label / skos:definition
P10	Missing disjointness	Important	Policy and Claim not declared disjoint
P11	Missing domain or range in properties	Important	clm:onPolicy without range
P19	Defining multiple domains or ranges in properties	Critical	see diagram below
P22	Using different naming conventions in the ontology	Minor	has_status vs hasClaimStatus
P24	Using recursive definitions	Important	Claim ≡ … some Claim
P41	No license declared	Important	add dcterms:license to the ontology header

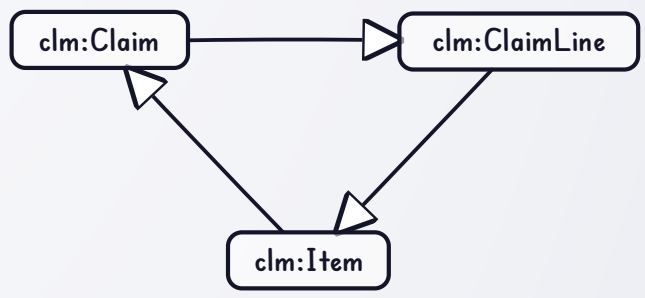
77.2 P19: two domains = intersection



domain = Policy ⊓ Claim
= empty if disjoint

In OWL, two rdfs:domain axioms mean the subject is **both**. Use one shared super class or a union, or two properties.

77.3 P06: cycle in the hierarchy



A subClassOf cycle makes all three classes **equivalent**: the reasoner will not complain, it silently merges them. Detect with a SPARQL path query: ?c rdfs:subClassOf+ ?c.

77.4 Automate or review?

- **Automate** (CI): P04, P08, P11, P22, P34-P36, P38, P41 via ROBOT report / SPARQL.
- Ask: does a query give the answer with no human judgement?

- **Review** (humans): P01, P05, P07, P10, P19, P24, P27-P31: need meaning, not syntax.
- Keep a "waived" list with reason + owner.

Sources: Poveda-Villalón, Gómez-Pérez & Suárez-Figueroa, OOPS!: An On-line Tool for Ontology Evaluation, IJSWIS 10(2) 2014 · OOPS! pitfall catalogue, oops.linkeddata.es/catalogue.jsp · W3C OWL 2 Primer (2012)

Rajesh Kumar Gupta

78 SHACL Validation at Scale

Part G · Accuracy, Consistency & Quality

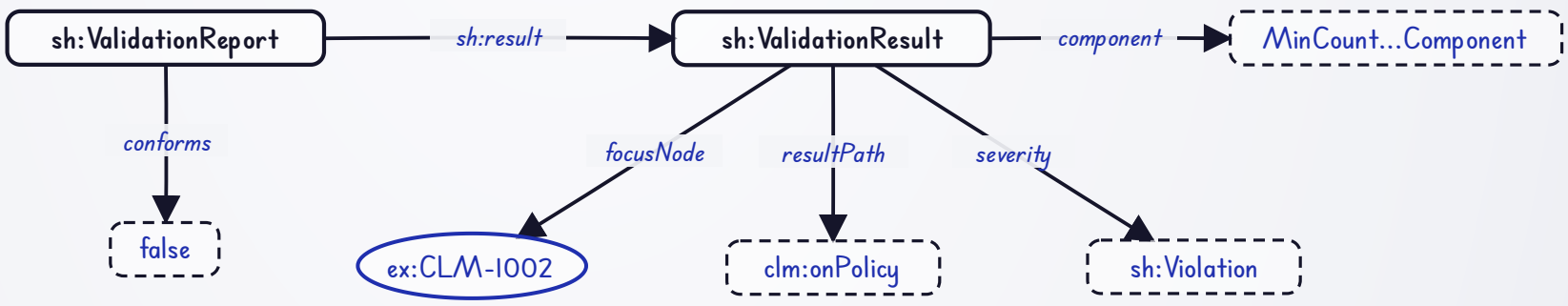
78.1 A real run on two claims

```
# shapes.ttl  (prefixes omitted: sh xsd clm ins clmsh)
clmsh:ClaimShape a sh:NodeShape ;
  sh:targetClass clm:Claim ;
  sh:property [ sh:path clm:onPolicy ;
      sh:class ins:Policy ;
      sh:minCount 1 ; sh:maxCount 1 ] ;
  sh:property [ sh:path clm:reportedOn ;
      sh:datatype xsd:date ; sh:minCount 1 ] ;
  sh:property [ sh:path clm:claimNote ;
      sh:maxLength 500 ;
      sh:severity sh:Warning ] .
# data.ttl: CLM-1001 complete; CLM-1002 has
#   reportedOn "14/03/2026" and no onPolicy
```

```
# real run, pySHACL 0.40 (output trimmed)
$ pyshacl -s shapes.ttl -f human data.ttl
Conforms: False   Results (2):
Constraint Violation in
    DatatypeConstraintComponent
  Focus Node: ex:CLM-1002
  Result Path: clm:reportedOn
  Message: Value is not Literal with
    datatype xsd:date
Constraint Violation in
    MinCountConstraintComponent
  Focus Node: ex:CLM-1002
  Result Path: clm:onPolicy
$ echo $?      -> 1   # gate can fail build
```

CLM-1001 conforms. CLM-1002 has a text date and no policy: two Violations. The Warning rule found nothing.

78.2 Anatomy of the validation report



78.3 Severity drives the gate

Severity	Gate action	Typical use
sh:Violation	block publish / merge	contract fields, keys, datatypes
sh:Warning	pass, open ticket to steward	new rule in rollout, soft limits
sh:Info	dashboard only	hints, style

78.4 Aggregate before you read

```
PREFIX sh: <http://www.w3.org/ns/shacl#>
SELECT ?shape ?comp ?sev (COUNT(?r) AS ?n)
WHERE {
  ?r a sh:ValidationResult ;
     sh:sourceShape ?shape ;
     sh:sourceConstraintComponent ?comp ;
     sh:resultSeverity ?sev .
} GROUP BY ?shape ?comp ?sev
  ORDER BY DESC(?n)
```

78.5 Tactics for millions of nodes

- **Target narrowly**: sh:targetClass, not "all subjects".
- **Partition**: validate per named graph, source or load batch, in parallel.
- **Validate the delta** plus the neighbours it touches, not the whole graph every load.
- **Fix inference mode**: decide if shapes see inferred types (pySHACL -i rdfs) and keep it the same in dev and prod.
- **Stage new rules**: ship as sh:Warning or sh:deactivated true; promote to Violation later.
- **Store reports** as RDF with load id + time: trend lines for free.
- Engines: pySHACL, Apache Jena SHACL, RDF4J ShaclSail, TopBraid SHACL API, plus triplestore built-ins.

Status: SHACL 1.0 is a W3C Recommendation (2017). SHACL 1.2 Core was still a Working Draft through 2026: check current status at w3.org before using 1.2 features.

Sources: W3C SHACL Recommendation (2017) · W3C SHACL 1.2 Core, Working Drafts 2025–2026 · pySHACL documentation (github.com/RDFLib/pySHACL)

⑦⑨ Competency Questions as Unit Tests

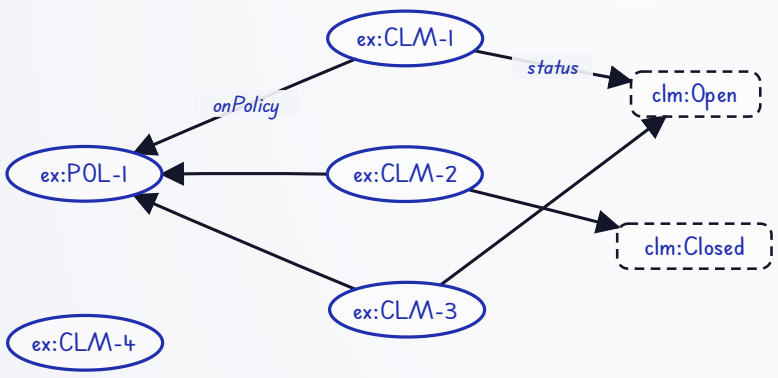
Part G · Accuracy, Consistency & Quality

79.1 Why CQs make good tests

Grüninger & Fox (1995) use competency questions to state what an ontology must be able to answer, and to evaluate it against them. Keet & Ławrynowicz (2016) push the idea into **test-driven development**: write the test first, then add axioms until it passes.

CQ in words (from SME) → SPARQL (SELECT / ASK) → Fixture (tiny TTL) → Expected (rows or true/false) → CI run (diff = fail)

79.2 Fixture + tests



Tiny hand-made fixture: 4 claims, 1 policy. CLM-4 is deliberately missing its policy to prove the guard test can fail.

```
# CQ-07 "Which open claims exist on policy POL-1?"
PREFIX clm: <https://ontology.example.com/clm/>
PREFIX ex:  <https://data.example.com/>
SELECT ?claim WHERE {
  ?claim a clm:Claim ; clm:onPolicy ex:POL-1 ;
         clm:hasClaimStatus clm:Open .
} ORDER BY ?claim
# expected (fixture): ex:CLM-1, ex:CLM-3
```

```
# CQ-08 guard: "no claim without a policy" ->
expect false
PREFIX clm: <https://ontology.example.com/clm/>
ASK { ?c a clm:Claim .
      FILTER NOT EXISTS { ?c clm:onPolicy ?p } }
```

79.3 Real run

```
# real run (rdflib 7.6) on fixture.ttl
CQ-07 open claims on POL-1:
     ['ex:CLM-1', 'ex:CLM-3']      PASS
CQ-08 ASK claim-without-policy: True
     FAIL (expected false)
     -> ex:CLM-4 has no clm:onPolicy
```

A failing guard is a **good** outcome here: it proves the test bites. In production fixtures, keep both a "good" and a "bad" file per rule.

79.4 Four kinds of test

Test kind	Pass rule	Good for
SELECT + expected set	actual rows = expected rows	answerability of a CQ
ASK guard	answer = expected boolean	business invariants
Violation query (robot verify)	zero rows returned	modelling rules on the TBox
Inference test	reasoned graph contains triple	axioms do their job

79.5 Monday checklist

- Store tests as files: *tests/cq-07.rq* + *cq-07.expected.csv*, one per CQ, ID linked to the requirement.
- Compare results order-free (sort rows) and on IRIs, not labels.
- Run against the **reasoned** ontology + fixture, so inferred types count.
- Every bug fixed = one new test (regression net grows with the model).

Sources: Grüninger & Fox, Methodology for the Design and Evaluation of Ontologies, IJCAI-95 Workshop (1995) · Keet & Ławrynowicz, Test-Driven Development of Ontologies, ESWC 2016 · W3C SPARQL 1.1 Query Language (2013) · ROBOT docs: verify

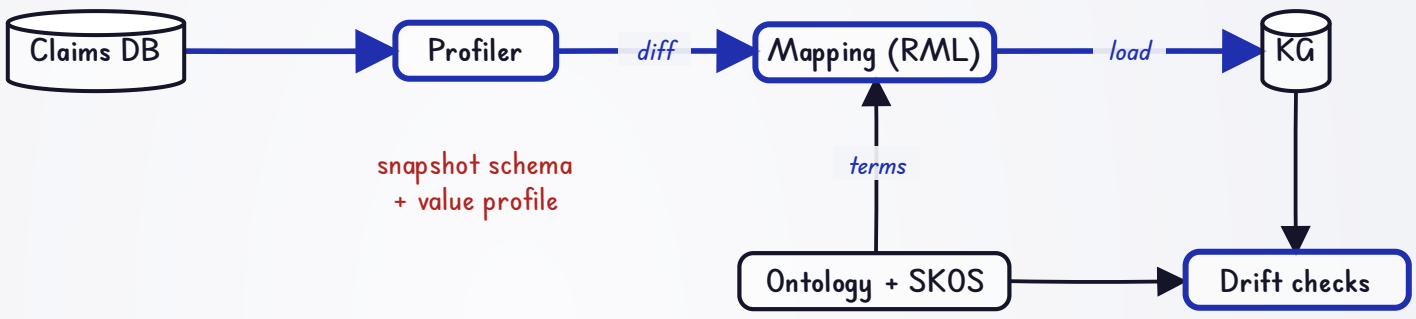
Rajesh Kumar Gupta

80 Conformance & Drift Detection

Part G · Accuracy, Consistency & Quality

80.1 Drift = the model and the real data move apart

Sources change without telling the ontology team: a renamed column, a new status code, a field that is now always empty. Conformance checking compares the **live** data against the **governed** model on every load.



80.2 Six drift checks

Drift type	Signal	Check
Source schema	column added / renamed / type changed	compare profile snapshots per load
Mapping coverage	source columns not mapped	mapped ÷ total columns, per table
Undeclared terms	data uses IRI not in ontology	SPARQL below
Code lists	value not in SKOS scheme	skos:inScheme check
Unexpected properties	extra predicates on a node	sh:closed true + sh:ignoredProperties
Empty classes	class has 0 instances for N loads	COUNT per class over time

80.3 Undeclared terms

```
# predicates in data, not declared in ontology
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?p WHERE {
 GRAPH <urn:g:data> { ?s ?p ?o }
 FILTER STRSTARTS(STR(?p),
        "https://ontology.example.com/")
 FILTER NOT EXISTS { GRAPH <urn:g:onto> {
   ?p a ?kind .
   VALUES ?kind { owl:ObjectProperty
    owl:DatatypeProperty
    owl:AnnotationProperty } } }
}
```

Real run on our fixture: **undeclared predicates: ['clm:reportDate']**: the source sent *reportDate*, the model says *reportedOn*.

80.4 Code-list drift

```
# status codes not in the governed scheme
PREFIX skos:
  <http://www.w3.org/2004/02/skos/core#>
PREFIX clm:
  <https://ontology.example.com/clm/>
SELECT ?claim ?st WHERE {
  ?claim clm:hasClaimStatus ?st .
  FILTER NOT EXISTS {
    ?st skos:inScheme clm:ClaimStatusScheme }
}
```

80.5 Decision rule

1. Unknown term: **quarantine** the triples in a named graph, do not drop them.
2. Open a change request: is it a real new concept or a source bug?
3. Fix the mapping or extend the model, then replay the quarantine.

Key Takeaway: Store each check result as a dqv:QualityMeasurement (page 86) with the load id: drift becomes a trend line, not a surprise.

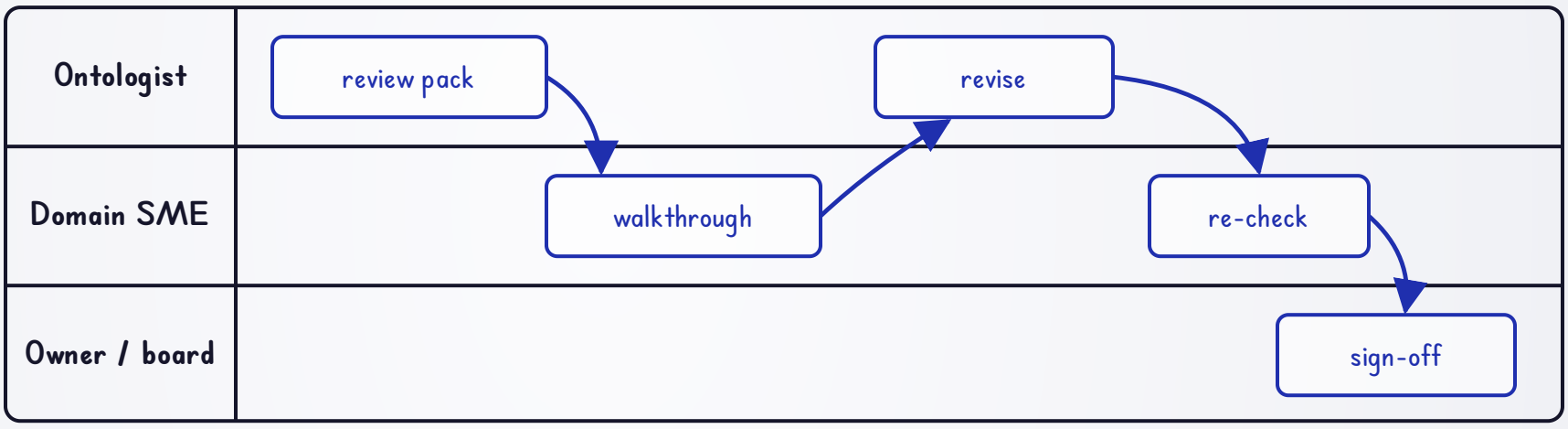
Sources: W3C R2RML: RDB to RDF Mapping Language (2012) · W3C SHACL Recommendation (2017): sh:closed · W3C SKOS Reference (2009) · W3C Data Quality Vocabulary (DQV), WG Note 2016

⑧① Expert Review & Business Validation

Part G · Accuracy, Consistency & Quality

81.1 What only people can check

A model can be consistent, pass every SHACL shape and still be **wrong about the business**. Expert review checks *meaning*: are the definitions true, are the boundaries where the business draws them, are the names the ones people use?



81.2 Definition review checklist

Seppälä et al. guideline (heading)	What the reviewer checks
Use the genus-differentia form	"A Claim is a demand (genus) for payment under a policy …"
Include exactly one genus / use the genus proximus	genus = the asserted parent class
Avoid circularity	"Claim: a claim made by …" fails
Avoid use/mention confusion	defines the thing, not the word "claim"
Avoid examples and lists	examples go to skos:example
Match textual and logical definitions	text and axioms say the same thing

81.3 Walkthrough with instance cards

- Show 10-20 **real** (masked) records rendered as cards: "Is this a Claim? a ClaimLine?"
- Include edge cases: reopened claim, subrogation, a claim with no payment.
- Read competency questions aloud and show the query answer.
- Record every disagreement as an issue, not as a hallway comment.

81.4 Record sign-off in the model

```
# prefixes omitted: skos dcterms prov clm
clm:Claim skos:changeNote
    """Definition approved by claims
     SME panel, CR-142"""@en ;
  dcterms:modified "2026-11-06" ;
  prov:wasAttributedTo
    <https://ontology.example.com/team/sme> .
```

81.5 Making review sessions work

- Send a **review pack** 3 working days ahead: diagram, definitions table, open questions.
- Time-box: 60 min, 15-25 terms. Review the diff, not the whole model.
- Two SMEs from different departments per term (e.g. claims and finance may each mean something different by "paid date").

Sources: Seppälä, Ruttenberg & Smith, Guidelines for writing definitions in ontologies, Ciência da Informação 46(1) 2017 · Noy & McGuinness, Ontology Development 101, Stanford KSL (2001) · W3C PROV-O (2013)

82 Ontology Metrics & Coverage

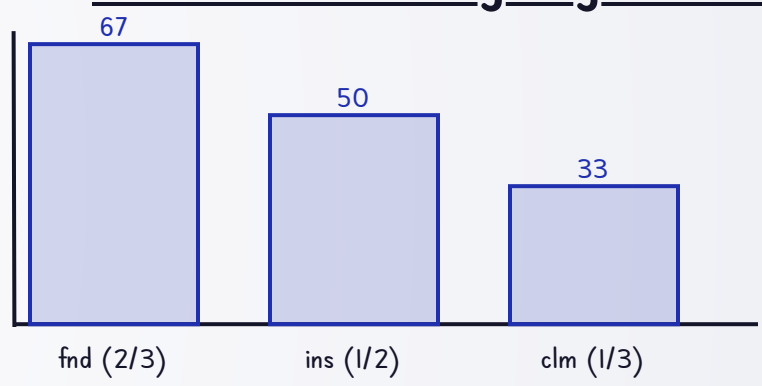
Part G · Accuracy, Consistency & Quality

82.1 Measure, then trend

Metrics do not prove an ontology is good; they flag where to look. Compute them with SPARQL on every build and watch the **trend**. Numbers below are a real run on our 8-class teaching fixture.

Metric (real run, tiny clm fixture)	Value
Named classes	8
Classes with skos:definition	4 (50%)
Orphan classes (no parent, child, domain, range)	2: clm:Reserve, fnd:Party
subClassOf axioms (named)	3
Inheritance richness IR = subclass links ÷ classes	3 ÷ 8 = 0.375
Fan-out (properties with domain = class)	clm:Claim 3, ins:Policy 1

82.2 Definition coverage by module



82.3 Definition coverage

```
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX skos:
<http://www.w3.org/2004/02/skos/core#>
SELECT (COUNT(?c) AS ?classes) (COUNT(?d) AS
?withDef)
WHERE { ?c a owl:Class . FILTER isIRI(?c)
      OPTIONAL { ?c skos:definition ?d } }
```

82.4 Orphan classes

```
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>
SELECT ?c WHERE { ?c a owl:Class . FILTER
isIRI(?c)
  FILTER NOT EXISTS { ?c rdfs:subClassOf ?x }
  FILTER NOT EXISTS { ?y rdfs:subClassOf ?c }
  FILTER NOT EXISTS { ?p rdfs:domain|rdfs:range
?c } }
```

82.5 Structural metrics (OntoQA)

Tartir et al. (2005) define schema metrics such as **inheritance richness** (average subclasses per class: deep vs. flat) and **relationship richness** (share of non-subClassOf relations: a taxonomy vs. a rich model).

robot measure gives entity / axiom counts and OWL 2 profile; with an *-reasoner* metrics set it adds the unsatisfiable-class count.

82.6 Record results as DQV

```
@prefix dqv: <http://www.w3.org/ns/dqv#> .
@prefix xsd:
<http://www.w3.org/2001/XMLSchema#> .
@prefix q:   <https://ontology.example.com/quality/> .
q:m-2026-11-06-defcov a dqv:QualityMeasurement ;
  dqv:computedOn
<https://ontology.example.com/clm/> ;
  dqv:isMeasurementOf q:DefinitionCoverage ;
  dqv:value "0.50"^^xsd:decimal .
q:DefinitionCoverage a dqv:Metric ;
  dqv:expectedDataType xsd:decimal .
```

Rule of thumb: Set floors, not targets: "definition coverage may not go down in a release" works. "Reach 95%" invites one-line junk definitions.

Sources: Tartir et al., OntoQA: Metric-Based Ontology Quality Analysis, IEEE ICDM Workshop (2005) · ROBOT docs: measure · W3C Data Quality Vocabulary (DQV), WG Note 2016

Rajesh Kumar Gupta

83 Regression & Release Gates

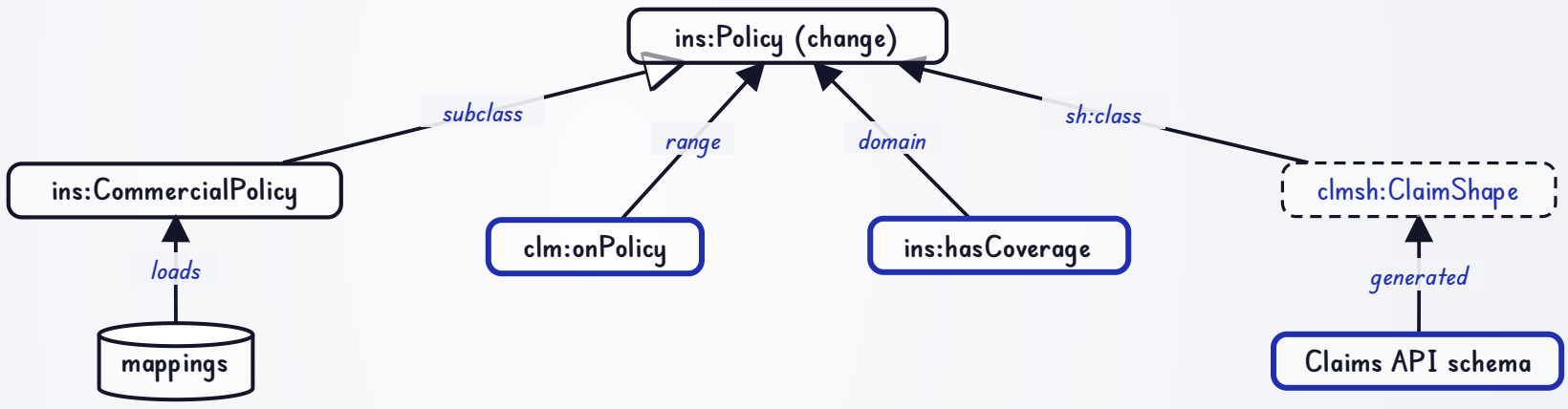
Part G · Accuracy, Consistency & Quality

83.1 Release gates in order (fail fast)



Each gate runs on the pull request; G1-G5 are automatic, G6 is a human reading **robot diff** output. Cheap gates first so a typo never waits for a 20-minute reasoner run.

83.2 Impact analysis before merge




```
# what depends on ins:Policy?
PREFIX rdfs:
  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sh:  <http://www.w3.org/ns/shacl#>
PREFIX ins: <https://ontology.example.com/ins/>
SELECT ?x ?how WHERE {
  { ?x rdfs:subClassOf+ ins:Policy .
    BIND("subclass" AS ?how) }
  UNION { ?x rdfs:domain ins:Policy .
    BIND("domain" AS ?how) }
  UNION { ?x rdfs:range ins:Policy .
    BIND("range" AS ?how) }
  UNION { ?x sh:targetClass|sh:class ins:Policy .
    BIND("shape" AS ?how) }
}
```

```
# real run (rdflib) on our tiny model
['ins:CommercialPolicy', 'subclass']
['ins:hasCoverage', 'domain']
['clm:onPolicy', 'range']
['_:n0495...', 'shape']   # property shape
#   inside clmsh:ClaimShape
```

Add your own UNIONs for mappings and generated API schemas (via an annotation such as x-ontology, page 67). The list goes into the change request.

83.3 Semantic versioning for ontologies

Change	Version bump	Why
add class / property, add definition	minor (2.2 → 2.3)	old data and queries still valid
fix typo in label, comment	patch (2.3.0 → 2.3.1)	no logical change
remove / rename term, narrow range, new disjointness	major (2 → 3)	old data may become invalid or inconsistent

semver.org defines MAJOR.MINOR.PATCH for APIs; applying it to an ontology is a team convention: write your own table like this one into the charter.

83.4 Version metadata (OWL 2)

```
# prefixes omitted: owl dcterms clm
@base <https://ontology.example.com/clm/> .
<> a owl:Ontology ;
  owl:versionIRI <2.3.0/> ;
  owl:priorVersion <2.2.1/> ;
  owl:backwardCompatibleWith <2.2.1/> ;
  dcterms:modified "2026-11-06" .
# keep deprecated terms 1+ minor cycle
clm:claimRef owl:deprecated true .
```

```
robot diff --left clm-2.2.1.owl --right clm-edit.owl \
     --format markdown --output results/diff.md
```

83.5 Regression rules

- Keep the previous release as a fixture: re-run its CQ tests and SHACL on the new model.
- Deprecate before delete: **owl:deprecated true** + replacement note, removed only in a major.
- A red gate is never "overridden" silently: waivers need an owner, reason and expiry date.

Sources: Jackson et al., ROBOT: A Tool for Automating Ontology Workflows, BMC Bioinformatics 20 (2019) · ROBOT docs: diff, report · Semantic Versioning 2.0.0 (semver.org) · W3C OWL 2 Structural Specification (2012)

Rajesh Kumar Gupta

84 Part G Recap & Quality Gates

Part G · Accuracy, Consistency & Quality

84.1 The quality stack

Business validity
expert review, definitions, walkthroughs (85)

Answerability
CQ unit tests, coverage metrics (83, 86)

Data conformance
SHACL shapes, drift checks (82, 84)

Conceptual soundness
OntoClean, OOPS! pitfalls (80, 81)

Logical consistency
reasoner + justifications (78, 79)

Syntax & hygiene
parse, robot report (87)

84.2 Quality gate checklist (print this)

#	Gate check	Tool	Pass rule	Auto?
1	Files parse, prefixes resolve	rdflib / ROBOT	no errors	yes
2	Labels, definitions, license	robot report	no ERROR rows	yes
3	Consistent + all classes satisfiable	ELK / HermiT	zero unsat	yes
4	Pitfalls (critical)	OOPS! / SPARQL	none open or waived	part
5	Fixtures conform	SHACL engine	zero Violations	yes
6	Competency questions	SPARQL tests	all PASS	yes
7	No drift on last load	drift queries	quarantine empty	yes
8	Metrics floors held	SPARQL / robot measure	no metric fell	yes
9	Diff reviewed, SME sign-off	robot diff + review	approved CR	no
10	Version + changelog	owl:versionIRI	semver bump correct	part

84.3 What to remember

- Reasoners find **contradictions**; SHACL finds **missing / malformed data**; only people find **wrong meaning**.
- Every check needs an owner, a pass rule and a place in CI.
- Explanations (justifications, SHACL results) belong in the ticket.

84.4 Start on Monday

1. Add parse + robot report + ELK to CI.
2. Write 10 CQ tests with a fixture.
3. Add SHACL for the 3 most-used classes.
4. Book a 60-min SME review of the top 20 definitions.

Key Takeaway: Quality is not a phase at the end: it is a set of small, automated gates that run on every change, plus a regular human review of meaning.

Sources: ISO/IEC 25012:2008 · Guarino & Welty (2002) · Poveda-Villalón et al. (2014) · Jackson et al., ROBOT (2019)

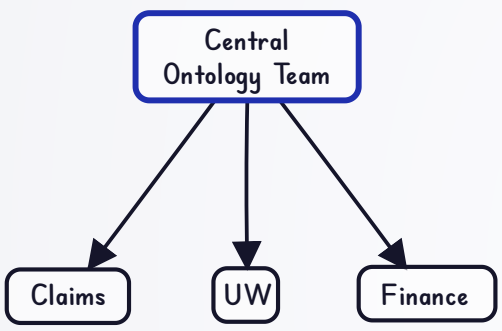
Rajesh Kumar Gupta

85 Operating Models

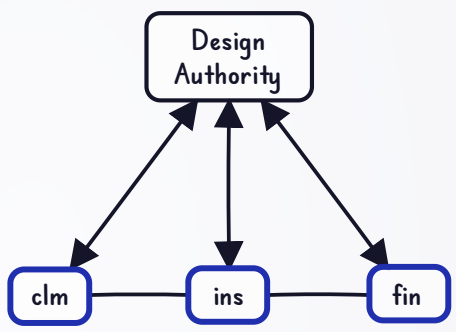
Part H · Governance, Coordination & Best Practices

85.1 Three ways to organise the work

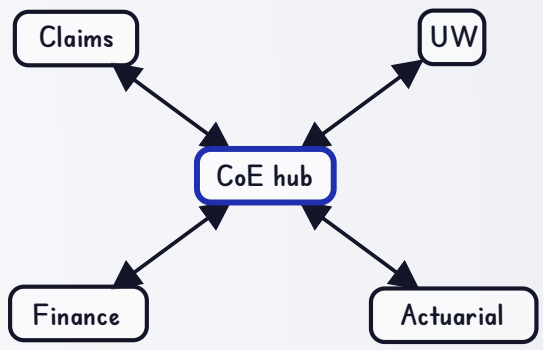
DAMA-DMBOK2 describes **centralized, replicated** and **federated** data-governance operating models. For ontology and canonical work most enterprises end up with one of the three shapes below; the **hub-and-spoke Centre of Excellence (CoE)** is a common practitioner variant of federated.



A. Centralized



B. Federated



C. Hub-and-spoke CoE

85.2 Compare before you choose

Model	Who models	Strength	Typical risk
Centralized	One enterprise team models fnd, ins, clm	One voice, consistent style, fast start	Bottleneck; domains feel it is "IT's model"
Federated	Each domain owns its module; a Design Authority owns fnd + rules	Scales; domain experts own meaning	Drift and duplicate concepts if rules are weak
Hub-and-spoke CoE	Hub sets patterns, tooling, fnd; spokes are embedded ontologists	Balance of control and reach	Hub under-staffed → spokes go native

85.3 Decision rules

- Fewer than ~3 consuming domains and no ontologists in the business → **start centralized**.
- Several domains already publish APIs/data products → **federate**, keep *fnd* central.
- Data-mesh organisation → federated computational governance: global rules automated in the platform, local models owned by domains (Dehghani 2022).

85.4 Map to team types

- Ontology team as a **platform** team: tooling, CI, registry, generators.
- Ontologists as an **enabling** team: coach domains, then step back.
- Domain modules owned by **stream-aligned** teams (Skelton & Pais 2019 team types).

85.5 Typical evolution path

Centralized (bootstrap fnd + first domain) → **Hub-and-spoke** (embed ontologists) → **Federated** (domains own modules) → **Automated rules** (CI + registry gates)

Key Takeaway: Pick the model for *today*'s maturity and write down the trigger for moving to the next one (e.g. "third domain onboarded"). The operating model is a decision record, not a permanent truth.

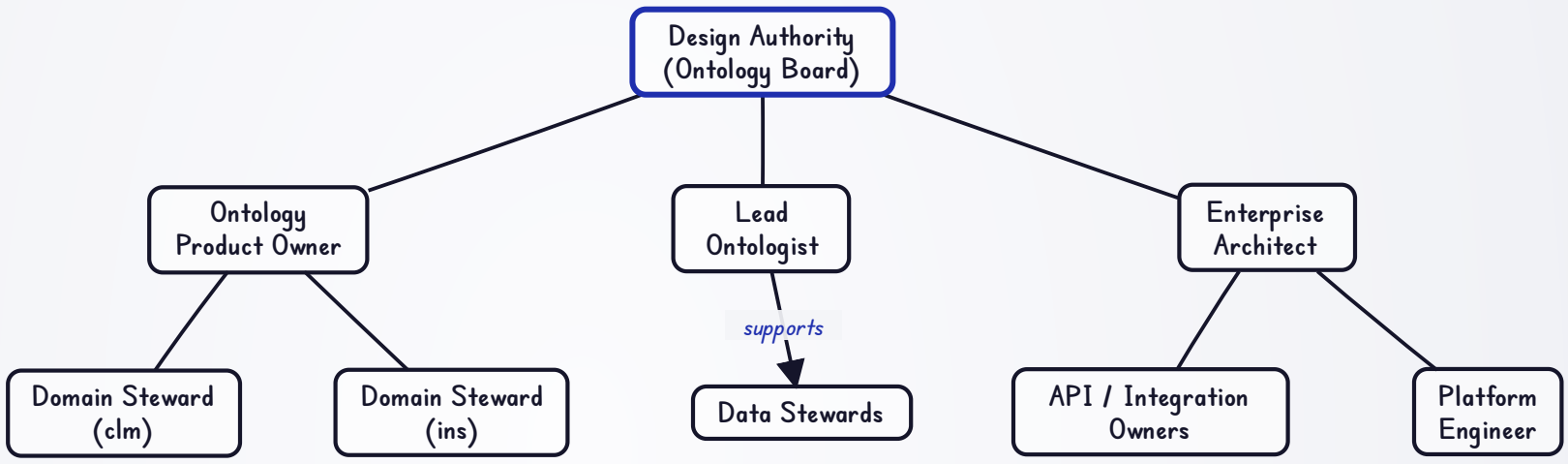
Sources: DAMA International, DAMA-DMBOK2 (2017), ch. Data Governance · Dehghani, Data Mesh (2022) · Skelton & Pais, Team Topologies (2019)

Rajesh Kumar Gupta

86 Roles & Responsibilities

Part H · Governance, Coordination & Best Practices

86.1 The cast around one enterprise ontology



86.2 Who owns what

Role	Owns / decides	Typical home
Ontology product owner	Roadmap, priorities, funding, release scope; accountable for the ontology as a product	Data office / CDO
Lead ontologist	Modelling patterns, fnd module, axioms, reasoner & SHACL quality, reviews of every PR	Ontology team (hub)
Domain steward	Meaning of terms in one module (clm, ins): definitions, code lists, sign-off	Business line (Claims, UW)
Data steward	Quality and mapping of source data to the classes; reports drift	Source-system teams
API / integration owner	Canonical schemas and API contracts generated from the model; consumer comms	Integration / platform
Platform engineer	Triplestore, CI pipeline, schema registry, generators, catalog	Platform team
Design Authority	Cross-domain and breaking changes, naming policy, exceptions, ADRs	Architecture board

86.3 Practical rules

- Name a **person** per role per module, not a team mailbox.
- One person may hold several roles early on; write it down.
- Domain stewards are business people: give them review tools (HTML docs, diagrams), not Turtle.

86.4 Role anti-patterns

- Ontologist as sole definer of business meaning.
- Steward in name only: never reviews a CR.
- No API owner: the model never reaches running systems.

Key Takeaway: Meaning is owned by the business (domain stewards); form and consistency are owned by the ontology team. Keep those two accountabilities separate and explicit.

Sources: DAMA International, DAMA-DMBOK2 (2017): data stewardship roles · Seiner, Non-Invasive Data Governance (2014) · Kendall & McGuinness, Ontology Engineering (2019)

87 RACI for Changes

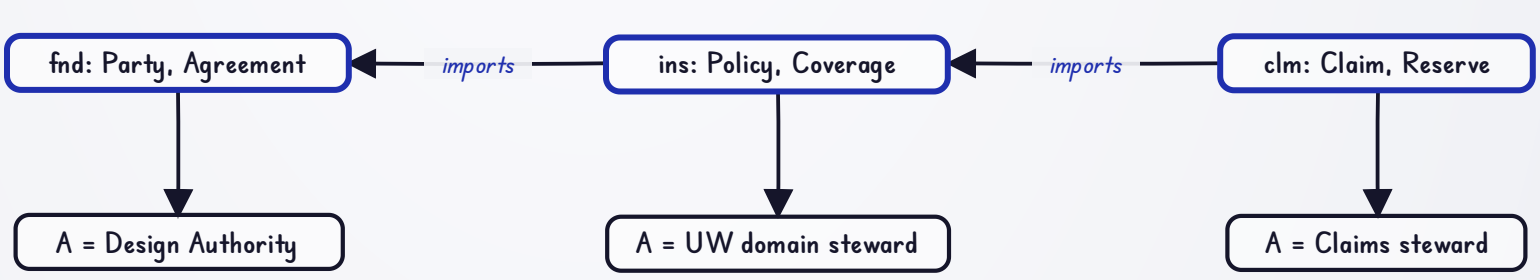
Part H · Governance, Coordination & Best Practices

87.1 RACI for ontology & canonical changes

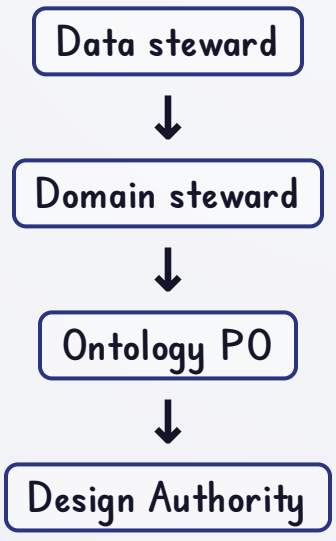
Responsible does the work · **A**ccountable signs off (exactly one per row) · **C**onsulted before (two-way) · **I**nformed after (one-way). DA = Design Authority, PO = ontology product owner, LO = lead ontologist, DSt = domain steward, DtS = data steward, API = API/integration owner, Plat = platform engineer.

Activity	DA	PO	LO	DSt	DtS	API	Plat
Raise change request (CR)	I	I	C	A/R	C	C	
Triage & classify CR		I	C	A/R	C	I	
Impact analysis (imports, APIs, data)		I	A/R	C	C	R	
Model change in a clm / ins module			R	A	C	C	
Change to fnd (shared core)	A	C	R	C	I	C	
Breaking change / deprecation	A	R	R	C	C	C	I
Approve business definitions		I	C	A/R	C	I	
CI gates + release build		A	R	I		I	R
Regenerate canonical / API schemas		I	C	I		A/R	C
Communicate release to consumers		A	C	C		R	I

87.2 Accountability follows module scope



87.3 Escalation path



87.4 How to make RACI real

- Encode it: a **CODEOWNERS** file in the ontology repo maps *clm/* to the claims steward and *fnd/* to the Design Authority, so the PR cannot merge without the right reviewer.
- Exactly one **A** per row; two A's means nobody decides.
- Review the matrix when the operating model changes (page 89).
- Keep "I" lists as subscribable channels, not e-mail CC lists.

Key Takeaway: A RACI that is not wired into the repository and the review tool is decoration. Put it in CODEOWNERS and branch protection.

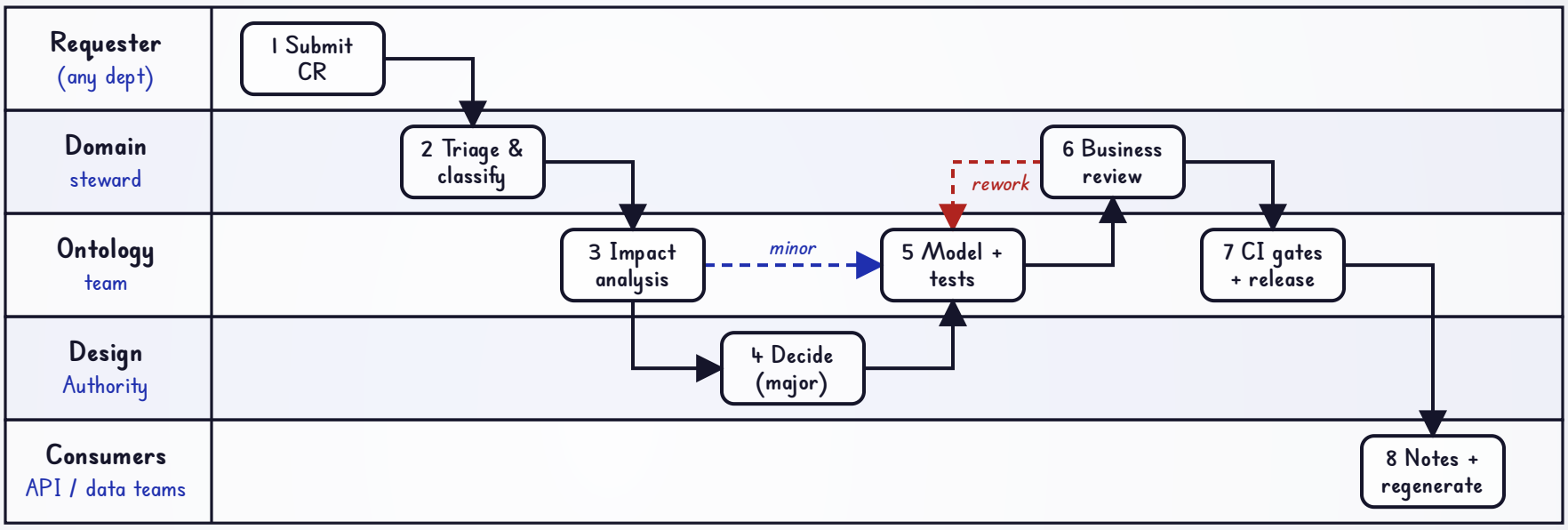
Sources: PMI, PMBOK Guide 6th ed. (2017): responsibility assignment matrix / RACI · DAMA International, DAMA-DMBOK2 (2017) · GitHub Docs: About code owners

Rajesh Kumar Gupta

88 Change Request Lifecycle

Part H · Governance, Coordination & Best Practices

88.1 Change-request lifecycle (swimlane)



Blue dashed = minor change skips the Design Authority; red dashed = business review sends it back to modelling.

88.2 Classify every CR first

Class	Examples (clm module)	Version bump	Path
Editorial	Fix typo in skos:definition, add altLabel	patch 2.4.1	Steward + ontologist review
Additive	New class clm:Subrogation, new ClaimStatus concept	minor 2.5.0	Steps 1-3, 5-8
Breaking	Rename property, tighten SHACL minCount, delete concept	major 3.0.0	All steps incl. DA + deprecation window

88.3 CR record (tracker)

```
# CR-0157  (illustrative)
title: Add "Withdrawn" to ClaimStatusScheme
requester: Claims Ops
module: clm
class: additive
cq: "Which claims did the claimant
     withdraw in 2026?"
impact: 3 APIs, 1 Kafka topic
target_release: clm 2.5.0
```

88.4 Rules that keep it moving

- Every CR carries a **competency question** or a concrete data example.
- Triage within a fixed time box (set your own service level).
- Impact analysis is automated where possible: SPARQL over the import graph + the schema registry list of consumers.
- Breaking changes need an **owl:deprecated** period before removal.
- Release notes are generated from merged CRs, never hand-written.

Key Takeaway: One path for everyone: even the ontology team raises CRs. A visible, predictable lifecycle is what makes departments trust (and fund) the model.

Sources: Semantic Versioning 2.0.0 (semver.org) · W3C OWL 2 Primer (2012): owl:deprecated · Grüninger & Fox (1995): competency questions

89 Design Authority & ADRs

Part H · Governance, Coordination & Best Practices

89.1 Charter: one page, signed

1. **Mandate** : owns fnd, naming/IRI policy, cross-domain and breaking changes
2. **Membership** : ontology PO (chair), lead ontologist, enterprise architect, 1 steward per domain, API owner
3. **Quorum & voting** : e.g. chair + lead ontologist + affected steward; consensus, chair breaks ties
4. **Inputs** : CRs classified "breaking" or touching fnd; waiver requests
5. **Outputs** : decisions as ADRs, published within the week
6. **Escalation** : to the data governance council

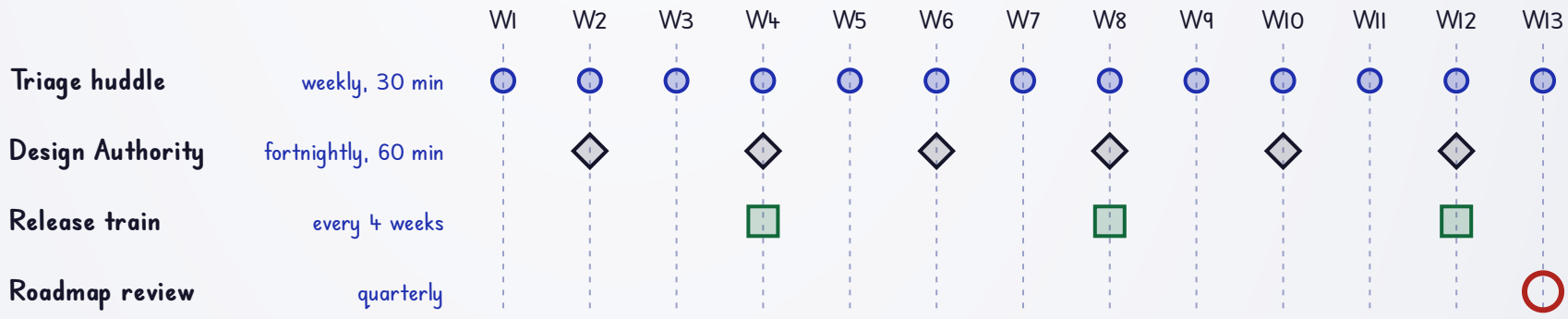
89.2 What it must NOT do

- Review every editorial or additive change (delegate to stewards).
- Debate definitions without a CQ or data example.
- Meet without a pre-read agenda.

89.3 Standing agenda (60 min)

1. **5 min** : actions from last meeting
2. **35 min** : breaking / fnd CRs, each with impact analysis
3. **10 min** : waiver requests and expiries
4. **10 min** : decisions read back; ADR owners named

89.4 Cadence over one quarter (illustrative)



89.5 Decision record (ADR style)

```
# ADR-0017: ClaimStatus as SKOS, not OWL classes
Status: Accepted  (2026-03-04)
Context: Claims, Finance and the portal use
  different status codes; values change yearly.
Decision: model clm:ClaimStatusScheme as a
  skos:ConceptScheme; each status a skos:Concept;
  link source codes with skos:exactMatch.
Consequences: + codes change without a schema
  release; - no OWL reasoning over statuses.
Supersedes: none
```

89.6 ADR habits

- Nygard's format: **title, context, decision, status, consequences**.
- Statuses: proposed → accepted → deprecated / superseded.
- Store ADRs in the ontology repo (*docs/adr/0017-...md*) so they version with the model.
- Link the ADR IRI from the affected term, e.g. with *rdfs:seeAlso*.
- Never edit an accepted ADR; write a new one that supersedes it.

Key Takeaway: The board's product is **decisions with reasons**. If a decision is not in an ADR within a week, it did not happen.

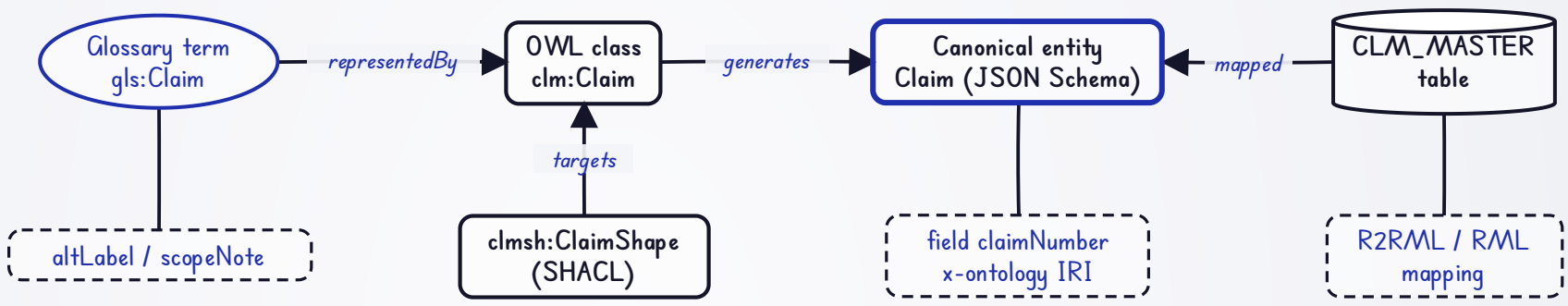
Sources: Nygard, "Documenting Architecture Decisions" (2011) · W3C SKOS Reference (2009) · DAMA International, DAMA-DMBOK2 (2017)

Rajesh Kumar Gupta

⑨⓪ Glossary ↔ Ontology ↔ Canonical

Part H · Governance, Coordination & Best Practices

90.1 One meaning, four representations



The **glossary** is the business-facing word, the **ontology** the formal meaning, the **canonical** the exchange structure and the **physical** layer the stored data. Alignment = explicit, queryable links between all four.

90.2 Glossary term in SKOS, linked to the class

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix gls:  <https://ontology.example.com/glossary/> .
@prefix gov:  <https://ontology.example.com/gov/> .
@prefix clm:  <https://ontology.example.com/clm/> .

# SKOS has no standard term-to-class link:
# declare your own annotation property
gov:representedBy a owl:AnnotationProperty .

gls:Claim a skos:Concept ;
  skos:inScheme gls:BusinessGlossary ;
  skos:prefLabel "Claim"@en ;
  skos:altLabel "Insurance claim"@en ;
  skos:definition """A request by an insured or
    third party for payment under a policy."""@en ;
  gov:representedBy clm:Claim .

clm:Claim a owl:Class ;
  rdfs:label "Claim"@en ;
  rdfs:seeAlso gls:Claim .
```

90.3 ISO/IEC 11179 lens

ISO/IEC 11179 splits a data element into an **object class** + **property** (the data element concept) and a **value domain**. That maps neatly to class, property and datatype / code list:

Term	Object class	Property	Value domain
Claim Reported Date	clm:Claim	clm:reportedOn	xsd:date
Claim Status	clm:Claim	clm:hasClaimStatus	ClaimStatus scheme
Paid Amount	clm:Payment	fnd:hasAmount	fnd:Money

90.4 Alignment rules

- One glossary term → one ontology element of record.
- Synonyms are *skos:altLabel*, never new classes.
- Same word, two meanings: two concepts with *skos:scopeNote*.
- Definitions follow ISO 704 style: genus + differentia.
- The steward edits the glossary; the ontologist edits the class; CI checks both links.

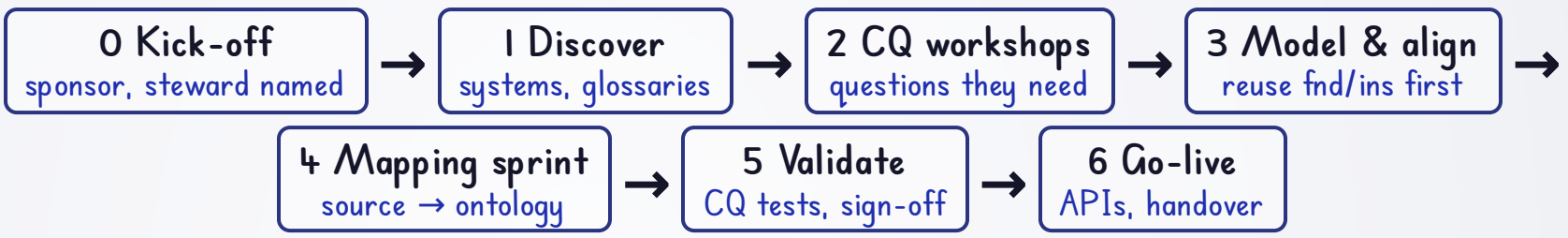
Key Takeaway: Run a nightly SPARQL check: every glossary term with status "approved" must have a representedBy link, and every class must point back. Gaps become stewardship tasks.

Sources: ISO/IEC 11179-1:2023 Metadata registries · W3C SKOS Reference (2009) · ISO 704:2022 Terminology work · DAMA International, DAMA-DMBOK2 (2017)

⑨① Onboarding Playbook

Part H · Governance, Coordination & Best Practices

91.1 Onboarding a department in seven steps



Durations depend on scope; a common practice is short, time-boxed iterations per step rather than one long analysis phase.

91.2 Phase card: what goes in, what comes out

Phase	Key activities	Artefact	Exit criterion
Discover	Inventory systems, reports, code lists, existing glossaries	Source inventory	Top sources + owners listed
CQ workshops	Elicit questions the department cannot answer today	Requirements doc + CQ list	CQs prioritised, each with sample answer
Model & align	Reuse fnd/ins terms; new terms only in the domain module	Module PR + ADRs	Reasoner clean, SHACL passes
Mapping sprint	R2RML/RML or ETL mappings for priority sources	Mappings + test data	CQ queries return expected rows
Validate	Steward walkthrough of definitions and diagrams	Sign-off record	Steward approves release

91.3 CQ workshop (90 min)

1. **10 min** : goal, glossary of the room
2. **25 min** : each person writes 3 questions they need answered
3. **20 min** : cluster; mark terms that recur
4. **20 min** : pick the top questions; give a sample answer
5. **15 min** : name owners and data sources

91.4 Sample claims CQs

- Which open claims on a policy have reserves but no payment yet?
- Which claims were reopened after closure this year?
- For a loss event, which policies and coverages are triggered?

91.5 Tips

- Bring their reports, not your model.
- Show a diagram of *their* terms on day one.
- Leave each session with a CQ list they agreed.

Key Takeaway: A department is onboarded when its own competency questions run as tests in CI and its steward signs the release, not when the classes exist.

Sources: Grüninger & Fox (1995) · Suárez-Figueroa et al., NeOn methodology (2012): ORSD · Poveda-Villalón et al., LOT (2022)

Rajesh Kumar Gupta

92 Adoption & Enforcement

Part H · Governance, Coordination & Best Practices

92.1 Carrots and sticks

Make it easy (incentives)	Make it required (enforcement)
Generators: ontology → JSON Schema, Avro, OpenAPI fragments in one command	API style guide rule: every schema property carries an *x-ontology* IRI
Searchable catalog / docs site with diagrams and examples	Lint in the API pull request; build fails on error
Ready-made SKOS code lists with source-code crosswalks	Schema registry compatibility mode set per topic
Ontologist office hours, embedded in squads	Design Authority waiver needed to skip, with expiry date
Public reuse dashboard per team	Architecture review checks canonical use for new integrations

92.2 Put the gates where teams already pass

API spec PR (OpenAPI 3.1) → Lint (x-ontology IRI valid) → SHACL check (sample payloads) → Registry (compatibility check) → Catalog (publish + lineage)

92.3 Lint rule (Spectral ruleset)

```
rules:
  x-ontology-required:
    description: Each schema property must
      carry an ontology IRI in x-ontology
    severity: error
    given: "$.components.schemas[*].properties[*]"
    then:
      field: x-ontology
      function: truthy
```

92.5 Measure reuse (SPARQL)

```
PREFIX api: <https://ontology.example.com/api/>
# API fields harvested from specs into the KG
SELECT ?api (COUNT(?f) AS ?fields)
       (COUNT(?iri) AS ?tagged)
WHERE {
  ?api api:hasField ?f .
  OPTIONAL { ?f api:ontologyTerm ?iri }
}
GROUP BY ?api
```

92.4 Registry gate

Confluent Schema Registry compatibility types include BACKWARD, FORWARD, FULL (each with a _TRANSITIVE variant) and NONE. Choose per topic; default in the registry is BACKWARD.

92.6 Waivers, not exceptions

- Logged as an ADR with owner and expiry.
- Reviewed at the quarterly roadmap review.
- Recurring waivers = the model is missing something.
- Lint also fails on IRIs marked owl:deprecated.

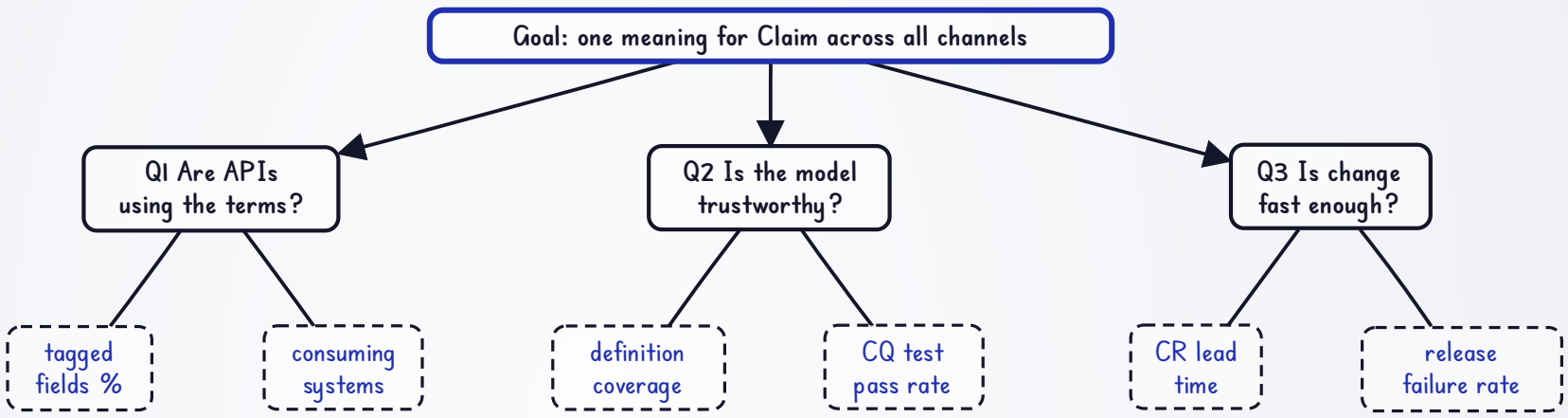
Key Takeaway: Enforcement without enablement breeds workarounds. Ship the generator and the docs first, switch the lint rule from *warn* to *error* second.

Sources: OpenAPI Specification 3.1.0 (2021) · Stoplight Spectral documentation (rulesets) · Confluent docs: Schema Evolution and Compatibility · Dehghani, Data Mesh (2022)

⑨③ KPIs & Measuring Value

Part H · Governance, Coordination & Best Practices

93.1 Goal → Question → Metric (GQM)



93.2 KPIs you can actually compute

KPI	How it is measured	Data source
Definition coverage	classes + properties with skos:definition / all in-namespace terms	SPARQL on release
Tagged-field ratio	API/schema fields with an x-ontology IRI / all fields	Specs harvested to KG
Consuming systems	distinct APIs, topics, data products that reference the ontology	Registry + catalog
CQ test pass rate	passing CQ tests / all CQ tests per release	CI logs
CR lead time	CR opened → released (median); cf. "lead time for changes"	Issue tracker
Release failure rate	releases needing a hotfix or rollback / all releases	Release log
Mapping coverage	mapped source columns / in-scope source columns	Mapping repo
Duplicate-concept rate	CRs closed as "already exists" (reuse found late)	Issue tracker

93.3 Coverage query (SPARQL)

```
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX skos:
<http://www.w3.org/2004/02/skos/core#>
SELECT (COUNT(DISTINCT ?c) AS ?classes)
     (COUNT(DISTINCT ?d) AS ?defined)
WHERE {
  ?c a owl:Class .
  FILTER(STRSTARTS(STR(?c),
        "https://ontology.example.com/"))
  OPTIONAL { ?c skos:definition ?x .
           BIND(?c AS ?d) }
}
```

93.4 Rules for honest metrics

- Baseline first: measure one quarter before setting any target.
- Report trends, not single numbers.
- Avoid vanity counts (triples, classes); bigger is not better.
- Pair every speed metric with a quality metric.
- Value stories (a question answered that could not be before) beat any ratio; record them with evidence.

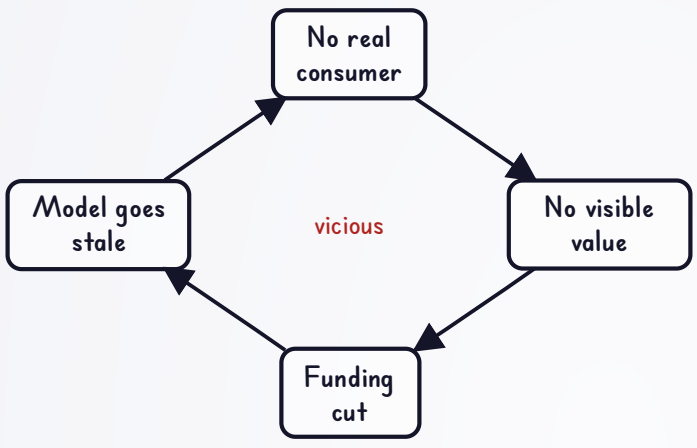
Key Takeaway: Measure adoption and trust, not size. Every KPI above is computed from artefacts you already have: the ontology, CI logs, the issue tracker and the schema registry.

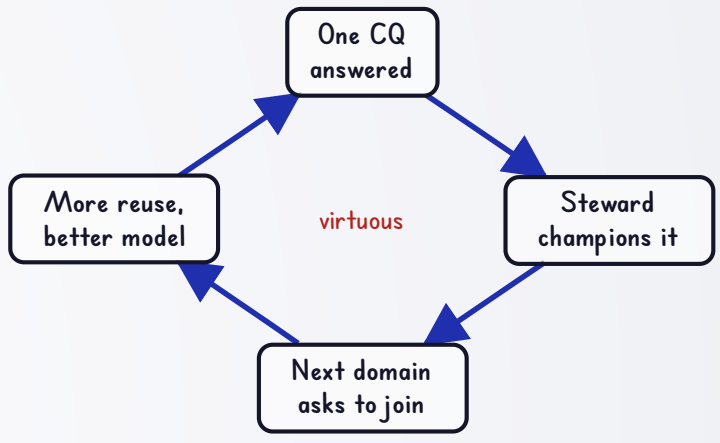
Sources: Basili, Caldiera & Rombach, GQM (1994) · Forsgren, Humble & Kim, Accelerate (2018) · W3C SPARQL 1.1 Query Language (2013)

94 Why Programs Fail

Part H · Governance, Coordination & Best Practices

94.1 Two cycles





Hepp (2007) argued that social and economic forces, not just technology, limit which useful ontologies get built and kept alive. Plan for those forces from day one.

94.2 Failure modes, warning signs, prevention

Failure mode	Warning sign	Prevention
Boil the ocean	Enterprise-wide model before one consumer	Start from one department's CQs
Ivory tower	Business cannot read the model	Stewards own definitions; HTML docs, diagrams
Tool first	Platform bought before use case	Use case + CQs, then tool categories
Canonical "god model"	Every field of every system in one schema	Canonical covers shared concepts only; bounded contexts keep the rest
Meaning without data	Classes with no mappings or instances	Mapping sprint in every onboarding
Governance bottleneck	CR queue grows; teams bypass	Delegate editorial/additive changes; automate gates
Identity chaos	owl:sameAs used for "similar"	Identifier policy; skos:closeMatch for fuzzy links
Orphan after go-live	No owner, no releases	Product owner + funded roadmap

94.3 Early-warning check (monthly)

- Can you name the systems that consumed the last release?
- Did a business steward approve a change this month?
- Is any CR older than your agreed service level?
- Did any team build a private copy of a shared concept?

94.4 Recovery playbook

1. **Freeze scope** : to one consumer and its CQs
2. **Re-name owners** : product owner + steward
3. **Ship** : one release that a running API uses
4. **Show** : the answered question to the sponsor

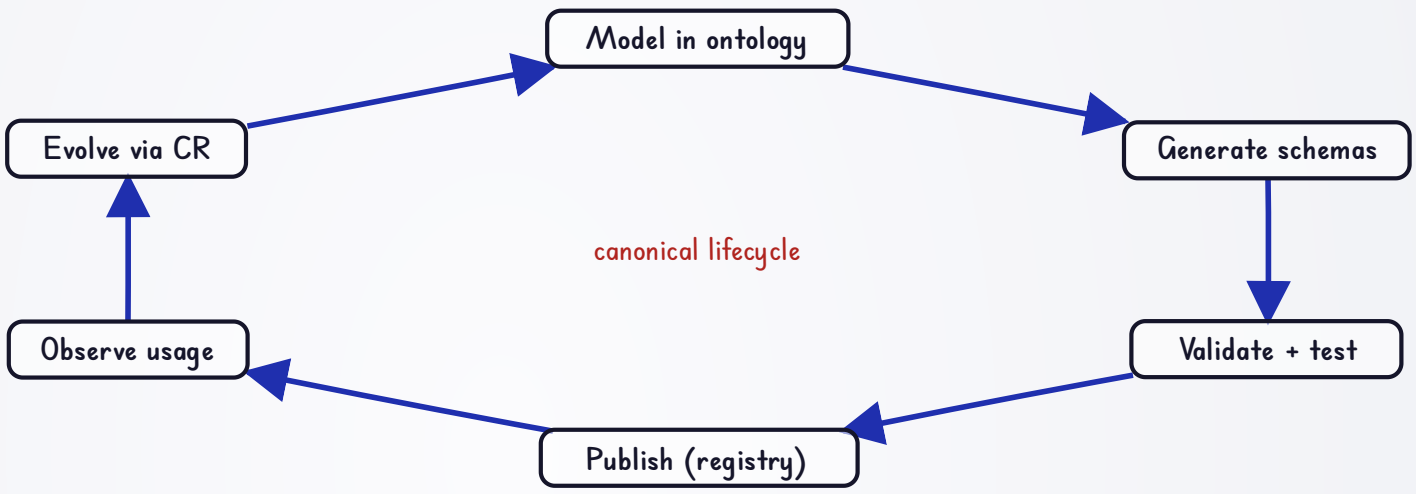
Key Takeaway: Most failures are organisational. Tie the ontology to a consumer, a steward and a budget line, and review these warning signs at every roadmap review.

Sources: Hepp, IEEE Internet Computing 11(1) (2007) · Hohpe & Woolf, EIP (2003): Canonical Data Model · Evans, Domain-Driven Design (2003) · Halpin et al., ISWC (2010)

95 Best Practices: Canonical

Part H · Governance, Coordination & Best Practices

95.1 The loop every canonical lives in



95.2 Scope & design

- ☐ Canonical covers **shared** concepts only (Party, Policy, Claim), not every system field.
- ☐ Every entity and field traces to an ontology IRI.
- ☐ Envelope (id, type, time, source, version) separate from body.
- ☐ Units, currencies, time zones explicit (fnd:Money, ISO 8601).
- ☐ No system-specific codes in the body; use SKOS code lists.
- ☐ Business keys and enterprise IRIs both carried.

95.3 Generation & validation

- ☐ JSON Schema / Avro / OpenAPI generated, never hand-edited.
- ☐ Generated artefacts checked into a release, with the ontology version.
- ☐ SHACL shapes validate sample payloads in CI.
- ☐ Example messages kept as test fixtures.

95.4 Versioning & compatibility

- ☐ SemVer on the canonical; breaking = major.
- ☐ Registry compatibility mode chosen per topic.
- ☐ Deprecation window published before removal.
- ☐ Old and new versions run side by side during migration.

95.5 Governance & adoption

- ☐ Owner per entity; changes only via CR.
- ☐ Extensions in a named namespace, reviewed regularly.
- ☐ Translation (anti-corruption) layers owned by the source team.
- ☐ Lint rule in API PRs; waivers expire.
- ☐ Consumers known from the registry, notified on release.

95.6 Before every publish

- ☐ Diff against last release reviewed.
- ☐ Release notes list breaking items first.
- ☐ Steward sign-off recorded.

Key Takeaway: Treat the canonical as a **generated product** of the ontology: if a field cannot be traced to a term, it is not canonical yet.

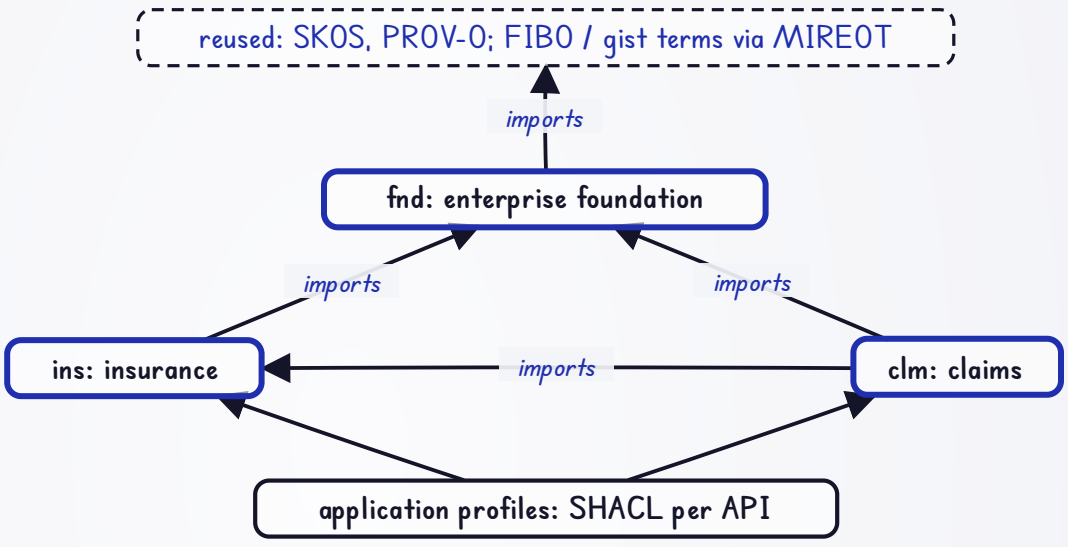
Sources: Hohpe & Woolf, EIP (2003) · JSON Schema Draft 2020-12 · Semantic Versioning 2.0.0 · Confluent docs: Schema Evolution and Compatibility

Rajesh Kumar Gupta

96 Best Practices: Ontology

Part H · Governance, Coordination & Best Practices

96.1 Module import graph to aim for



96.2 Requirements & scope

- ☐ Every module starts from written competency questions.
- ☐ CQs become SPARQL tests in CI.
- ☐ Scope statement lists what is **out** of scope.

96.3 Reuse & alignment

- ☐ Search standards before minting (FIBO, gist, SKOS, PROV-O).
- ☐ Import whole modules only if you accept their axioms; else MIREOT-style term extraction.
- ☐ Alignment axioms kept in a separate bridge file.

96.4 Naming & IRIs

- ☐ Stable, persistent IRIs under one base you control.
- ☐ Classes UpperCamel, properties lowerCamel, labels in plain language.
- ☐ owl:versionIRI on every release.

96.5 Definitions

- ☐ Every class and property has one skos:definition.
- ☐ Genus + differentia; no circular definitions.
- ☐ Steward approval recorded per term.

96.6 Axioms, shapes, reasoning

- ☐ OWL for meaning, SHACL for data rules; do not mix the two jobs.
- ☐ Reasoner run on every PR: no unsatisfiable classes.
- ☐ Choose a profile (EL / QL / RL) knowingly.

96.7 Release & operations

- ☐ Ontology-as-code: Git, PR review, CI (e.g. ROBOT, SHACL, CQ tests).
- ☐ owl:deprecated before any removal.
- ☐ Release notes, diagrams and HTML docs generated per release.
- ☐ Metrics trended every release.

Key Takeaway: Small, modular, tested and owned beats big and complete. Ship the first module when its CQs pass, then grow by onboarding the next department.

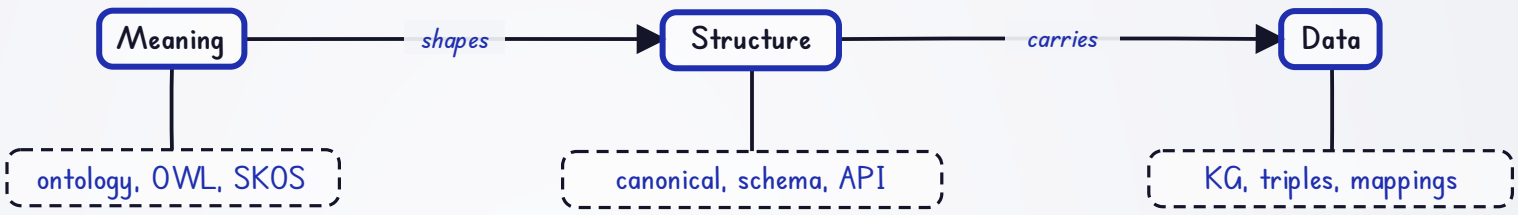
Sources: Noy & McGuinness, Ontology Development 101 (2001) · OBO Foundry Principles; Courtot et al., MIREOT (2011) · W3C Cool URIs for the Semantic Web (2008) · Jackson et al., ROBOT (2019)

Rajesh Kumar Gupta

97 Master Glossary

Part H · Governance, Coordination & Best Practices

97.1 Three families of terms



97.2 A – Z

ABox – assertions about individuals (instance data).

ADR – architecture decision record: context, decision, status, consequences.

Annotation property – OWL property for documentation; no logical effect.

Business glossary – governed list of business terms and definitions.

Canonical data model – shared, application-independent message/data format (Hohpe & Woolf).

Class – set of individuals sharing a meaning, e.g. clm:Claim.

Closed world (CWA) – what is not stated is false; databases, SHACL validation.

Code list – controlled set of values; here a SKOS concept scheme.

Competency question (CQ) – question the ontology must answer; becomes a test.

Conceptual model – business-level model of things and relations, no tech detail.

Data element – ISO/IEC 11179: object class + property + value domain.

Data steward – person accountable for quality and mapping of data.

Design Authority – board deciding cross-domain and breaking changes.

Domain steward – business owner of meaning within one module.

Entity resolution – deciding which records denote the same real thing.

GraphRAG – retrieval-augmented generation using a graph as context.

Individual – instance of a class, e.g. one claim.

IRI – internationalized resource identifier; global name of a term.

JSON-LD – W3C JSON syntax for linked data via @context.

Knowledge graph – graph of entities and relations, typed by an ontology.

Labeled property graph – nodes and edges with key-value properties (e.g. GQL).

Mapping (R2RML/RML) – rules that turn source data into RDF.

MIREOT – importing only the external terms you need.

Named graph – set of triples with its own IRI; for provenance, access.

Object / data property – links individual to individual / to a literal value.

Ontology – explicit specification of a conceptualization (Gruber 1993).

Open world (OWA) – missing facts are unknown, not false; OWL reasoning.

OWL 2 – W3C ontology language; profiles EL, QL, RL.

RACI – responsible, accountable, consulted, informed matrix.

RDF – W3C data model of subject-predicate-object triples.

Reasoner – engine that infers facts and checks consistency (HermiT, ELK).

SemVer – MAJOR.MINOR.PATCH versioning; major = breaking.

SHACL – W3C language for validating RDF data against shapes.

SKOS – W3C vocabulary for thesauri, taxonomies, code lists.

SPARQL – W3C query language for RDF.

Subclass (rdfs:subClassOf) – every member of A is a member of B.

TBox – terminology: classes, properties, axioms.

Triplestore – database for RDF triples, queried with SPARQL.

Upper ontology – domain-neutral top level, e.g. BFO, DOLCE, gist.

Value domain – ISO/IEC 11179: permitted values of a data element.

Note: Definitions here are short working glosses for this notebook; the defining source is cited on the page where each term is introduced.

Sources: Gruber (1993) · ISO/IEC 11179-1:2023 · W3C OWL 2, SKOS, SHACL, SPARQL 1.1 Recommendations · Hohpe & Woolf (2003)

Rajesh Kumar Gupta

98 References I

Part H · Governance, Coordination & Best Practices

All 139 works cited on the Sources lines of this notebook, numbered once. Page 1: standards & specifications.

W3C Recommendations, Notes & Reports

1. W3C (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission.
2. W3C (2005). Representing Specified Values in OWL: "value partitions" and "value sets" (Rector, ed.). Working Group Note.
3. W3C (2006). Defining N-ary Relations on the Semantic Web (Noy & Rector, eds.). Working Group Note.
4. W3C (2008). Cool URIs for the Semantic Web (Sauermann & Cyganiak, eds.). Interest Group Note.
5. W3C (2009). SKOS Simple Knowledge Organization System Reference (Miles & Bechhofer, eds.). Recommendation. SKOS Primer (Isaac & Summers, eds.), Working Group Note.
6. W3C (2012). OWL 2 Web Ontology Language: Document Overview, Primer, Structural Specification and Profiles (Second Edition). Recommendations.
7. W3C (2012). R2RML: RDB to RDF Mapping Language (Das, Sundara & Cyganiak, eds.). Recommendation.
8. W3C (2013). PROV-O: The PROV Ontology (Lebo, Sahoo & McGuinness, eds.). Recommendation.
9. W3C (2013). SPARQL 1.1 Query Language (Harris & Seaborne, eds.), SPARQL 1.1 Update and SPARQL 1.1 Federated Query. Recommendations.
10. W3C (2014). RDF 1.1 Concepts and Abstract Syntax (Cyganiak, Wood & Lanthaler, eds.). Recommendation.
11. W3C (2014). RDF 1.1 Semantics, RDF 1.1 Turtle, RDF 1.1 TriG and RDF Schema 1.1. Recommendations.
12. W3C (2015). Linked Data Platform 1.0 (Speicher, Arwe & Malhotra, eds.). Recommendation.
13. W3C (2016). Data on the Web Best Practices: Data Quality Vocabulary (DQV) (Albertoni & Isaac, eds.). Working Group Note.
14. W3C (2017). Data on the Web Best Practices. Recommendation.
15. W3C (2017). Shapes Constraint Language (SHACL) (Knublauch & Kontokostas, eds.). Recommendation; SHACL Advanced Features, Working Group Note.
16. W3C (2018). ODRL Information Model 2.2 (Iannella & Villata, eds.). Recommendation. Also ODRL Vocabulary & Expression 2.2.
17. W3C (2019). The Profiles Vocabulary (PROF) (Atkinson & Car, eds.). Working Group Note.
18. W3C (2020). JSON-LD 1.1: A JSON-based Serialization for Linked Data; JSON-LD 1.1 Processing Algorithms and API. Recommendations.
19. W3C (2024). Data Catalog Vocabulary (DCAT) – Version 3. Recommendation.
20. W3C Data Privacy Vocabularies and Controls CG (2024). Data Privacy Vocabulary (DPV), Version 2.0. Final Community Group Report (later versions at w3id.org/dpv).
21. W3C (2026). RDF 1.2 Concepts and Abstract Data Model. Candidate Recommendation Snapshot (Apr 2026) – check current status at w3.org.
22. W3C (2026). RDF 1.2 Turtle; SPARQL 1.2 Query Language; SHACL 1.2 Core. Working Drafts – check current status at w3.org.
23. W3C Knowledge Graph Construction CG (2026). RML-Core. Draft Community Group Report.

ISO / IEC

24. ISO 704:2022. Terminology work – Principles and methods.
25. ISO 1087:2019. Terminology work and terminology science – Vocabulary.
26. ISO 3166-1:2020. Codes for the representation of names of countries and their subdivisions – Part 1: Country code.
27. ISO 4217:2015. Codes for the representation of currencies.
28. ISO 8601-1:2019. Date and time – Representations for information interchange – Part 1: Basic rules.
29. ISO 17442-1:2020. Financial services – Legal entity identifier (LEI) – Part 1: Assignment. See gleif.org.
30. ISO 20022. Financial services – Universal financial industry message scheme. iso20022.org.
31. ISO/IEC 9075-16:2023. Information technology – Database languages SQL – Part 16: Property Graph Queries (SQL/PGQ).
32. ISO/IEC 11179-1:2023. Information technology – Metadata registries (MDR) – Part 1: Framework.
33. ISO/IEC 21838-1:2021 and 21838-2:2021. Top-level ontologies (TLO) – Part 1: Requirements; Part 2: Basic Formal Ontology (BFO).
34. ISO/IEC 21838-3:2023. Top-level ontologies (TLO) – Part 3: Descriptive ontology for linguistic and cognitive engineering (DOLCE).
35. ISO/IEC 25012:2008. Software engineering – SQuaRE – Data quality model.
36. ISO/IEC 39075:2024. Information technology – Database languages – GQL.

OMG · IETF · NIST · ANSI

37. ANSI/X3/SPARC Study Group on Data Base Management Systems (1975). Interim report. *FDT – Bulletin of ACM SIGMOD*, 7(2).
38. Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R. & Scarfone, K. (2014). *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. NIST SP 800-162.
39. IETF (2002). RFC 3339: Date and Time on the Internet: Timestamps (Klyne & Newman).
40. IETF (2022). RFC 9110: HTTP Semantics (Fielding, Nottingham & Reschke, eds.). STD 97.
41. OMG (2014). *Ontology Definition Metamodel (ODM)*, Version 1.1.
42. OMG (2017). *Unified Modeling Language (UML)*, Version 2.5.1.

Industry & community specifications

43. ACORD. *ACORD Standards* (insurance data & messaging standards). acord.org.
44. Apache Software Foundation. *Apache Avro Specification*, Version 1.12.0. avro.apache.org.
45. CDISC. *CDISC Foundational Standards*. cdisc.org.
46. CNCF. *CloudEvents Specification*, Version 1.0.2. cloudevents.io.
47. DCMI Usage Board (2020). *DCMI Metadata Terms*. Dublin Core Metadata Initiative.
48. EDM Council. *Financial Industry Business Ontology (FIBO)*. spec.edmcouncil.org/fibo; github.com/edmcouncil/fibo.
49. GraphQL Specification Project (2025). *GraphQL*, September 2025 Edition. spec.graphql.org.
50. HL7 International (2023). *FHIR Release 5*. hl7.org/fhir.
51. JSON Schema. *JSON Schema Draft 2020-12* (Core and Validation). json-schema.org.
52. OAGi. *Open Applications Group Integration Specification (OAGIS)*. oagi.org.
53. OBO Foundry. *OBO Foundry Principles*. obofoundry.org.
54. OpenAPI Initiative (2021). *OpenAPI Specification*, Version 3.1.0; Version 3.2.0 (2025).
55. Preston-Werner, T. *Semantic Versioning 2.0.0*. semver.org.
56. Semantic Arts. *gist* (minimalist upper ontology for the enterprise). semanticarts.com.

Rajesh Kumar Gupta

99 References II

Part H · Governance, Coordination & Best Practices

Page 11: books & peer-reviewed papers, authors A–L.

Books & papers (A–L)

57. Ackoff, R. L. (1989). From data to wisdom. *Journal of Applied Systems Analysis*, 16, 3–9.

58. Allemang, D., Hendler, J. & Gandon, F. (2020). *Semantic Web for the Working Ontologist*, 3rd ed. ACM Books.

59. Arp, R., Smith, B. & Spear, A. D. (2015). *Building Ontologies with Basic Formal Ontology*. MIT Press.

60. Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F. (eds.) (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

61. Basili, V. R., Caldiera, G. & Rombach, H. D. (1994). The Goal Question Metric approach. In *Encyclopedia of Software Engineering*. Wiley.

62. Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34–43.

63. Borgo, S., Ferrario, R., Gangemi, A., Guarino, N., Masolo, C., Porello, D., Sanfilippo, E. M. & Vieu, L. (2022). DOLCE: A descriptive ontology for linguistic and cognitive engineering. *Applied Ontology*, 17(1), 45–69.

64. Borst, W. N. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente.

65. Calvanese, D. et al. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3), 471–487.

66. Carroll, J. J., Bizer, C., Hayes, P. & Stickler, P. (2005). Named graphs, provenance and trust. *Proc. WWW 2005*, ACM, 613–622.

67. Chen, P. P. (1976). The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9–36.

68. Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G. & Stefanidis, K. (2020). An overview of end-to-end entity resolution for big data. *ACM Computing Surveys*, 53(6).

69. Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387.

70. Courtot, M. et al. (2011). MIREOT: The minimum information to reference an external ontology term. *Applied Ontology*, 6(1), 23–33.

71. DAMA International (2017). *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed. Technics Publications.

72. Dehghani, Z. (2020). Data mesh principles and logical architecture. martinfowler.com, 3 Dec 2020.

73. Dehghani, Z. (2022). *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly.

74. Dimou, A. et al. (2014). RML: A generic language for integrated RDF mappings of heterogeneous data. *LDOW 2014*, CEUR-WS Vol. 1184.

75. Edge, D. et al. (2024). From local to global: A Graph RAG approach to query-focused summarization. arXiv:2404.16130.

76. Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

77. Fellegi, I. P. & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328), 1183–1210.

78. Fernández-López, M., Gómez-Pérez, A. & Juristo, N. (1997). METHONTOLOGY: From ontological art towards ontological engineering. *AAAI Spring Symposium on Ontological Engineering*.

79. Forsgren, N., Humble, J. & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution.

80. Fowler, M. (2014). BoundedContext. *bliki*, martinfowler.com, 15 Jan 2014.

81. Glimm, B., Horrocks, I., Motik, B., Stoilos, G. & Wang, Z. (2014). HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3), 245–269.

82. Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199–220.

83. Grüninger, M. & Fox, M. S. (1995). Methodology for the design and evaluation of ontologies. *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*.

84. Guarino, N. (1998). Formal ontology and information systems. *Proc. FOIS '98*, IOS Press, 3–15.

85. Guarino, N. & Welty, C. (2002). Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2), 61–65.

86. Guarino, N. & Welty, C. A. (2004). An overview of OntoClean. In *Handbook on Ontologies*, Springer, 151–171 (2009 edition: 201–220).

87. Halpin, H., Hayes, P. J., McCusker, J. P., McGuinness, D. L. & Thompson, H. S. (2010). When owl:sameAs isn't the same: An analysis of identity in Linked Data. *ISWC 2010*, LNCS 6496, 305–320.

88. Hepp, M. (2007). Possible ontologies: How reality constrains the development of relevant ontologies. *IEEE Internet Computing*, 11(1), 90–96.

89. Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A. & Presutti, V. (eds.) (2016). *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. IOS Press.

90. Hogan, A. et al. (2021). Knowledge graphs. *ACM Computing Surveys*, 54(4), Article 71.

91. Hohpe, G. & Woolf, B. (2003). *Enterprise Integration Patterns*. Addison-Wesley. Canonical Data Model pattern: enterpriseintegrationpatterns.com.

92. Horridge, M. (2011). *Justification Based Explanation in Ontologies*. PhD thesis, University of Manchester.

93. Jackson, R. C. et al. (2019). ROBOT: A tool for automating ontology workflows. *BMC Bioinformatics*, 20, 407.

94. Kalyanpur, A., Parsia, B., Horridge, M. & Sirin, E. (2007). Finding all justifications of OWL DL entailments. *ISWC 2007*, LNCS 4825, 267–280.

95. Kazakov, Y., Krötzsch, M. & Simančík, F. (2014). The incredible ELK. *Journal of Automated Reasoning*, 53(1), 1–61.

96. Keet, C. M. & Ławrynowicz, A. (2016). Test-driven development of ontologies. *ESWC 2016*, LNCS 9678, 642–657.

97. Kendall, E. F. & McGuinness, D. L. (2019). *Ontology Engineering*. Synthesis Lectures on the Semantic Web. Morgan & Claypool.

98. Kimball, R. & Ross, M. (2013). *The Data Warehouse Toolkit*, 3rd ed. Wiley.

99. Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly.

100. Lassila, O. & McGuinness, D. L. (2001). The role of frame-based representation on the Semantic Web. Stanford KSL Technical Report KSL-01-02.

101. Linstedt, D. & Olschimke, M. (2015). *Building a Scalable Data Warehouse with Data Vault 2.0*. Morgan Kaufmann.

Rajesh Kumar Gupta

100 References III

Part H · Governance, Coordination & Best Practices

Page 111: books & papers M–Z, then tools & documentation.

Books & papers (M–Z)

102. Moxon, S. A. T. et al. (2021). The Linked Data Modeling Language (LinkML): A general-purpose data modeling framework grounded in machine-readable semantics. *ICBO 2021*, CEUR-WS Vol. 3073.

103. Musen, M. A. (2015). The Protégé project: A look back and a look forward. *AI Matters*, 1(4), 4–12.

104. Neumann, T. & Weikum, G. (2008). RDF-3X: A RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1), 647–659.

105. Noy, N. F. & McGuinness, D. L. (2001). Ontology Development 101: A guide to creating your first ontology. Stanford KSL Technical Report KSL-01-05.

106. Nygard, M. (2011). Documenting architecture decisions. Cognitect blog, Nov 2011.

107. Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J. & Wu, X. (2024). Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7), 3580–3599.

108. PMI (2017). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 6th ed. Project Management Institute.

109. Poveda-Villalón, M., Fernández-Izquierdo, A., Fernández-López, M. & García-Castro, R. (2022). LOT: An industrial oriented ontology engineering framework. *Engineering Applications of Artificial Intelligence*, 111, 104755.

110. Poveda-Villalón, M., Gómez-Pérez, A. & Suárez-Figueroa, M. C. (2014). OOPS! (OntOlogy Pitfall Scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems*, 10(2), 7–34.

111. Robinson, I. (2006). Consumer-driven contracts: A service evolution pattern. martinfowler.com, 12 Jun 2006.

112. Seiner, R. S. (2014). *Non-Invasive Data Governance*. Technics Publications.

113. Seppälä, S., Ruttenberg, A. & Smith, B. (2017). Guidelines for writing definitions in ontologies. *Ciência da Informação*, 46(1).

114. Sequeda, J., Allemang, D. & Jacob, B. (2023). A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL databases. arXiv:2311.07509.

115. Singhal, A. (2012). Introducing the Knowledge Graph: things, not strings. Google Official Blog, 16 May 2012.

116. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A. & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2), 51–53.

117. Skelton, M. & Pais, M. (2019). *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution.

118. Smith, B. et al. (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25(11), 1251–1255.

119. Studer, R., Benjamins, V. R. & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1–2), 161–197.

120. Suárez-Figueroa, M. C., Gómez-Pérez, A. & Fernández-López, M. (2012). The NeOn methodology for ontology engineering. In *Ontology Engineering in a Networked World*. Springer.

121. Tartir, S., Arpinar, I. B., Moore, M., Sheth, A. P. & Aleman-Meza, B. (2005). OntoQA: Metric-based ontology quality analysis. *IEEE ICDM Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*.

122. Uschold, M. & Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 93–136.

123. Uschold, M. & King, M. (1995). Towards a methodology for building ontologies. *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*.

124. Vrandečić, D. (2009). Ontology evaluation. In *Handbook on Ontologies*, Springer, 293–313.

125. Zaveri, A. et al. (2016). Quality assessment for Linked Data: A survey. *Semantic Web*, 7(1), 63–93.

Tools, documentation & web resources

126. Confluent. *Schema Evolution and Compatibility*, Schema Registry documentation. docs.confluent.io.

127. GitHub. *About code owners* (CODEOWNERS file), GitHub Docs. docs.github.com.

128. HyperGraphQL. GraphQL interface for querying and serving linked data. github.com/hypergraphql/hypergraphql.

129. Kimball Group. *Four-Step Dimensional Design Process*. kimballgroup.com.

130. LinkML. *LinkML documentation* (generators: JSON Schema, JSON-LD context, GraphQL). linkml.io.

131. Neo4j. *Cypher Manual*; *Neo4j GraphQL Library* documentation. neo4j.com/docs.

132. Ontology Design Patterns portal. ontologydesignpatterns.org.

133. OOPS! (OntOlogy Pitfall Scanner!). *Pitfall Catalogue*. oops.linkeddata.es/catalogue.jsp.

134. PyLD. JSON-LD processor written in Python. github.com/digitalbazaar/pyld.

135. RDFLib. *rdflib* (Python RDF library) and *pySHACL* (SHACL validator). github.com/RDFLib.

136. RML. *RDF Mapping Language* specification and resources. rml.io.

137. ROBOT. *ROBOT documentation* (merge, reason, report, verify, diff, explain, measure…). robot.obolibrary.org.

138. Stoplight. *Spectral* open-source API linter, rulesets documentation. github.com/stoplightio/spectral.

139. W3C. *Standards and drafts* index. w3.org/TR.

Standards are cited by edition; check w3.org / iso.org for current status.

Rajesh Kumar Gupta