GRAPH RESEARCH LABS

AI-Assisted Ontology Engineering – Methodology and Prompt Library

Graph Research Labs

2026

AI-Assisted Ontology Engineering Workshop

DATE

June 25, 2026

Dougal Watt
Graph Research Labs

www.graphresearchlabs.com

How to use this handout

This handout contains the methodology, the prompt artefacts for the worked example, a production readiness checklist, and a lightweight conformance template you can use to model an ontology with an LLM.

The methodology itself is portable. It works with Claude, with GPT-class models, with Gemini, and with any other frontier LLM you can access, including open-weight open source models. The prompts in this handout are written for general-purpose models. Where a prompt benefits from a particular feature (longer context, tool use, structured output), the prompt notes that explicitly.

Three things to remember

1. LLMs do not generate ontologies. They assist human ontologists. The methodology assumes the human is in control and directing the LLM, which is assisting.
2. Every ontology fix should trigger a resync. Change a class, regenerate the SHACL. Change a property, re-run the competency questions. The discipline of keeping downstream artefacts in sync is a key element that separates this methodology from ad hoc LLM use.
3. Adversarial critique is non-negotiable. LLMs are sycophantic and will confidently execute changes that may cause downstream problems. The only reliable way to surface weaknesses in their output is to set up an adversarial pass such as a sceptical reviewer, a domain expert, a regulator, or a first-principles thinker. This critique probes for failure and helps you iterate towards a quality outcome. LLMs have been shown to produce considerable improvements in outputs when asked to check their work in this manner.

How this handout is organised

- Page 3: the methodology in one page, plus the five key principles.
- Page 4: when NOT to use the LLM. Read this before you do anything else.
- Page 5: our worked example ontology.
- Pages 6 to 19: the seven prompt artefacts. Each artefact has a purpose statement, the prompt text in a copy-pasteable box, notes on variations, and a short example interaction.
- Pages 20 to 22: the worked example: applying the methodology end-to-end to a cross-border payments and AML ontology.
- Page 23: the lightweight conformance template, to be used as your in-workshop scoring sheet.
- Page 24: where to go next.

Help and automation

Graph Research Labs has a range of tools to help automate your ontology and knowledge graph programmes:

- GRL OntoLinter (basic) - free linter to check ontology quality.
- GRL OntoLinter (advanced) – tool with advanced quality checks for ontologies.
- GRL Ontology Manager – create, edit, govern and publish ontologies for teams.
- GRL Generators – connect instantly to a semantic graph database, creates instant APIs, React apps, MPC servers, data products and governed agents from an ontology – useful for testing and deployment into production environments as well as to create stand alone data assets quickly.
- GRL Semantic Agent harness – creates governed agents to extend agent accuracy and compliance.
- GRL Governance Metagraph – provides centralised view of all provenance and governance data from GRL tools.

The GRL Agentic Ontology Engineering methodology

The methodology has seven phases that take an ontology from initial framing through to production-readiness. The arc is iterative: you can re-enter at any phase as the ontology evolves. It is also modular, as each phase works as a standalone activity if that is all you have time for.

Phase	Purpose	Human's job	LLM's job
1. Review	Review the initial draft ontology (human or LLM created).	Human review of initial ontology version.	*Summarise, explain.*
2. Define	LLM generates ontology competency questions (CQs) to test your ontology.	Guide and shape CQs based on industry/company knowledge.	*Generate, expand, and clarify CQs.*
3. Critique	Surface weaknesses adversarially before they reach the assess step.	Train & configure agents with the adversarial perspectives.	*Play the antagonist roles.*
4. Assess	Apply the human-decided fix to the ontology.	Decide what to change and apply it.	*Pair-modeller dialogue; sanity-check.*
5. Validate	Use tooling such as reasoner, linters and CQ tests to validate ontology changes.	Run the tools, assess outputs.	*Generate or regenerate SHACL shapes and/or CQs.*
6. Conform	Score against the conformance criteria.	Read the report; decide what to fix next.	*Summarise findings; suggest priorities.*
7. Release	Sign-off, version, document, release.	Sign-off and decision documentation.	*Generate change-log entries; documentation.*

Five key principles

1. **Augment, not generate.** The LLM is an assistant; it does not author the ontology. The human ontologist remains responsible for every axiom that ends up in the published artefact.
2. **Patterns and focus.** LLMs excel at applying known patterns to new material and at focusing on narrow, well-formed questions. They struggle with unconstrained generation. Provide the pattern and the focus, and let the LLM apply these.
3. **Always validate after change.** Every edit, whether yours or the LLM's, must be followed by a reasoner run, a SHACL run, a lint run, and a CQ test re-run. The validation phase is what makes the methodology trustworthy.
4. **Sync downstream artefacts.** An ontology rarely lives alone. SHACL shapes, JSON-LD contexts, competency questions, and documentation — every dependent artefact must be regenerated or updated whenever the ontology changes. The fix-and-resync discipline is key to this methodology.
5. **Antagonistic agents elevate quality.** LLMs are sycophantic by default. Adversarial framings with sceptical reviewers, domain experts, regulators, and first-principles thinkers reliably surface weaknesses that direct review misses. Configure the antagonists then let them do their job.

When NOT to use the LLM

The methodology should be used to augment human-driven ontology engineering. There are areas where the LLM should be kept out of the loop entirely or where its output should be treated as suggestion only and heavily curated or rewritten by a human. These areas matter most when your ontology is critical for downstream reasoning, certification, regulation, or safety-critical decisions.

Six areas where the LLM should be curated, treated with extreme suspicion or excluded:

1. Foundational class structure. The upper-level distinctions of your ontology, particularly those that align to gist, BFO, DOLCE, or another upper ontology, should be human-authored, and the upper-ontology classes themselves should be reused by IRI rather than redefined. LLM suggestions here are confidently and consistently wrong and propagate downward through everything.
2. Equivalence axioms. owl:equivalentClass and owl:equivalentProperty declarations have profound reasoning consequences. The LLM cannot anticipate the downstream effects. Human authoring with reasoner verification is mandatory.
3. Anything critical for downstream reasoning. If a critical inference depends on an axiom, do not let the LLM author it. Examples: subsumption hierarchies that drive permission inheritance, property characteristics (transitivity, symmetry) that change closure, disjointness assertions that gate consistency.
4. Areas of legitimate domain disagreement. If your domain has an unsettled debate (for example, how to model events in clinical data, how to model time in financial transactions), the LLM will pick a side without telling you. Use antagonistic reasoning agents, and surface the debate to a human ontologist.
5. Naming of stable identifiers. IRI structure, prefix conventions, and identifier patterns should be standardised by the team and provided to the LLM as constraints, not negotiated each conversation. Inconsistent identifiers are the silent killer of ontology integration.
6. Anything safety or compliance-critical. If your ontology informs flight clearance, certification, regulatory reporting, clinical decisions, or financial settlements, the human-in-the-loop bar is higher and the audit trail must be complete. The LLM is an assistant in research and development; it is not a decision-maker in production. In these cases, a Semantic Agent Harness is a must-use part of the ontology development process, as this provides governed guardrails, conformance to regulatory requirements, long-term working / shared memory, and a clear, provable audit trail.

> *Practical rule. If you are about to commit an LLM-suggested axiom that you would not be comfortable defending in a peer review, do not commit it. Iterate further, or rewrite it yourself.*

Our Ontology example

Purpose

This description of our worked example ontology is intended to be used in our LLM prompting in the artefacts below. The ontology uses gist version 14 as the upper ontology, but the techniques listed in this document can be used with any upper ontology. They can also be used with upper and middle ontologies if your project is structured that way.

Description

The Cross-Border Payments and AML Ontology (CBPA) describes how a commercial or correspondent bank handles cross-border payments under AML and FIU oversight. It captures Customers and the Accounts they hold, the Banks at which Accounts are held (including correspondent banks that hold Nostro and Vostro arrangements for cross-border settlement), the physical Branches at which Banks operate, the time-bounded RiskAssessment of a Customer, the time-bounded ComplianceState of an Account, the BalanceSnapshot at a point in time, and the KYCReviewEvents that re-rate a Customer's risk over time.

Payments in CBPA are kept distinct from payment instructions. A PaymentInstruction is the planning artefact: it is authorised by a Customer or signatory, has a requested settlement window, declares a purpose (trade settlement, family remittance, salary, investment), requests a settlement channel (SWIFT, RTGS, CHIPS, SEPA, ACH), carries a lifecycle status (drafted, authorised, released, cancelled, superseded), and instructs a magnitude (the amount denominated in a currency). A Payment is the occurrence of an instruction: it has actual payer and payee Accounts, an actual time interval, actually settles through a Channel, optionally has an outcome (cleared, rejected, returned, under investigation), and carries an end-to-end payment reference. A PaymentInstruction should be linked to its Payment, and a Payment optionally references the instruction it implements. Instructions may exist without occurrences (they were drafted but never released); occurrences typically reference an instruction but may be improvised settlement legs through a correspondent bank. The ontology is intended to support AML monitoring, sanctioned-party screening, beneficiary verification, suspicious activity reporting, cross-border settlement reconciliation, and the audit-trail questions a regulator (such as the Korean Financial Supervisory Service or the Financial Intelligence Unit) would ask of any of these.

Artefact 1 — Competency question generation

Purpose

Generate ten to twelve competency questions that define what your ontology should be able to answer. Note that for large ontologies, you may need to generate more. Competency questions are the contract between the ontology and the application; they make the ontology testable and they give a definition of “done” that can be measured by you and your LLM.

When to use

At the start of every modelling session. For greenfield ontologies, before any class definition. For existing ontologies, as a structured way to surface the implicit purpose so it can be made explicit.

The prompt

```
You are an expert ontology engineer experienced in gist, BFO, and domain-specific
upper ontologies. Reuse upper-ontology classes by IRI; do not redefine them. Your
task is to generate competency questions for the ontology I am about to describe.

Competency questions must be:
- Measurable: each question must have a clear pass/fail answer condition.
- Scoped: each question must be answerable using only the ontology and instance data.
- Useful: each question must reflect a real use case — something a downstream
application or a domain user would actually ask.
- Distinct: no two questions should ask substantially the same thing.

For each competency question, provide:
1. The question in plain language.
2. The use case it serves.
3. The ontology elements (classes, properties) that would need to exist for it to be
answerable.
4. A draft SPARQL skeleton (do not worry about correctness; sketch the shape).

Generate 10 to 12 competency questions covering the breadth of the domain. Avoid
lookup-style questions (e.g. "what is the label of X"); favour questions that
exercise structural inference, classification, or aggregation.

DOMAIN DESCRIPTION:
<paste your domain description here>

Begin.
```

Notes and variations

1. If you get back lookup-style questions, add a constraint to the prompt: "No lookup questions. Every question must require at least two hops or a classification."
2. If you get back vague questions, add: "Each question must specify the answer type and the answer condition. ‘Find Payments that...’ is acceptable; ‘Tell me about Payments’ is not."
3. If your ontology is multilingual, add: "The working language is <language>. All questions must be expressible in <language>."
4. If you are working module-by-module on a large ontology, scope the prompt to one module at a time and reference the upstream modules by IRI.

Example interaction (excerpt)

Prompt (after pasting the CBPA domain description): Generate 10 to 12 CQs.

```
Response (CQ #3):

Q: Which Customers in a Bank's high-risk segment have at least one Payment in the
past 30 days denominated in USD and routed via SWIFT?

Use case: AML alert triage — the morning queue for the compliance officer.

Required elements: Customer, Account, Payment, RiskAssessment, RiskRating,
hasRiskAssessment, payer, denominatedIn, viaChannel.

SPARQL skeleton:
SELECT ?cust WHERE {
  ?cust a :Customer ; :hasRiskAssessment ?ra ; :holdsAccount ?acc .
  ?ra gist:isCategorizedBy :HighRisk .
  ?pay a :Payment ; :payer ?acc ; :denominatedIn :USD ; :viaChannel :SWIFT .
  ?pay gist:occursIn ?ti .
  # filter ?ti within the last 30 days
}
```

Artefact 2 — Adversarial critique skill

Purpose

Run any LLM-generated artefact (competency questions, class definitions, SHACL shapes, instance data) through four antagonistic perspectives that probe for weaknesses the LLM is unlikely to surface in direct review. The output is a list of weaknesses, each tied to a specific perspective, that you then either fix, accept as out of scope, or escalate to a domain expert.

When to use

After every generation step. After every fix. Before every commit. The adversarial pass is not optional in this methodology.

The prompt

```
You are going to critique the artefact below from four adversarial perspectives.
Treat each perspective independently. Do not soften your criticism. Your job is to
surface weaknesses, not to validate.

PERSPECTIVE 1 — Sceptical reviewer.
Assume the artefact is wrong somewhere. Find what is wrong. Report at least three
specific weaknesses. For each, name the weakness, explain why it matters, and propose
a fix.

PERSPECTIVE 2 — Domain expert.
Assume you are a senior practitioner in the relevant domain. Find what is domain-
naive i.e. what would a real practitioner spot as ignorant of how the work actually
happens? Report at least two specific weaknesses with the same structure as
Perspective 1.

PERSPECTIVE 3 — Regulator or auditor.
Assume you have to certify the ontology for use in a regulated domain (AML/FIU,
Korean Financial Supervisory Service, Basel III, EU PSD2 — pick the most relevant).
What would block certification? What is missing for audit, traceability, or
compliance? Report at least two weaknesses.

PERSPECTIVE 4 — First-principles thinker.
Strip away the assumptions. What is being taken for granted that should not be? Where
is the artefact applying a pattern by reflex when a different pattern would serve
better? Report at least two weaknesses.

After all four perspectives, give a final ranked list of the top five weaknesses
across all perspectives, ordered by severity. For each, indicate: severity
(high/medium/low), estimated effort to fix (small/medium/large), and whether it
should block release.

ARTEFACT TO CRITIQUE:
<paste the artefact here>

Begin with Perspective 1.
```

Notes and variations

1. Substitute perspectives based on context. For an academic ontology, swap "regulator" for "peer reviewer". For a startup, swap "regulator" for "maintainer who inherits this in three years".

2. If the LLM softens its critique, add to each perspective: "You are graded on weaknesses found, not on collegiality. A perspective that returns no weaknesses has failed its task."

3. Run each perspective in a separate fresh conversation agent. If you find that running them in one conversation produces less independent critique, use separate conversations or agents. The cost is one more paste; the gain is sharper output.

4. Always end with the ranked list. Without a ranking, the output is too long to act on. The ranking forces the LLM to commit to what matters most.

Example interaction (excerpt)

Output from Perspective 3 — regulator:

```
Weakness R1 (HIGH): The PaymentInstruction class has no provenance properties. In a
regulated context I would need to know who authored each instruction, when it was
approved, by what authority, and when it was superseded. Add prov:wasAttributedTo,
prov:generatedAtTime, and an explicit approval state.

Weakness R2 (MEDIUM): KYCReviewEvent has no link to a regulatory standard. For AML
compliance I need to traceably attribute every review to a certified compliance
officer and a specific regulatory standard (FATF Recommendation 10, Korean FSC's CDD
rules, etc.). Add reviewPerformedBy and reviewedUnderStandard.
```

Artefact 3 — SHACL generation

Purpose

Generate SHACL shapes that enforce the structural constraints implied by your competency questions and class definitions. The SHACL shapes are how you turn the ontology into a checkable contract, as every data instance datum is testable against the shapes.

When to use

Whenever a class or property changes. Whenever a competency question is added or modified. As part of the resync discipline in the Assess phase of the methodology.

The prompt

```
You are an expert in SHACL shape authoring. Your task is to generate SHACL shapes
that enforce the structural constraints implied by the class definitions and
competency questions provided below.

For each class, generate a NodeShape that:
- Targets the class.
- Declares the cardinality of every essential property (minCount, maxCount).
- Declares the value type or value class for every property (datatype, class).
- Declares any disjointness, equivalence, or value range constraints implied by the
class definition.
- Includes a sh:message in plain English for every constraint, suitable for surfacing
to a non-technical user.

For each competency question that implies a structural constraint (for example,
"every Payment must have at least one SettlementLeg"), generate a corresponding
PropertyShape or NodeShape.

Use the prefix sh: <http://www.w3.org/ns/shacl#> and assume the ontology prefix is :.
Output Turtle. Do not include the ontology declarations themselves; only the SHACL.
Order the output: NodeShapes first (one block per class), then standalone
PropertyShapes derived from competency questions.

CLASS DEFINITIONS:
<paste class definitions here, one per class, in the form: ClassName: description>

COMPETENCY QUESTIONS:
<paste your CQs here, one per line>

Begin.
```

Notes and variations

1. If you need closed shapes (no extra properties allowed beyond those declared), add to the prompt: "All NodeShapes must use sh:closed true with sh:ignoredProperties (rdf:type)."
2. If you need to enforce language tags on labels, add: "For every rdfs:label or skos:prefLabel, declare sh:languageIn (‘en’ ‘ko’ ...) with the working languages."
3. Always run the generated SHACL through a SHACL validator (e.g. pyshacl or topbraid) before commit. LLMs generate plausible-looking SHACL that occasionally fails strict validation. The validator catches this in seconds.
4. Do not let the LLM regenerate SHACL you have hand-curated. Provide hand-curated shapes as "locked" inputs and instruct the LLM to add to them, not replace them.

Example output (excerpt)

```
:PaymentShape a sh:NodeShape ;
  sh:targetClass :Payment ;
  sh:property [
    sh:path :payer ;
    sh:minCount 1 ; sh:maxCount 1 ;
    sh:class :Account ;
    sh:message "A Payment must have exactly one payer Account." ;
  ] ;
  sh:property [
    sh:path :viaChannel ;
    sh:minCount 1 ; sh:maxCount 1 ;
    sh:class :ChannelType ;
    sh:message "A Payment must be routed through exactly one Channel." ;
  ] .
```

Artefact 4 — Modeller dialogue (fix and resync)

Purpose

Discuss a specific modelling problem with the LLM as a sceptical pair-modeller. The output is an agreed fix that you apply to the ontology, plus a list of downstream artefacts that need to be regenerated. This is the prompt that drives the Validate and Assess phases of the methodology.

When to use

Once you have an audit finding (from the workshop linter or a reasoner inconsistency) that you have decided to fix. Use a fresh Claude conversation; do not pollute the dialogue with prior context.

The prompt

```
You are an expert ontology engineer with deep knowledge of OWL 2, gist, BFO, and
common modelling patterns. We are going to discuss a specific modelling problem in my
ontology. Your role is sceptical pair-modeller, not solver. You should:
- Ask clarifying questions before proposing fixes.
- Propose at least two distinct fix options whenever a fix is offered.
- For each option, name the trade-offs explicitly.
- Push back if I select a fix that has a less-good alternative; do not capitulate to
my preference if you think it is wrong.
- Once we agree on a fix, list every downstream artefact that needs to be regenerated
or updated as a result. Be specific: name the SHACL shape, the competency question,
the JSON-LD context, the API spec, the documentation page.

The problem:
<describe the problem in detail. Include: the violation as reported by the audit
tool, the affected classes or properties, the relevant CQs, and any constraints you
cannot violate.>

Current relevant ontology fragment (in Turtle):
<paste the relevant fragment, including upstream class definitions if any>

Begin by asking your clarifying questions. Do not propose a fix until you have asked
and I have answered.
```

Notes and variations

1. If the LLM proposes a fix without asking questions, respond with: "Stop. Ask the clarifying questions first. I will not accept a fix proposal without prior questioning."
2. If the LLM agrees with you too quickly, respond with: "You are being sycophantic. Argue the other side properly before accepting my position."
3. If you reach an impasse, ask: "Summarise the disagreement in one paragraph. I will take it to my colleagues for resolution." Do not let the LLM decide a contested point alone.
4. Always force the downstream-artefact list. If the LLM closes the conversation without listing what needs to be resynced, ask explicitly. The list is the deliverable.

Example interaction pattern

```
You:
Problem: The workshop linter reports GIST-002 (Plan/occurrence collapse) on
class :Payment. The violation is that :Payment is asserted as both gist:Event and
gist:Task.

Affected classes: :Payment, :SettlementLeg, :TestPayment.
Relevant CQs: CQ7 ("Which payment instructions were authorised but never settled?")
and CQ9 ("Which Payments cleared via SWIFT in May?").
Constraint: I cannot break the gist alignment.
Ontology fragment: [Turtle]
```

```
Response (LLM, before any fix proposal):
Before I propose anything, three clarifying questions:
1. Is :Payment intended to denote the act of executing the payment on a timeframe,
the plan for the payment, or both? In your CQs, CQ9 treats it as an occurrence and
CQ7 is about plans.
2. Does your domain require that a single Payment identifier persist across the
instruction phase, the settlement phase, and the reconciliation phase? Or are these
separate identifiers in your operational reality?
3. Are :SettlementLeg and :TestPayment intended as sub-occurrences of the payment
execution, or as alternative kinds of payment?

Answer these and I will propose at least two fix options.
```

Artefact 5 — Test data and CQ test generation

Purpose

Generate synthetic instance data and SPARQL test queries that exercise the competency questions. The output is a regression test suite for your ontology: one set of instance data that any change to the ontology should still be able to answer correctly.

When to use

After your competency questions have been adversarially critiqued and you have an agreed set. Re-run whenever the ontology changes; failing tests indicate either a bug in the ontology change or a CQ that needs updating.

The prompt

```
You are going to generate two artefacts: synthetic instance data and SPARQL test
queries derived from a set of competency questions.

For the synthetic instance data:
- Generate 25 to 40 instances spanning the classes referenced by the CQs.
- Ensure that for every CQ, there is at least one instance configuration that answers
"yes" and one that answers "no".
- Cover edge cases: empty collections, boundary values, instances missing optional
properties, instances with maximum cardinality.
- Use realistic-looking but obviously synthetic identifiers (e.g. :Payment_TST001 not
:Payment_MT103_REAL42).
- Output Turtle.

For the SPARQL test queries:
- Generate one query per CQ.
- For each query, also generate the expected result against the synthetic instance
data.
- Output a JSON manifest mapping each query to its expected result count and a sample
result row, suitable for use as a regression test suite.

ONTOLOGY CLASSES AND PROPERTIES (in Turtle):
<paste the relevant class and property declarations>

COMPETENCY QUESTIONS WITH SPARQL SKELETONS:
<paste the CQs with their SPARQL skeletons from Artefact 1>

Generate the instance data first, then the test queries. Stop after each section so I
can review.
```

Notes and variations

1. If you need internationalised test data, add: "Generate labels in <list of languages>. Every instance must have a label in every language."
2. If you are working in a regulated domain, add: "Every instance must have provenance: who created it, when, and the synthetic-data flag."
3. If the instance data ends up trivially answering every CQ, re-run with: "The current instance data is too uniform. Add adversarial cases: instances that look like they should answer the CQ but should not, and instances that look like they should not but do."

Artefact 6 — Style guide as system prompt

Purpose

A re-usable system prompt that captures your team's modelling conventions and applies them automatically every time the LLM proposes a change. The style guide is your guardrail; it is the difference between an LLM that produces idiomatic output for your team and one that produces generic suggestions.

When to use

Set this as the system prompt for every Claude conversation that touches your ontology. Update it whenever your team adopts a new convention or retires an old one.

The template

```
You are working on an ontology that follows the conventions below. Apply these
conventions to every suggestion, every fix, every generated artefact. Do not deviate
from these conventions even if the user asks; if the user asks for a deviation,
surface the conflict and ask for confirmation.

ALIGNMENT.
This ontology aligns to: <gist 14 / BFO 2020 / DOLCE / your upper ontology>.
Upper-level distinctions are NEVER to be redefined. Reuse upper-ontology classes by
IRI (rdfs:subClassOf gist:Agreement, NOT a redeclaration); do not re-declare them.
When proposing a new class, FIRST check whether a gist class already covers it
(gist:isDirectPartOf, gist:isCategorizedBy, gist:Specification, gist:Event,
gist:Task, gist:Agreement, gist:Magnitude).

NAMING.
Class names: PascalCase, singular, nouns. (Payment, PaymentInstruction,
ComplianceState)
Property names: camelCase, verbs or relationships. (heldBy, viaChannel,
hasRiskAssessment)
IRI structure: <baseIRI>/<module>/<localName>
Identifier patterns: <see team policy doc>

ANNOTATIONS.
Every class must have skos:prefLabel (English) and skos:definition.
rdfs:label and rdfs:comment are NOT used; SKOS is the team standard.
Definitions follow Aristotelian form: 'A <Genus> that <differentia>.'
Examples: 'An Agreement between a Bank and one or more Customers to hold and transact
funds.' for Account.

MODELLING PATTERNS.
Type discrimination: use gist:Category instances and gist:isCategorizedBy for type
variation (e.g. ChannelType: SWIFT, RTGS, CHIPS). NEVER subclass for type variation.
This is the gist anti-pattern that LLMs reach for first.
Plan vs occurrence: gist:Task for plans, gist:Event for occurrences. They are
distinct classes linked via an instructionFor (or similar) relation. Never collapse.
Composition vs subsumption: gist:isDirectPartOf for composition, rdfs:subClassOf only
for subsumption — never for has-a relationships.
Time-varying qualities: use a Specification subclass with gist:occursIn for the
temporal extent. Do not subclass per state.

PROFILE.
Target profile: OWL 2 <DL/EL/RL/QL — pick one>.
No constructs that violate the profile. Flag any suggestion that would.
```

```
FORBIDDEN PATTERNS.
No owl:equivalentClass without explicit human authorisation.
No transitive properties on data with cycles.
No annotation properties used as object properties.
No untyped literals.

OUTPUT FORMAT.
Always output Turtle for ontology fragments.
Always state the affected module and IRI explicitly.
Never output a fix without the corresponding SHACL update suggestion.
```

Notes and variations

1. Tailor the alignment, naming, annotations, and patterns sections to your team's actual conventions. The template is the scaffolding; the content is yours.
2. Keep the forbidden patterns list short and ruthless. Five entries, all enforced. Long lists are ignored by the LLM and by your team.
3. Version the style guide. Treat it as a controlled artefact. Every team change should go through review.
4. Tailor your style guide. It should reflect conventions from the upper ontology you selected.

Artefact 7 — Production readiness checklist

Purpose

A twelve-point checklist that an ontology should pass before being released for production use. The checklist is the threshold between "useful in research" and "safe to depend on". Apply rigorously.

When to use

Before every production release. As an audit tool when inheriting an ontology from a previous team. As the success criterion for a remediation project.

Check	Tool	Status
All competency questions have a passing SPARQL test against the canonical instance data.	*SPARQL runner*	☐
The reasoner runs without inconsistencies or unsatisfiable classes.	*HermiT / Pellet / ELK*	☐
Every NodeShape in the SHACL graph validates without violation against the canonical instance data.	*pyshacl / topbraid*	☐
The ontology declares its OWL 2 profile and has no profile violations.	*OWL profile checker / OntOLint*	☐
No undeclared classes, properties, or annotation properties.	*OntOLint*	☐
Every class has skos:prefLabel and skos:definition (or your team’s equivalent).	*OntOLint*	☐
The ontology has owl:versionIRI set to the release version.	*Manual / OntOLint*	☐
The ontology change-log records the deltas since the previous release.	*Diff tool*	☐
All decisions about alternative modelling options are documented (Y-statements or equivalent).	*Manual*	☐
Downstream artefacts are regenerated and tested: SHACL, JSON-LD context, OpenAPI spec, SQL DDL, documentation.	*Plugin / manual*	☐
The ontology has been adversarially critiqued by at least three perspectives and findings actioned.	*Artefact 2 prompt*	☐
A peer reviewer has signed off the release.	*Manual*	☐

> *Reading the checklist. Every item must be ticked or explicitly waived with a documented reason. A waived item is not a failed item, but every waiver should be reviewed at the next release. The checklist is also the basis for the lightweight conformance template later in this handout.*

Worked example: applying the methodology end-to-end

This example walks through one complete cycle of the methodology applied to the Cross-Border Payments and AML Ontology (CBPA) — the exemplar used in the workshop demonstration. The cycle takes a single high-value audit finding (the gist anti-pattern of plan/occurrence collapse) and works it through the Fix, Validate, and Conform beats.

The starting point

The workshop linter (GIST-002) and the reasoner together report that cbpa:Payment is asserted as both rdfs:subClassOf gist:Event AND rdfs:subClassOf gist:Task. The flawed ontology has collapsed the plan and the occurrence into a single class, with both planning-time properties (instructedAmount, plannedSettlementTime, authoriser) and occurrence-time properties (actualSettlementTime, actualPayerAccount) attached to the same class. CQ7 ("Which PaymentInstructions were authorised but never settled?") is unanswerable because there is no way to express an instruction without an occurrence — they are the same thing.

This is gist anti-pattern 2 (Event vs Task confusion) from the workshop demonstration. It is the most pedagogically valuable fix because it cannot be papered over with a property tweak; the collapsed class must be split.

Step 1: discuss the fix with the modeller

Open a fresh Claude conversation. Set Artefact 6 (the style guide) as the system prompt. Paste Artefact 4 (the modeller dialogue prompt) with the problem description filled in.

```
LLM (after clarifying questions about whether instructions persist across the
lifecycle and whether settlement legs are sub-events of payments or sub-plans of
instructions):

I see two fix options.

OPTION A — Split into PaymentInstruction (gist:Task) and Payment (gist:Event), linked
via an instructionFor object property.
  - PaymentInstruction carries the planning-time properties: instructsAmount,
requestedSettlementWindow, declaredPurpose, requestedChannel, authorisedBy,
instructionStatus.
  - Payment carries the occurrence-time properties: payer, payee, viaChannel,
settlementStatus, actual settlement leg events.
  - instructionFor: PaymentInstruction -> Payment (one instruction plans one payment;
the payment may not yet have occurred).
  - SettlementLeg remains a subClassOf Payment (an event subclass).
  Pros: gist-idiomatic; CQ7 becomes answerable; clean separation matches operational
reality and ISO 20022 messaging which distinguishes pacs.008 instruction from
pacs.002 status report.
  Cons: existing instance data must be migrated — every Payment instance must be
split into a paired PaymentInstruction and Payment.

OPTION B — Keep Payment single; add a paymentState property with values Drafted,
InFlight, Settled.
  - One class, distinguished by state.
  - State-based queries replace the plan/occurrence distinction.
  Pros: no instance data migration.
  Cons: violates the gist Task/Event distinction; conflates two things that gist
deliberately separates; the state pattern hides the plan/occurrence question rather
than answering it; CQ7 becomes a state filter rather than a structural query.
```

```
My recommendation: Option A. The Task/Event distinction is one of the central gist
patterns; the cost of preserving it is one-time data migration; the cost of violating
it is permanent representational debt. Option B is the LLM-classic shortcut: it
papers over the modelling problem instead of solving it.
```

You agree with the recommendation. The LLM then lists the downstream artefacts:

```
Downstream artefacts to update for Option A:
1. SHACL: split the existing PaymentShape into PaymentShape (Event-only properties)
and PaymentInstructionShape (Task-only properties). The two shapes must enforce the
disjointness — Payment must NOT carry instructsAmount; PaymentInstruction must NOT
carry actualSettlementTime.
2. CQs: CQ5 and CQ7 use the new structure. Update the SPARQL skeletons in the CQ
register. CQ7 becomes a NOT EXISTS pattern over the instructionFor relation.
3. Instance data migration: every existing Payment instance becomes a paired
PaymentInstruction + Payment. Generate the migration SPARQL UPDATE.
4. JSON-LD context: add PaymentInstruction as a top-level type; add instructionFor as
a reference property.
5. Documentation: add a section explaining the plan/occurrence distinction; this is
the kind of architectural decision worth documenting verbosely.
6. Change log: record the migration with a Y-statement.
```

Step 2: apply the fix

Edit CBPA in Protégé: 1. Remove rdfs:subClassOf gist:Task from :Payment. 2. Create the new class :PaymentInstruction rdfs:subClassOf gist:Task. 3. Move planning-time properties (instructedAmount renamed to instructsAmount, plus requestedSettlementWindow, declaredPurpose, requestedChannel, authorisedBy, instructionStatus) so they bear on :PaymentInstruction. 4. Create the object property :instructionFor with domain :PaymentInstruction and range :Payment. 5. Save.

Step 3: regenerate SHACL

Open Artefact 3 (the SHACL generation prompt) in a fresh conversation. Provide the corrected class definitions for both Payment and PaymentInstruction. Generate the two NodeShapes. Critically, include the disjointness constraints (sh:maxCount 0 on the wrong-class properties).

Step 4: migrate the instance data

Generate a SPARQL UPDATE that splits each existing Payment into a paired PaymentInstruction + Payment. Run against your test data first; verify the row counts (one Payment becomes one PaymentInstruction + one Payment). Then run against production data only after the SHACL passes on the migrated test data.

Step 5: re-run the validation suite

1. Reasoner. Run HermiT in Protégé. Confirm Payment and PaymentInstruction are now consistent and disjoint.
2. SHACL. Run pyshacl over the corrected ontology against the migrated instance data. Confirm zero violations on the new PaymentShape and PaymentInstructionShape — particularly that no Payment carries plan-time properties and no PaymentInstruction carries occurrence-time properties.
3. CQs. Run the SPARQL test suite. Confirm CQ7 now returns the expected instructions-without-occurrences. Confirm CQ5 still works against the new structure.
4. Re-run the workshop linter. Confirm the dual-subclass anti-pattern is gone.

Step 6: update the conformance score

Open the lightweight conformance template (next page). Increment "CQs passing" by one (CQ7 now answers). Decrement "SHACL violations" by the count of violations that have cleared. Decrement "workshop linter findings remaining" accordingly. The structural quality of the ontology has measurably improved; the score reflects that.

Step 7: document the decision

Add a design decision entry to the change log:

```
2026-07-02 Split Payment into PaymentInstruction (gist:Task) and Payment
(gist:Event).
In the context of cross-border payment monitoring under AML/FIU oversight,
facing the need to distinguish instructions-without-occurrences and occurrences-
against-instructions,
we decided to split the previously-collapsed Payment class,
to achieve the gist-idiomatic Task/Event separation,
accepting one-time instance data migration cost.
Approved by <reviewer>. Modeller dialogue transcript saved to
decisions/2026-07-02-payment-instruction-split.md.
```

The fix is complete. Total elapsed time: roughly twenty minutes including the LLM dialogue and the data migration. The next fix follows the same pattern. This is the methodology in production use.

Lightweight conformance template

Use this template at the end of any modelling session to score the ontology. The same template is used during the workshop's Beat D4. Take a baseline at the start of a remediation project and re-score after each fix to see the trend.

Metric	Source	Today	Target
Number of competency questions defined.	*Artefact 1 output*		**10–12**
Number of competency questions passing as SPARQL tests.	*SPARQL runner over Artefact 5 data*		**All defined**
Number of SHACL violations.	*pyshacl over instance data*		**0**
Number of workshop linter findings remaining (Critical + Important).	*Workshop linter*		**0 critical**
Number of OWL profile violations.	*Profile checker / OntOLint*		**0**
Number of unsatisfiable classes.	*Reasoner*		**0**
Reasoner consistency.	*Reasoner*	Y / N	**Y**
Number of classes lacking skos:prefLabel or skos:definition.	*OntOLint / manual*		**0**
Number of properties lacking rdfs:domain or rdfs:range.	*Manual / OntOLint*		**0**
owl:versionIRI declared.	*Manual*	Y / N	**Y**

Three next steps for your ontology

Identify the three highest-priority items that came out of today's session. Write them down before you leave the room.

1. _____

2. _____

3. _____

> *The full conformance report. The lightweight template covers the essentials. The full workshop Linter conformance report adds: full lint findings by category, deltas against the previous run, recommended fix order, severity-weighted score, profile-specific recommendations, and a SHACL coverage report.*

Where to go next

Use this handout

Apply the methodology to your own ontology in your next modelling session. Even one step such as the adversarial critique on a single class definition, or the SHACL regeneration on one property, is enough to see the difference between LLM-as-generator and LLM-as-controlled-assistant.

The workshop linter, prompts, and CBPA sample ontology are at https://graphresearchlabs.com/workshop-with-yitae-jeong-2026-links/

Stay in touch

- Email: dougal.watt@graphresearchlabs.com
- Web: https://www.graphresearchlabs.com
- Connect on LinkedIn: Dougal Watt — Graph Research Labs