ARCO: Technical Implementation & Architectural Decisions

System: *Neuro-Symbolic Compliance Verification Engine* ***Version:*** *1.0 (Prototype)* ***Date:*** *December 22, 2025*

---

1. Architectural Decision: Modeling Latent Risk via BFO Dispositions

The Problem: *Traditional compliance tags (e.g., "High Risk: True") fail to capture conditional capabilities. A system may have hardware for biometric identification (High Risk) that is currently disabled via software.* ***The Solution:*** *We utilize* ***BFO 2.0 Dispositions*** *to model capability as a "Realizable Entity" that inheres in the hardware (`Object Aggregate`), independent of its current `Process` realization.* ***Impact:*** *This allows the system to flag "Latent Liability" even when the software is configured to "Off."*

Implementation Artifact (`ARCO_core_gold_fixed_v3.ttl`):

```
#################################################################
# REALITY-SIDE UNIVERSALS (BFO)
# Decision: Model System as Object Aggregate bearing Dispositions
#################################################################

:System rdf:type owl:Class ;
  rdfs:subClassOf bfo:0000027 ; # Object Aggregate
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty ro:0000053 ; # bearer of
    owl:someValuesFrom :CapabilityDisposition
  ] .

:BiometricIdentificationCapability rdf:type owl:Class ;
  rdfs:label "Biometric Identification Capability" ;
  rdfs:subClassOf :CapabilityDisposition .
  # Note: This exists whether realized in a process or not.
```

---

2. Architectural Decision: Structural Integrity via SHACL

The Problem: *OWL Reasoners (Open World Assumption) are poor at validating data completeness. They infer missing data rather than flagging it as an error.* ***The Solution:*** *We*

*inject a **SHACL Validation Layer** (`pyshacl`) prior to the reasoning step. This enforces a "Closed World" constraint on the graph, rejecting any `AssessmentDocumentation` that fails to link a System to its Regulatory Content. **Impact:** Prevents "Garbage In, True Out" scenarios where incomplete data passes audit silently.*

Implementation Artifact (`assessment_documentation_shape.ttl`):

```
#################################################################
# Shape: Enforce Linkage between System and Regulation
#################################################################

:AssessmentDocumentationShape
  a sh:NodeShape ;
  sh:targetClass :AssessmentDocumentation ;

  sh:property [
    sh:path iao:0000136 ;          # is about
    sh:class :System ;
    sh:minCount 1 ;                # Constraint: Must explicitly link to
System
  ] ;

  sh:property [
    sh:path iao:0000136 ;          # is about
    sh:class :RegulatoryContent ;
    sh:minCount 1 ;                # Constraint: Must explicitly link to
Regulation
  ] .
```

---

3. Architectural Decision: Deterministic Audit via SPARQL ASK

***The Problem:** Auditors require binary (Pass/Fail) verdicts. Standard knowledge graph queries (`SELECT`) return lists that require post-processing interpretation, introducing ambiguity. **The Solution:** We utilize **SPARQL ASK** queries which return a boolean `xsd:boolean` value. The query pattern matches the specific absence of compliance or the presence of risk features. **Impact:** The output is mathematically deterministic and machine-readable (JSON Boolean).*

Implementation Artifact (`ask_assessment_doc_process_wiring.sparql`):

```
PREFIX : <https://arco.ai/ontology/core#>
PREFIX iao: <http://purl.obolibrary.org/obo/IAO_>

# Objective: Verify if AssessmentDoc connects System to Annex III Condition
ASK WHERE {
  :AssessmentDoc_001 iao:0000136 :Sentinel_ID_System .
  :AssessmentDoc_001 iao:0000136 :AnnexIII_Condition_Q1 .
}
# Output: TRUE (Verification Successful)
```

---

4. Execution Pipeline (Neuro-Symbolic Bridge)

Orchestration: *The Python runtime (`rdflib` + `pyshacl`) manages the transition from probabilistic extraction (LLM-generated instances) to deterministic validation.*

Pipeline Code (`run_checks.py`):

```
Python
def run_checks(data_graph: Graph):
    # Step 1: Structural Validation (SHACL)
    conforms, _, report_text = validate(
        data_graph=data_graph,
        shacl_graph=Graph().parse(SHAPES),
        inference="none"
    )
    if not conforms:
        raise ValidationError(f"Graph Structure Invalid: {report_text}")

    # Step 2: Compliance Reasoning (SPARQL)
    # Only executes if Structure is Valid
    run_sparql_audit(data_graph)
```