Description Logic

(From Natural Language to Ontology Reasoning)

Policy • Claim • Rules • Proofs • LLM & GraphRAG

Complete Developer Study Notes

Concepts • Practical Examples • Projects • Enterprise Architecture

$C \sqsubseteq D$ $C \sqcap \exists R.D$ $\forall R.C$ $\geq n\ R.C$ $\top$ $\bot$

Prepared by: Rajesh Kumar Gupta

20 Pages

Rajesh Kumar Gupta

1 Introduction to Description Logic

1.1 What is Description Logic (DL) ?

- A family of **formal knowledge representation languages** - decidable fragments of First-Order Logic (FOL).
- Describes a domain using **concepts** (classes), **roles** (relations) and **individuals** (instances).
- It is the **logical foundation of OWL**: OWL 2 DL = DL SROIQ(D).
- Gives every ontology statement a precise meaning, so a **reasoner** can prove new facts.

1.2 Building Blocks

DL Term	OWL Term	Example
Concept (C)	Class	Policy, Claim
Role (R)	Object Property	isAgainst, hasCoverage
Concrete role	Data Property	claimedAmount
Individual (a)	Named Individual	POL-77, CLM-101
Axiom	Axiom	Claim ⊑ Event

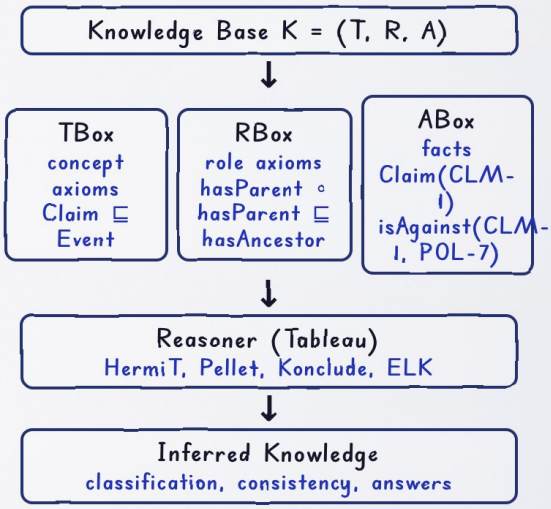
1.3 Short History

1. **1970s** : Semantic networks & frames - no formal semantics.
2. **1985** : KL-ONE - first "terminological" system.
3. **1991** : ALC - the basic DL (Schmidt-Schauß & Smolka).
4. **2004** : OWL 1 DL = SHOIN(D).
5. **2009 / 2012** : OWL 2 DL = SROIQ(D) (W3C).

1.4 DL vs First-Order Logic

	FOL	DL
Variables	Explicit (∀x)	Hidden (variable-free)
Decidable ?	No	Yes (by design)
Readability	Hard	Close to language
Example	∀x Claim(x) → ∃y isAgainst(x,y) ∧ Policy(y)	Claim ⊑ ∃isAgainst.Policy

1.5 A DL Knowledge Base



1.6 Why DL Matters Today ?

1. **Precise meaning** : no ambiguity between teams and systems.
2. **Automatic inference** : hidden facts become explicit.
3. **Consistency checking** : catch contradictory rules / data early.
4. **Explainability** : every inference has a proof (justification).
5. **AI grounding** : gives LLMs a verifiable "truth layer".
6. **Reuse** : one definition used from enterprise model down to APIs.

1.7 Where DL is Used

- **Healthcare**: SNOMED CT (EL family, 350k+ concepts).
- **Finance**: FIBO (OWL 2 DL).
- **Insurance**: coverage & claim eligibility classification.
- **Enterprise KGs, semantic web, AI agents.**

1.8 Reading DL Aloud

Formula	Say
Claim ⊑ Event	"every claim is an event"
∃isAgainst.Policy	"against some policy"
∀covers.Home	"covers only homes"

Rajesh Kumar Gupta

② DL Syntax & Semantics

2.1 Concept Constructors (ALC +)

Syntax	Meaning	Example
$\top$	Top - everything	$\top$
$\bot$	Bottom - nothing	$\bot$
$\neg C$	Not C	$\neg$ActivePolicy
$C \sqcap D$	C and D	Policy $\sqcap$ Commercial
$C \sqcup D$	C or D	Person $\sqcup$ Organization
$\exists R.C$	some R-link to a C	$\exists$hasCoverage.Flood
$\forall R.C$	all R-links go to C	$\forall$covers.Residential
$\geq n\ R.C$	at least n	≥ 2 hasCoverage.Coverage
$\leq n\ R.C$	at most n	≤ 1 hasNamedInsured.$\top$
$\{a\}$	nominal (one individual)	{Flood}
R^-	inverse role	isAgainst$^-$ = hasClaim
$\exists R.Self$	self link	$\exists$insures.Self

2.2 Formal Semantics

- An **interpretation** $I = (\Delta^I, \cdot^I)$: a domain Δ of objects + a mapping.
- Concept $C^I \subseteq \Delta$, role $R^I \subseteq \Delta \times \Delta$, individual $a^I \in \Delta$.

$$(C \sqcap D)^I = C^I \cap D^I$$
$$(\neg C)^I = \Delta \setminus C^I$$
$$(\exists R.C)^I = \{\, x \mid \exists y: (x,y) \in R^I \text{ and } y \in C^I \,\}$$
$$(\forall R.C)^I = \{\, x \mid \forall y: (x,y) \in R^I \rightarrow y \in C^I \,\}$$

2.3 Tiny Interpretation

$\Delta = \{ p1, p2, c1 \}$
$Policy^I = \{ p1, p2 \}$
$Claim^I = \{ c1 \}$
$isAgainst^I = \{ (c1, p1) \}$
$(\exists isAgainst.Policy)^I = \{ c1 \}$
$(\exists isAgainst^-.Claim)^I = \{ p1 \}$ # policies with claims

2.4 DL Naming - The Alphabet Soup

Letter	Adds
AL	Attributive language: $\sqcap$, $\forall R.C$, $\exists R.\top$, atomic $\neg$
C	Full negation $\neg C$ (ALC)
S	ALC + transitive roles
H	Role hierarchy ($R \sqsubseteq S$)
R	Role chains, reflexive, disjoint roles
O	Nominals {a}
I	Inverse roles R^-
N	Number restrictions $\geq n\ R$
Q	Qualified numbers $\geq n\ R.C$
F	Functional roles
(D)	Datatypes (strings, numbers, dates)
E / EL	Existential + conjunction only (fast)

2.5 DL ↔ OWL Profiles

DL	Language	Reasoning
SHOIN(D)	OWL 1 DL	NExpTime
SROIQ(D)	OWL 2 DL	N2ExpTime
EL++	OWL 2 EL	Polynomial
DL-Lite	OWL 2 QL	LogSpace (data)
DLP (Horn)	OWL 2 RL	Polynomial (rules)

2.6 Manchester Syntax (Readable OWL)

DL	Manchester
$C \sqcap D$	C and D
$C \sqcup D$	C or D
$\neg C$	not C
$\exists R.C$	R some C
$\forall R.C$	R only C
$\geq 2\ R.C$	R min 2 C
$=1\ R.C$	R exactly 1 C
$\{a\}$	{a} (value: R value a)

Rajesh Kumar Gupta

③ DL Knowledge Base: TBox, RBox, ABox

3.1 TBox - Terminology (Schema)

Axiom	Name	Reads as
$C \sqsubseteq D$	Subsumption (GCI)	every C is a D
$C \equiv D$	Equivalence (definition)	C is exactly D
$C \sqcap D \sqsubseteq \bot$	Disjointness	nothing is both
$\exists R.\top \sqsubseteq C$	Domain	R starts at a C
$\top \sqsubseteq \forall R.C$	Range	R ends at a C

- **Primitive** concept: only necessary conditions ($\sqsubseteq$).
- **Defined** concept: necessary + sufficient ($\equiv$) - the reasoner can classify individuals into it.

3.2 RBox - Role Axioms

Axiom	Example
$R \sqsubseteq S$	hasNamedInsured $\sqsubseteq$ hasParty
$R \equiv S^-$	hasClaim $\equiv$ isAgainst$^-$
Trans(R)	Trans(isPartOf)
Func(R)	Func(isAgainst)
$R \circ S \sqsubseteq T$	isAgainst $\circ$ hasCoverage $\sqsubseteq$ claimHasCoverage
Disj(R, S)	Disj(isInsuredBy, isBrokerOf)

3.3 ABox - Assertions (Data)

Assertion	Example
C(a)	HomeownerPolicy(POL-77)
R(a, b)	isAgainst(CLM-101, POL-77)
$\neg$C(a)	$\neg$ActivePolicy(POL-12)
$a \neq b$	POL-77 $\neq$ POL-78
U(a, v)	claimedAmount(CLM-101, 15000)

3.4 Mini Insurance KB (Example)

TBox:

$\text{HomeownerPolicy} \sqsubseteq \text{Policy}$

$\text{Policy} \sqsubseteq \exists \text{hasPolicyholder.Party}$

$\text{Policy} \sqcap \text{Claim} \sqsubseteq \bot$

$\text{Claim} \sqsubseteq =1\ \text{isAgainst.Policy}$

$\text{ActivePolicy} \equiv \text{Policy} \sqcap \exists \text{hasStatus.}\{\text{Active}\}$

ABox:

HomeownerPolicy(POL-77)

hasStatus(POL-77, Active)

isAgainst(CLM-101, POL-77)

3.5 What the Reasoner Infers

① Policy(POL-77) : from HomeownerPolicy $\sqsubseteq$ Policy.

② ActivePolicy(POL-77) : Policy + status Active matches the definition.

③ $\exists$hasPolicyholder.Party(POL-77) : a policyholder exists - even if unknown (open world).

④ $\neg$Claim(POL-77) : from disjointness.

3.6 Open World & No Unique Names

- OWA: unknown $\neq$ false. Missing policyholder is not an error in DL.
- No UNA: two names may denote the same object unless stated $a \neq b$.
- Use SHACL / rules for closed-world checks ("policyholder must be present").

Rajesh Kumar Gupta

④ Natural Language → Description Logic

4.1 NL → DL Method (8 Steps)

1. Normalize Sentence
"Each / every / all" → universal
↓
2. Find Concepts
nouns → classes (Policy, Claim)
↓
3. Find Roles
verbs / "has" → properties
↓
4. Find Quantifiers
some / only / at least / exactly
↓
5. Find Logic Words
and / or / not / unless / if
↓
6. Pick Axiom Type
$\sqsubseteq$ (rule) vs $\equiv$ (definition) vs ABox fact
↓
7. Formalize
DL → Manchester → OWL
↓
8. Verify
reasoner: satisfiable ? expected inferences ?

4.2 Linguistic Cues

Words	DL
is a, is a kind of	$\sqsubseteq$
is defined as, means, iff	$\equiv$
has (some), at least one	$\exists$
only, exclusively	$\forall$
no, never, cannot be both	$\sqcap \ldots \sqsubseteq \bot$
not	$\neg$
and / or	$\sqcap$ / $\sqcup$

4.3 Quick Example

"Every auto policy covers only vehicles."
$\Longrightarrow$ AutoPolicy $\sqsubseteq \forall$covers.Vehicle

4.4 Translation Patterns

Natural Language	Description Logic
Every A is a B	$A \sqsubseteq B$
A is exactly a B that is C	$A \equiv B \sqcap C$
Every A has some B	$A \sqsubseteq \exists$has.B
A has only Bs	$A \sqsubseteq \forall$has.B
A has at least n Bs	$A \sqsubseteq \geq n$ has.B
A has exactly one B	$A \sqsubseteq =1$ has.B
No A is a B	$A \sqcap B \sqsubseteq \bot$
A is B or C	$A \sqsubseteq B \sqcup C$
If x has R to a C, x is A	$\exists R.C \sqsubseteq A$
x is an A	$A(x)$
x relates to y by R	$R(x, y)$
R is the reverse of S	$R \equiv S^-$
R then S implies T	$R \circ S \sqsubseteq T$

4.5 Common Traps

① **"only" ≠ "some"** : $\forall R.C$ is TRUE for things with NO R-links. Usually write both: $\exists R.C \sqcap \forall R.C$.

② **"and" in English** : "policies for cars and boats" = Car $\sqcup$ Boat (or!), not $\sqcap$.

③ **Rule vs definition** : "Active policies are policies with status active" is a definition ($\equiv$).

④ **Hidden variables** : "the same policy" needs a role chain or a rule (DL is tree-shaped).

⑤ **Negation** : "not covered" in OWA needs explicit facts or closed-world rules.

- Always **read the DL back** into English and confirm with the business owner.

Rajesh Kumar Gupta

⑤ Policy Examples: NL → DL

5.1 | Policy: Simple → Complex

L1 "Every homeowner policy is an insurance policy."
⟹ HomeownerPolicy ⊑ Policy

L2 "Every policy has at least one policyholder who is a party."
⟹ Policy ⊑ ≥1 hasPolicyholder.Party

L3 "A policy is either personal or commercial, never both."
⟹ Policy ⊑ PersonalPolicy ⊔ CommercialPolicy
PersonalPolicy ⊓ CommercialPolicy ⊑ ⊥

L4 "An active policy is exactly a policy whose status is Active."
⟹ ActivePolicy ≡ Policy ⊓ ∃hasStatus. {Active}

L5 "A homeowner policy insures only residential properties, and insures at least one."
⟹ HomeownerPolicy ⊑ ∃insures.ResidentialProperty ⊓ ∀insures.ResidentialProperty

L6 "A flood-covered policy is a policy that has some coverage which covers the flood peril."
⟹ FloodCoveredPolicy ≡ Policy ⊓ ∃hasCoverage.(Coverage ⊓ ∃coversPeril. {Flood})

L7 "A commercial package policy has at least two coverages and exactly one named insured, who must be an organization."
⟹ CommercialPackagePolicy ⊑ CommercialPolicy ⊓ ≥2 hasCoverage.Coverage ⊓ =1 hasNamedInsured.⊤ ⊓ ∀hasNamedInsured.Organization

L8 "Every policy has exactly one effective date and exactly one expiry date."
⟹ Policy ⊑ =1 effectiveDate.date ⊓ =1 expiryDate.date

L9 "A policy that carries a flood exclusion does not cover flood."
⟹ Policy ⊓ ∃hasExclusion. {FloodExclusion} ⊑ ¬FloodCoveredPolicy

L10 "A high-risk policy is an active policy insuring a property located in a flood zone, or one with more than 3 prior claims."
⟹ HighRiskPolicy ≡ ActivePolicy ⊓ (∃insures.(∃locatedIn.FloodZone) ⊔ ≥4 hasClaim.Claim)

5.2 | Same L6 in Manchester + Turtle

```
Class: FloodCoveredPolicy
  EquivalentTo:
    Policy and (hasCoverage some
      (Coverage and (coversPeril value
Flood)))

:FloodCoveredPolicy owl:equivalentClass [
  owl:intersectionOf ( :Policy
    [ a owl:Restriction ;
      owl:onProperty :hasCoverage ;
      owl:someValuesFrom [ owl:intersectionOf
        ( :Coverage [ a owl:Restriction ;
            owl:onProperty :coversPeril ;
            owl:hasValue :Flood ] ) ] ] ) ] .
```

5.3 | What We Gain

- Any policy with a flood coverage is **auto-classified** as FloodCoveredPolicy.
- L9 + L6 together → reasoner flags a policy with BOTH flood coverage and flood exclusion as **inconsistent**.

Rajesh Kumar Gupta

6 Claim Examples: NL → DL

6.1 Claim: Simple → Complex

L1 "Every claim is an event."
⟹ Claim ⊑ Event

L2 "Every claim is made against exactly one policy."
⟹ Claim ⊑ =1 isAgainst.Policy

L3 "Every claim arises from some loss event, and a loss event is caused by some peril."
⟹ Claim ⊑ ∃arisesFrom.LossEvent
LossEvent ⊑ ∃causedBy.Peril

L4 "A flood claim is a claim arising from a loss caused by flood."
⟹ FloodClaim ≡ Claim ⊓ ∃arisesFrom.(∃causedBy.{Flood})

L5 "A high-value claim is a claim whose claimed amount is over 100,000."
⟹ HighValueClaim ≡ Claim ⊓ ∃claimedAmount.decimal[> 100000]

L6 "A claim made against an inactive policy is a non-covered claim."
⟹ Claim ⊓ ∃isAgainst.¬ActivePolicy ⊑ NonCoveredClaim

L7 "A frequent claimant is a party who is claimant on at least 4 claims."
⟹ FrequentClaimant ≡ Party ⊓ ≥4 isClaimantOf.Claim

L8 "A referral claim is a high-value claim, or any claim whose claimant is a frequent claimant."
⟹ ReferralClaim ≡ HighValueClaim ⊔ (Claim ⊓ ∃hasClaimant.FrequentClaimant)

L9 "A subrogation claim is a claim with some recovery from a third party."
⟹ SubrogationClaim ≡ Claim ⊓ ∃hasRecovery.(∃recoveredFrom.ThirdParty)

L10 "A closed claim has no open reserve."
⟹ ClosedClaim ⊑ ∀hasReserve.¬OpenReserve

L11 "A potentially covered flood claim is a flood claim against an active, flood-covered policy."
⟹ PotentiallyCoveredFloodClaim ≡ FloodClaim ⊓ ∃isAgainst.(ActivePolicy ⊓ FloodCoveredPolicy)

6.2 Where Pure DL Stops

"A claim is covered if the peril of ITS loss is a peril covered by THE SAME policy the claim is against."

This needs two paths to meet at the **same** peril (a triangle). DL concepts are **tree-shaped** - they cannot say "the same y".

6.3 Workarounds

1. **Role chain** : isAgainst ∘ hasCoverage ∘ coversPeril ⊑ claimPolicyPeril
   arisesFrom ∘ causedBy ⊑ claimLossPeril
2. **SWRL / Datalog rule** : bind variables explicitly (page 12).
3. **SPARQL / SHACL rule** : closed-world check at query time.

6.4 Key Points

- Concepts = reusable "views" of claims (HighValue, Referral).
- Use rules where variables must be shared.

6.5 Test Individuals

```
Claim(T1)  isAgainst(T1, POL77)
arisesFrom(T1, L9)  causedBy(L9, Flood)
claimedAmount(T1, 250000)
# expected after reasoning:
# T1 : FloodClaim, HighValueClaim,
#      ReferralClaim, PotentiallyCovered...
```

Rajesh Kumar Gupta

7 Complex Sentence → DL → OWL

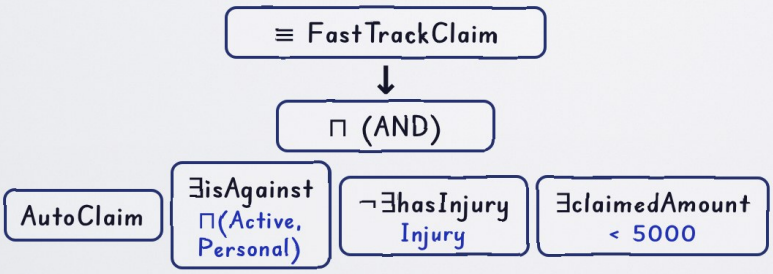
7.1 Worked Example - One Sentence

"A fast-track claim is an auto claim, against an active personal policy, with no injury and a claimed amount below 5,000."

7.2 Step-by-Step Parse

Phrase	Element	DL
fast-track claim	defined concept	FastTrackClaim ≡ …
an auto claim	superclass	AutoClaim
against	role	∃isAgainst.
an active personal policy	nested concept	(ActivePolicy ⊓ PersonalPolicy)
with no injury	negated existential	¬∃hasInjury.Injury
below 5,000	datatype facet	∃claimedAmount.decimal[< 5000]

7.3 Parse Tree



7.4 Final DL Axiom

FastTrackClaim ≡ AutoClaim
⊓ ∃isAgainst.(ActivePolicy ⊓ PersonalPolicy)
⊓ ¬∃hasInjury.Injury
⊓ ∃claimedAmount.decimal[< 5000]

7.5 When a Sentence Needs a Rule

"A total-loss claim is an auto claim whose repair estimate exceeds the vehicle value."
Compares TWO values of the same claim → not expressible in OWL DL.
Rule: AutoClaim(?c) ∧ repairEstimate(?c, ?r) ∧ vehicleValue(?c, ?v) ∧ swrlb:greaterThan(?r, ?v) → TotalLossClaim(?c)

7.6 Manchester Syntax

```
Class: FastTrackClaim
  EquivalentTo:
    AutoClaim
    and (isAgainst some
          (ActivePolicy and
PersonalPolicy))
    and (not (hasInjury some Injury))
    and (claimedAmount some
          xsd:decimal[< 5000])
```

7.7 SHACL (Closed-World Twin)

```
:FastTrackShape a sh:NodeShape ;
  sh:targetClass :AutoClaim ;
  sh:property [ sh:path :hasInjury ;
                sh:maxCount 0 ] ;
  sh:property [ sh:path :claimedAmount
;
                sh:maxExclusive 5000
] .
```

7.8 OWA Warning

- ¬∃hasInjury.Injury is only TRUE if the KB **proves** no injury exists, e.g. via a closing axiom or "=0 hasInjury" fact.
- In practice: DL **defines** the concept; SHACL / rules **check** missing data with closed-world semantics.

7.9 Checklist

1. Every noun mapped to an ontology class ?
2. Every verb mapped to a declared role ?
3. Quantifier (some / only / n) chosen deliberately ?
4. Reasoner says concept is satisfiable ?
5. Test individuals classify as expected ?

Rajesh Kumar Gupta

⑧ Reasoning Tasks in DL

8.1 Standard Reasoning Tasks

Task	Question
Satisfiability	Can concept C have any instance ?
Subsumption	Is every C also a D ? ($K \models C \sqsubseteq D$)
Equivalence	Do C and D mean the same ?
Consistency	Is the whole KB free of contradiction ?
Instance check	Is a an instance of C ? ($K \models C(a)$)
Classification	Compute the full class hierarchy
Realization	Most specific classes of each individual
Query answering	All a such that C(a) / conjunctive queries

8.2 All Reduce to Satisfiability

$C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable
$C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$
$K \models C(a)$ iff $K \cup \{ \neg C(a) \}$ is inconsistent
C unsat iff $K \cup \{ C(x) \}$ is inconsistent

- So one engine (a **tableau** or hypertableau) answers everything.

8.3 Complexity (Worst Case)

Logic	Concept satisfiability
EL / EL++	PTime
ALC	PSpace (ExpTime with TBox)
SHIQ	ExpTime
SHOIN	NExpTime
SROIQ	N2ExpTime

8.4 Instance Checking Example

- Query: $K \models FloodClaim(CLM\text{-}101)$?
- Facts: arisesFrom(CLM-101, L1), causedBy(L1, Flood).
- Definition L4 → YES, even though nobody typed "FloodClaim".

8.5 Classification Example

Asserted:

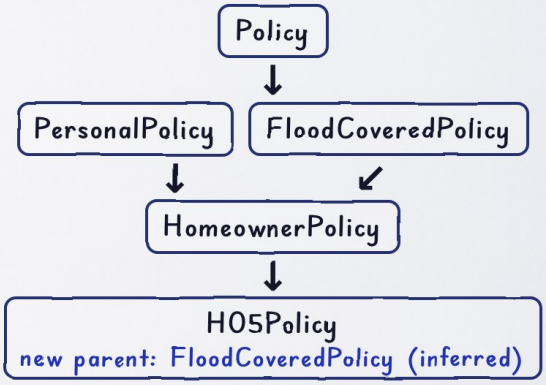
$HomeownerPolicy \sqsubseteq PersonalPolicy$

$PersonalPolicy \sqsubseteq Policy$

$FloodCoveredPolicy \equiv Policy \sqcap \exists hasCoverage.FloodCoverage$

$HO5Policy \sqsubseteq HomeownerPolicy \sqcap \exists hasCoverage.FloodCoverage$

Inferred hierarchy:



8.6 Consistency Example

$Policy \sqcap Claim \sqsubseteq \bot$

Claim(X-9), Policy(X-9)

- → KB is **inconsistent**: X-9 cannot be both.
- A data-loading bug (wrong type mapping) is caught by logic.

8.7 Unsatisfiable Class

$BadPolicy \sqsubseteq \geq 2\ hasNamedInsured.\top \sqcap \leq 1\ hasNamedInsured.\top$

- → $BadPolicy \sqsubseteq \bot$ - a modeling error shown in red in Protégé.

8.8 Query Answering

- "All referral claims" = all a with $K \models ReferralClaim(a)$.
- Answers include **inferred** members, not only asserted ones.

Rajesh Kumar Gupta

9 Theorem Proving in Ontologies

9.1 Proof in the Ontology World

- Like a theorem prover: **axioms** = premises, **query** = theorem.
- Most DL reasoners use **proof by refutation**: assume the opposite, derive a **clash** (contradiction).
- Tableau = build a model step by step; if every branch clashes, the theorem holds.

9.2 Tableau Expansion Rules (ALC)

Rule	If x has ...	Then add
$\sqcap$	$(C \sqcap D)(x)$	$C(x)$ and $D(x)$
$\sqcup$	$(C \sqcup D)(x)$	branch: $C(x) \mid D(x)$
$\exists$	$(\exists R.C)(x)$	new y: $R(x,y)$, $C(y)$
$\forall$	$(\forall R.C)(x)$, $R(x,y)$	$C(y)$
$\sqsubseteq$	TBox $C \sqsubseteq D$	$(\neg C \sqcup D)(x)$ for every x
$\geq n$	$(\geq n\ R.C)(x)$	n new distinct y_i

- **Clash**: $A(x)$ and $\neg A(x)$, or $\bot(x)$.
- Concepts first put in **Negation Normal Form** (NNF): push $\neg$ inward.
- **Blocking** stops infinite loops with cyclic TBoxes.

9.3 NNF Rules

$\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$
$\neg(C \sqcup D) \rightarrow \neg C \sqcap \neg D$
$\neg\exists R.C \rightarrow \forall R.\neg C$
$\neg\forall R.C \rightarrow \exists R.\neg C$

9.4 Properties of a Good Prover

1. **Sound** : never proves something false.
2. **Complete** : finds every true entailment.
3. **Terminating** : always stops (decidable logic).

9.5 Worked Proof

Abbrev: FC = FloodClaim, FCP = FloodCoveredPolicy, iA = isAgainst
Prove: $FC \sqcap \exists iA.FCP \sqsubseteq \exists iA.Policy$

Negate goal - assume some x with:
$FC \sqcap \exists iA.FCP \sqcap \neg\exists iA.Policy$
NNF: $\neg\exists iA.Policy \rightarrow \forall iA.\neg Policy$

1. $FC(x)$ [$\sqcap$-rule]
2. $(\exists iA.FCP)(x)$ [$\sqcap$-rule]
3. $(\forall iA.\neg Policy)(x)$ [$\sqcap$-rule]
4. $iA(x,y)$, $FCP(y)$ [$\exists$ on 2]
5. $\neg Policy(y)$ [$\forall$ on 3,4]
6. $FCP \equiv Policy \sqcap \ldots$ $\Rightarrow Policy(y)$ [TBox]
7. $Policy(y)$, $\neg Policy(y)$ $\Rightarrow$ CLASH ✗

Every branch clashes
$\Rightarrow$ subsumption PROVED ✓

9.6 Justifications (Explanations)

- A **justification** = minimal set of axioms that entails the result.
- Protégé "Explain inference" / OWL API ExplanationGenerator.
- This is the audit trail for regulators and the material an LLM can turn into plain English.

9.7 Other Proof Methods

Method	Used in
Hypertableau	HermiT
Tableau + optimizations	Pellet, FaCT++, Konclude
Consequence-based	ELK (EL), Sequoia
Resolution / Datalog	KAON2, RDFox (rules)

9.8 Proof → Explanation

Justification for FloodCoveredPolicy(POL77):
{ AI definition, hasCoverage(POL77, FL01), coversPeril(FL01, Flood) }
→ LLM: "Your policy is flood-covered because endorsement FL01 covers flood."

Rajesh Kumar Gupta

10 Rules in the Ontology World

10.1 What is a Rule ?

- A rule is an IF (body / antecedent) → THEN (head / consequent) statement.
- In logic: a Horn clause - a conjunction of atoms implies one atom.
- Rules add what DL cannot say alone: shared variables, arithmetic, dates, defaults, actions.

Body: Claim(?c) ∧ isAgainst(?c, ?p) ∧ Lapsed(?p)

Head: → NonCovered(?c)

10.2 Kinds of Rules in Ontology World

1. **DL axioms** : implicit rules: $C \sqsubseteq D$ means "if C then D" (monotonic, OWA).
2. **RDFS / OWL-RL entailment** : fixed W3C rule sets run by triple stores.
3. **SWRL** : Horn rules with variables over OWL classes & properties.
4. **Datalog / RDFox** : recursive rules, very fast, incremental.
5. **SHACL-AF rules** : sh:TripleRule / sh:SPARQLRule - derive triples.
6. **SPARQL CONSTRUCT / SPIN** : query-based derivation (closed world).
7. **N3 rules** : Notation3 logic (EYE reasoner).
8. **Business / production rules** : Drools, DMN decision tables - actions + priorities.
9. **Integrity constraints** : SHACL shapes - "must" checks, not new facts.

10.3 Rule Engines

Engine	Rules
HermiT / Pellet	OWL + DL-safe SWRL
RDFox	Datalog, incremental
GraphDB	OWL-RL + custom rulesets
TopBraid / pySHACL	SHACL-AF rules
Drools / Camunda	production rules, DMN

10.4 Rules by Purpose

Type	Purpose	Example
Derivation	create new facts	late-reported claim
Classification	assign a class	fast-track claim
Integrity	reject bad data	claim needs a policy
Computation	calculate values	payable = loss - deductible
Reaction (ECA)	event → action	FNOL → assign adjuster
Default / exception	"usually, unless"	covered unless excluded

10.5 Monotonic vs Non-Monotonic

	DL / SWRL	Business rules / SPARQL
New fact can retract old ?	No (monotonic)	Yes
Negation	classical ¬	negation-as-failure (NOT EXISTS)
World	Open	Closed
Good for	definitions, consistency	decisions, exceptions, defaults

10.6 DL-Safe Rules

- Unrestricted SWRL + OWL is **undecidable**.
- **DL-safe**: rule variables bind only to **named individuals** → decidable (HermiT, Pellet support this).

10.7 Choosing

1. **Meaning of a term** : DL axiom.
2. **Same-variable logic** : SWRL / Datalog.
3. **Missing / wrong data** : SHACL.
4. **Decision with priorities / amounts** : DMN or production rules, fed by the ontology.

Rajesh Kumar Gupta

⑪ Rule Examples for Policy & Claim

11.1 SWRL: Covered Peril (Same Variable)

```
Claim(?c) ^ isAgainst(?c, ?p) ^
ActivePolicy(?p) ^ hasCoverage(?p, ?cov) ^
coversPeril(?cov, ?peril) ^
arisesFrom(?c, ?loss) ^ causedBy(?loss, ?peril)
  -> PerilCoveredClaim(?c) ^
     respondingCoverage(?c, ?cov)
```

- The shared **?peril** and **?p** are exactly what DL could not express.

11.2 SWRL: Loss Inside Policy Term

```
Claim(?c) ^ isAgainst(?c, ?p) ^
arisesFrom(?c, ?l) ^ lossDate(?l, ?d) ^
effectiveDate(?p, ?s) ^ expiryDate(?p, ?e) ^
swrlb:greaterThanOrEqual(?d, ?s) ^
swrlb:lessThanOrEqual(?d, ?e)
  -> LossInPolicyPeriod(?c)
```

11.3 Datalog (RDFox) - Recursive

```
[?x, :controls, ?z] :-
   [?x, :controls, ?y], [?y, :controls, ?z] .

[?c, a, :ReferralClaim] :-
   [?c, :hasClaimant, ?pty],
   [?pty, :isClaimantOf, ?other],
   [?other, a, :SIUFlaggedClaim] .
```

11.4 SPARQL CONSTRUCT Rule

```
CONSTRUCT { ?c :hasDuplicateOf ?d }
WHERE {
  ?c a :Claim ; :isAgainst ?p ;
     :arisesFrom/:lossDate ?dt .
  ?d a :Claim ; :isAgainst ?p ;
     :arisesFrom/:lossDate ?dt .
  FILTER (?c != ?d) }
```

11.5 N3 Rule

```
{ ?c :isAgainst ?p . ?p :hasStatus :Lapsed }
  => { ?c a :NonCoveredClaim } .
```

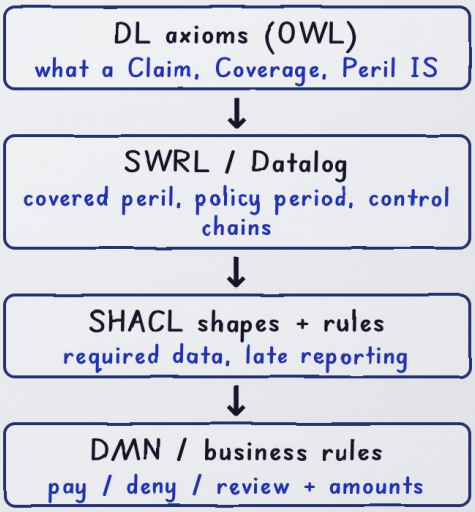
11.6 SHACL-AF: Late Reporting

```
:LateRule a sh:NodeShape ;
 sh:targetClass :Claim ;
 sh:rule [ a sh:SPARQLRule ; sh:construct """
  CONSTRUCT { $this a :LateReportedClaim
}
  WHERE { $this :fnolDate ?f ;
          :arisesFrom/:lossDate ?l .
    FILTER(?f - ?l >
"P30D"^^xsd:duration) }"""] .
```

11.7 DMN Decision Table - Payable

Covered ?	Excluded ?	Late ?	Decision
true	false	false	PAY: min(loss - deductible, limit)
true	false	true	REVIEW
true	true	-	DENY (exclusion)
false	-	-	DENY (no coverage)

11.8 Layered Rule Stack



- Each layer **reuses the same ontology terms** - no re-definition.

Rajesh Kumar Gupta

⑫ DL in the Enterprise: Model → API

12.1 Enterprise → API Concept Reuse

Enterprise Core Ontology (DL)
Party, Policy, Coverage, Claim, Peril, Money - definitions & axioms

↓

Domain Modules
Claims, Underwriting, Billing - specialize core (⊑)

↓

Defined Concepts (≡)
HighValueClaim, FastTrackClaim, ActivePolicy

↓

Application Profiles
SHACL shapes = closed subset for one service

↓

API Contracts
JSON Schema / OpenAPI / GraphQL generated from shapes

↓

Microservices, UIs, AI agents
consume the same meaning

12.2 How DL Constructs Enable Reuse

Construct	Reuse benefit
$C \sqsubseteq D$	API "AutoClaim" automatically IS an enterprise Claim
$C \equiv D$	one definition of HighValueClaim reused by API filter, BI, rules
$R \sqsubseteq S$	specific API fields roll up to enterprise relations
R^-	both API directions (claim→policy, policy→claims)
∘ chains	shortcut fields (claim.policyHolder) are derived, not stored

- Generated contracts stay in sync: change the axiom, regenerate shapes → schemas → SDKs.

12.3 Example: Defined Concept as API

$$\text{HighValueClaim} \equiv \text{Claim} \sqcap \exists \text{claimedAmount.decimal}[> 100000]$$

```
# OpenAPI (generated)
/claims/high-value:
  get:
    summary: Claims that are HighValueClaim
    x-ontology-class: clm:HighValueClaim
    responses:
      200:
        content: application/json:
          schema: { $ref: '#/Claim' }
```

```
# Backend query - uses the definition
SELECT ?c WHERE { ?c a clm:HighValueClaim }
# reasoner / rules keep the class populated
```

12.4 Mapping System Terms

$\text{crm:Customer} \sqsubseteq \text{ent:PartyInRole}$

$\text{claimsys:ClaimRecord} \equiv \text{ent:Claim}$

$\text{pas:PolicyHolderLink} \sqsubseteq \text{ent:hasPolicyholder}$

- Each source keeps its names; DL axioms align them to one enterprise meaning.

12.5 Benefits

① **Single source of truth** : change the definition once, all APIs follow.

② **Consistency proof** : reasoner checks new API concepts do not contradict enterprise ones.

③ **Self-describing APIs** : x-ontology-class links JSON fields to meaning.

④ **AI ready** : agents read the same definitions to call APIs correctly.

Rajesh Kumar Gupta

⑬ Building LLM Systems with DL

13.1 Neuro-Symbolic Architecture

Unstructured input
chat, FNOL email, PDFs, adjuster notes

↓

LLM (neural)
extract entities & facts using ontology vocabulary

↓

ABox assertions
Claim(cl), causedBy(ll, Flood) ...

↓

DL Reasoner + Rules (symbolic)
classify, check consistency, prove

↓

Verified facts + justification
what is true and WHY

↓

LLM (neural)
explain the proof in plain language

13.2 Division of Labour

LLM does	DL reasoner does
Understand language	Guarantee logical correctness
Extract facts / propose axioms	Prove, classify, detect contradictions
Handle fuzzy input	Give exact, repeatable answers
Explain to humans	Produce auditable justifications

13.3 LLM-Drafted Axiom Review Loop

Policy wording clause

↓

LLM drafts DL / Manchester axiom

↓

Reasoner: satisfiable ? consistent ?

↓

Test individuals classify correctly ?

↓

Ontologist approves → Git release

13.4 Five Integration Patterns

1. **LLM → ABox** : extract facts, reasoner validates & infers.
2. **LLM → TBox** : LLM drafts axioms from policy wording; reasoner checks satisfiability; expert approves.
3. **Reasoner as guardrail** : reject LLM answers that contradict the KB.
4. **Reasoner as tool** : agent calls "classify(claim)" / "explain(inference)".
5. **LLM as explainer** : turn justification axioms into a customer letter.

13.5 Code (owlready2 + HermiT)

```
from owlready2 import *
onto = get_ontology("claims.owl").load()

with onto:                  # facts from LLM extraction
    pol  = onto.HomeownerPolicy("POL_77")
    pol.hasStatus = [onto.Active]
    cov  = onto.Coverage("COV_1",
               coversPeril=[onto.Flood])
    pol.hasCoverage = [cov]
    clm  = onto.Claim("CLM_101",
               isAgainst=[pol])

sync_reasoner_hermit(infer_property_values=True)
print(pol.is_a)   # [.., FloodCoveredPolicy,
ActivePolicy]
print(clm.is_a)
```

13.6 Research Signals

- LLMs alone are still inconsistent on contract coverage questions; **LLM + logic program** with guided translation does best.
- Industry is shifting to **neuro-symbolic** insurance AI for explainable decisions.

13.7 Reasoner as an Agent Tool

```
tools = [{
 "name": "classify_claim",
 "description": "Run DL reasoner; return
   inferred classes + justification",
 "input_schema": {"claim_id": "string"} }]
```

Rajesh Kumar Gupta

⑭ Natural Language → Query via Ontology

14.1 NL → Query Pipeline

NL Question
"Show open flood claims over 100k on active home policies"

↓

Entity & Term Linking
flood → :Flood, home policy → :HomeownerPolicy

↓

Schema Retrieval (Ontology RAG)
classes, properties, domain/range, definitions

↓

Type-aware Query Generation
LLM writes SPARQL / Cypher / DL query

↓

Validate
parse + terms exist + domain/range fit

↓

Execute with Reasoning
defined classes expand automatically

↓

Answer + Evidence
rows, IRIs, justification

14.2 Why Ontology Helps

- **Type-aware grounding**: only properties whose domain fits the current variable are offered to the LLM - fewer empty / wrong queries.
- **Defined classes** shrink queries: "high value" = one class, not a FILTER the LLM must guess.

14.3 Typical LLM Query Errors

1. **Invented property** : "hasFloodAmount" does not exist.
2. **Wrong direction** : policy isAgainst claim.
3. **Literal vs IRI** : "Flood" string instead of :Flood.

14.4 Same Question - Three Queries

DL Query (Protégé):

```
FloodClaim and HighValueClaim
 and hasStatus value Open
 and isAgainst some
     (HomeownerPolicy and ActivePolicy)
```

SPARQL (with reasoning on):

```
SELECT ?c WHERE {
  ?c a :FloodClaim , :HighValueClaim ;
     :hasStatus :Open ;
     :isAgainst ?p .
  ?p a :HomeownerPolicy , :ActivePolicy . }
```

Cypher:

```
MATCH (c:Claim:FloodClaim {status:'Open'})
      -[:IS_AGAINST]->
(p:HomeownerPolicy:Active)
WHERE c.claimedAmount > 100000 RETURN
c
```

14.5 Prompt Skeleton

```
SYSTEM: Use ONLY these ontology terms.
CLASSES: {name, definition, superclasses}
PROPERTIES: {name, domain, range}
EXAMPLES: 5 verified NL -> SPARQL pairs
RULES: read-only; prefer defined classes;
       return JSON {query, terms_used}
```

14.6 Validation Checks

1. Query parses (rdflib / ANTLR).
2. All IRIs exist in the ontology.
3. Each triple fits domain / range.
4. Empty result → retry with feedback.

14.7 Benefits

1. **Accuracy** : fewer wrong / empty queries.
2. **Shorter queries** : defined classes replace long filters.
3. **Explainable** : terms_used + reasoning trace.

Rajesh Kumar Gupta

⑮ GraphRAG ↔ Description Logic

15.1 How AI & GraphRAG Help DL

① **Populate the ABox** : extract facts from millions of documents at scale.

② **Ontology learning** : suggest new concepts, synonyms, candidate axioms.

③ **Alignment** : propose mappings between system vocabularies ($C \equiv D$).

④ **Explain proofs** : turn justifications into plain language.

⑤ **Query interface** : natural-language access to a formal KB.

15.2 How DL Helps GraphRAG

① **Typed graph** : retrieval follows meaningful, typed edges.

② **Inferred edges** : reasoner adds hidden links → better multi-hop recall.

③ **Defined classes** : ready-made "views" to retrieve (ReferralClaim).

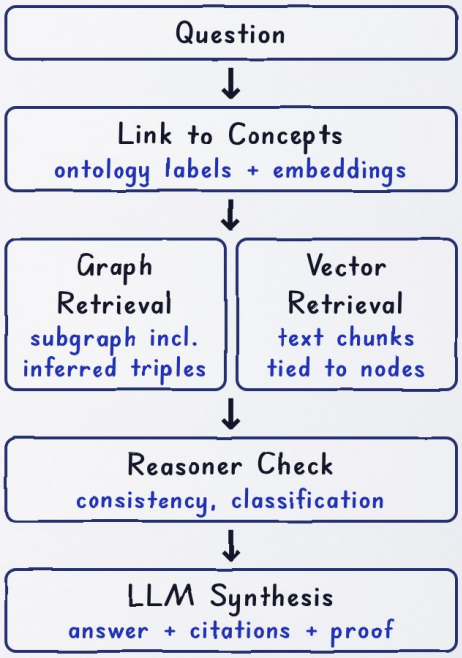
④ **Consistency filter** : drop retrieved / generated facts that contradict the KB.

⑤ **Provenance + proof** : answers cite triples AND the axioms used.

15.3 Benefit Summary

Problem	Without DL	With DL
Hallucination	plausible but wrong	checked vs axioms
Multi-hop	misses implicit links	inferred links
Synonyms	string match	one concept, many labels
Audit	"the model said"	justification set

15.4 DL-Enhanced GraphRAG Flow



15.5 Retrieval Using Inference

```
# before reasoning: claim -> policy only
# after reasoning (role chain):
#  claimHasCoverage = isAgainst o
hasCoverage
SELECT ?cov ?text WHERE {
  :CLM-101 :claimHasCoverage ?cov .
  ?cov :hasWordingChunk ?chunk .
  ?chunk :text ?text }   # context for
LLM
```

15.6 Guardrail Check (Python)

```
def accept(facts, onto):
    with onto: add_assertions(facts)
    try:
        sync_reasoner_hermit()
        return True
    except
OwlReadyInconsistentOntologyError:
        rollback(facts); return False
```

- Inconsistent LLM output is **rejected before** it reaches the user.

Rajesh Kumar Gupta

⑯ Chat & Ontology-based Adjudication

16.1 Ontology Adjudication Flow

1. Chat FNOL
   customer describes the loss
↓
2. LLM Extraction
   facts mapped to ontology terms + evidence
↓
3. Clarify Gaps
   SHACL finds missing data → bot asks
↓
4. ABox Update
   claim, loss, peril, dates, amounts
↓
5. DL Reasoning
   classify: FloodClaim, CoveredPeril, Excluded ...
↓
6. Rules / DMN
   deductible, limits, pay / deny / review
↓
7. Justification
   axioms + facts used for the decision
↓
8. LLM Response
   plain-language answer with citations
↓
9. Human Adjuster
   approves REVIEW & high-value cases

16.2 Concepts the Bot Relies On

- FloodClaim, ActivePolicy, FloodCoveredPolicy
- CoveredClaim, ExcludedClaim, ReferralClaim
- LossInPolicyPeriod, LateReportedClaim

16.3 Sample Chat

Customer: My basement flooded after heavy rain on 14 March, damage about $18,000.

Bot: Sorry to hear that. Is the property at 12 Oak St on policy HO-2026-77 ? Was the water from rain / surface flooding or a burst pipe ?

Customer: Yes, rain - the river overflowed.

Bot: Thanks. Your policy has a Flood coverage endorsement (limit $50,000, deductible $1,000), active on the loss date. This loss is **covered**; estimated payable $17,000, subject to adjuster inspection.
Why: Flood endorsement FL-01 • Policy active 01-Jan to 31-Dec • No flood exclusion.

16.4 Why Ontology Makes Chat Safer

① **Grounded answers** : bot states only facts proven by the KB.
② **Targeted questions** : missing required data drives the next question.
③ **Consistent decisions** : same facts → same result, every time.
④ **Explainable** : every sentence maps to axioms / evidence.
⑤ **Regulatory fit** : auditable trail for complaints & fair-claims rules.

16.5 Guardrails

- The LLM **never decides coverage alone** - it extracts and explains.
- Low confidence or inconsistency → human review.
- Decision text is generated FROM the justification, not free text.

Rajesh Kumar Gupta

⑰ Adjudication Worked Example

17.1 Facts (ABox from Chat + Systems)

HomeownerPolicy(POL77) hasStatus(POL77, Active)
effectiveDate(POL77, 2026-01-01)
expiryDate(POL77, 2026-12-31)
hasCoverage(POL77, FL01) Coverage(FL01)
coversPeril(FL01, Flood) limit(FL01, 50000)
deductible(FL01, 1000)
Claim(CLM101) isAgainst(CLM101, POL77)
arisesFrom(CLM101, L1) causedBy(L1, Flood)
lossDate(L1, 2026-03-14) lossAmount(L1, 18000)

17.2 Axioms + Rules Used

A1: FloodCoveredPolicy ≡ Policy ⊓ ∃hasCoverage.(∃coversPeril.{Flood})

A2: FloodClaim ≡ Claim ⊓ ∃arisesFrom.(∃causedBy.{Flood})

A3: ActivePolicy ≡ Policy ⊓ ∃hasStatus.{Active}

R1 (SWRL): covered peril on same policy → PerilCoveredClaim
R2 (SWRL): loss date within term → LossInPolicyPeriod
R3 (DMN): payable = min(loss - deductible, limit)

17.3 Decision Logic

CoveredClaim ≡ PerilCoveredClaim ⊓ LossInPolicyPeriod
⊓ ∃isAgainst.ActivePolicy ⊓ ¬ExcludedClaim

17.4 Letter Generated From the Proof

Dear customer, your flood claim CLM101 is approved. Your policy HO-2026-77 was active on 14 March 2026 and includes Flood coverage FL01 (limit $50,000). After the $1,000 deductible, we will pay $17,000.

17.5 Inference Trace (Proof)

① POL77 : ActivePolicy : A3 + hasStatus Active.
② POL77 : FloodCoveredPolicy : A1 + FL01 covers Flood.
③ CLM101 : FloodClaim : A2 + L1 causedBy Flood.
④ CLM101 : PerilCoveredClaim : R1: same peril Flood on FL01.
⑤ CLM101 : LossInPolicyPeriod : R2: 2026-03-14 within term.
⑥ ¬ExcludedClaim : closed-world rule: no exclusion found for Flood.
⑦ CLM101 : CoveredClaim : definition 17.3.
⑧ Payable = 17,000 : R3: min(18000 - 1000, 50000).

17.6 Decision Record (JSON)

```
{ "claim": "CLM101",
  "decision": "PAY", "payable": 17000,
  "classes": ["FloodClaim","CoveredClaim"],
  "justification": ["A1","A2","A3",
                    "R1","R2","R3"],
  "evidence": ["chat#msg3","policy FL01"],
  "reviewRequired": false }
```

17.7 Counter-Example

- If POL77 had **hasExclusion** FloodExclusion → ExcludedClaim(CLM101) → DMN gives **DENY**, and the proof cites the exclusion axiom.
- If both flood coverage and flood exclusion exist → reasoner reports **inconsistency** → human review.

Rajesh Kumar Gupta

18 Limits, Best Practices & Tools

18.1 Limits of Description Logic

1. **Tree shape** : cannot relate two branches ("same policy") without rules.
2. **No arithmetic** : sums, date gaps, payments need rules / code.
3. **Open world** : "not present" ≠ "false"; need SHACL / closure.
4. **No UNA** : must assert differentFrom or use keys.
5. **Monotonic** : no exceptions / defaults without rule layers.
6. **Cost** : expressive DLs can be slow on huge KBs - choose a profile.

18.2 Choosing an OWL Profile

Need	Pick
Huge hierarchy, fast classification	OWL 2 EL (ELK)
Query over relational DBs	OWL 2 QL (Ontop)
Rules in triple store at scale	OWL 2 RL (GraphDB, RDFox)
Rich definitions, design-time checks	OWL 2 DL (HermiT, Pellet)

18.3 Tools

Tool	Role
Protégé + DL Query	modeling, explanations
HermiT / Pellet	OWL 2 DL reasoning
ELK / Konclude	fast EL / parallel reasoning
owlready2 / OWL API	Python / Java programming
RDFox / GraphDB	scalable rules

18.4 Learning Path

1. ALC syntax & semantics
2. Protégé + DL Query hands-on
3. Tableau proofs by hand
4. SWRL / SHACL rules
5. owlready2 + LLM extraction
6. GraphRAG with reasoning

18.5 Best Practices

1. **Start from sentences** : write the business rule in NL, then translate; keep both.
2. **Prefer definitions** : ≡ enables automatic classification.
3. **Use some + only together** : avoid vacuous ∀.
4. **Disjointness** : declare siblings disjoint to catch errors.
5. **Test with individuals** : unit-test every defined class.
6. **Layer logic** : DL for meaning, rules for joins / maths, SHACL for data, DMN for decisions.
7. **Keep justifications** : store proofs with decisions.
8. **Human in the loop** : LLM-proposed axioms are reviewed before release.

18.6 Common Mistakes

1. **Domain/range as validation** : they INFER types instead of rejecting data.
2. **⊓ vs ⊔ confusion** : English "and" often means OR.
3. **Overusing nominals** : {a} everywhere slows reasoning.
4. **Roles as classes** : "Claimant" modeled as subclass of Person.
5. **Trusting the LLM** : no reasoner check before writing to KB.

18.7 Key Takeaway

Natural language states the business rule; DL makes it precise; the reasoner proves it; rules fill the gaps; the LLM talks to people.

Rajesh Kumar Gupta

19 Enterprise Ontology Architecture

19.1 Layered Ontology Architecture

Upper / Foundational
BFO, gist, OWL-Time, PROV-O - OWL 2 DL

↓

Enterprise Core Ontology
Party, Role, Agreement, Policy, Coverage, Claim, Peril, Money

↓

Domain Modules
Underwriting, Claims, Billing, Reinsurance (⊑ core)

↓

Defined Concepts & Rules
≡ ActivePolicy, CoveredClaim + SWRL / Datalog

↓

Application Profiles
SHACL shapes per product / service (closed world)

↓

Integration & API Layer
R2RML mappings, OpenAPI, GraphQL, events

19.2 DL Expressivity per Layer

Layer	Profile	Reasoner
Upper + Core	OWL 2 DL (SROIQ)	HermiT (design time)
Large taxonomies	OWL 2 EL	ELK
Runtime data	OWL 2 RL + rules	GraphDB / RDFox
Source DB access	OWL 2 QL	Ontop (virtual)
Data contracts	SHACL	pySHACL / TopBraid

19.3 Architecture Principles

1. **One meaning, many uses** : core definitions are imported, never copied.
2. **Separation of concerns** : DL = meaning, rules = joins & maths, SHACL = data, DMN = decisions.
3. **Right logic per layer** : rich DL at design time, tractable profiles at runtime.
4. **Everything versioned** : ontology, shapes, rules, mappings in Git with CI.

19.4 Runtime Platform Architecture

Ontology Repo
Git + CI

Source Systems
PAS, Claims, CRM

↓

Reasoner + Rule Service
classify, check, prove

Knowledge Graph
triples + named graphs

↓

GraphRAG + Vector Index
typed retrieval

LLM Gateway
extract, query, explain

↓

Consumers
APIs, adjudication chat, agents (MCP), BI

19.5 Core Platform Services

Service	Responsibility
Ontology registry	publish versions, IRIs, docs
Reasoning service	classification, consistency, justifications
Rule service	SWRL / Datalog / DMN execution
Validation service	SHACL on every load and LLM output
Query service	SPARQL / Cypher / NL→query
Provenance service	PROV-O lineage of every fact & decision

19.6 Governance & Lifecycle

Propose → Model (DL) → Reason + Test → Approve → Release

- Ontology board: business owner + ontologist + architect.
- Breaking change = major version + deprecation period.
- Every decision stores ontology version + justification.

19.7 Key Takeaway

The enterprise ontology is the **semantic backbone**: DL defines meaning once, the reasoner keeps it consistent, and every API, rule, graph and LLM reuses it.

Rajesh Kumar Gupta