

# Knowledge Base Creation and Curation using LLMs

## Milestone 1 Report

Joshua Cook

October 28, 2024

### Introduction

This milestone report presents the initial implementation of an automated knowledge base creation and curation system utilizing Large Language Models (LLMs). The system's foundation is the Chelle Ubiquitous Language (CUL, Appendix), a comprehensive domain modeling schema that provides a standardized vocabulary for knowledge representation. This first milestone focused on implementing the conceptual layer of the CUL through a series of focused prompts and building the supporting infrastructure. The system demonstrates how LLMs can be used to extract, define, and organize knowledge from organizational documentation while maintaining the strict structural requirements defined by the CUL. This approach represents a significant step toward automated knowledge management that preserves semantic relationships and supports practical implementation.

- video demonstration: ([Link to video](#))
- code repository: ([Link to repo](#))

### 1. System Architecture

The system implements the Chelle Ubiquitous Language (CUL) through a modern web application architecture. The architecture consists of three main components: a backend API layer, a frontend interface, and a containerized development environment.

#### Backend (FastAPI)

- Lightweight API layer for LLM integration using Claude-3-Sonnet
- Endpoints supporting structured prompts for knowledge extraction
- MongoDB for storing concept and relationship data

### Frontend (Streamlit)

- Clean, minimalist interface reflecting CUL structure
- Three main views mapping directly to CUL layers:
  - Concepts Management
  - Relationships Visualization
  - Operational Elements
- Real-time updates and interactive visualizations

### Development Environment

- Docker-based setup ensuring reproducibility
- Key dependencies: Anthropic’s Claude API, NetworkX for graph visualization, Plotly for interactive charts

## 2. Core Prompting Strategy

The current implementation focuses on the Conceptual Layer of the CUL through a prompting strategy. By breaking down the complex task of knowledge extraction into discrete components, the system uses three core prompts that work in concert to identify, research, and define concepts from organizational documentation. This approach mirrors the CUL’s emphasis on clear term structure, with each prompt targeting a specific aspect of concept formation while maintaining the strict dependency ordering required by the language specification.

The CUL enforces a strict hierarchy in concept formation where each component builds upon previously established elements. This is reflected in the prompting strategy in several ways:

- The Concept Identifier prompt targets the most primitive element - the lexeme. It performs pure extraction without requiring any prior concept knowledge, making it the foundation of the knowledge extraction pipeline.
- The Concept Researcher prompt operates independently on the source material but requires the lexeme as context, creating a natural dependency that mirrors the CUL’s requirement that citations be tied to specific terms.
- The Concept Definer prompt represents the final synthesis, requiring both the lexeme and its citations as input. This mirrors the CUL’s requirement that definitions must be constructed using only previously established terms and verified citations.

This structured approach ensures that each component of a concept is properly validated before being assembled into the final concept entity, maintaining the CUL’s emphasis on verifiable and well-ordered knowledge representation.

## Concept Identification

This prompt focuses on extracting lexemes (primitive terms) from organizational documentation:

### ## Task Summary

You are an extractor of terminology from company documentation. Terminology means any named entity, whether publicly known or internal to the company. For something to classify as a term, it must:

- Be a word or phrase that is unlikely to be understood by someone outside the company.
- Be no more than a few words long.
- Be defined within the documentation you are searching.

Your goal is to take a given document and identify all the terms within it. You do not need to extract the definition.

Take care not to include terms that are already present in the list of terms you have already extracted. That list is provided below, under "Current Terms".

Take care not to extract multiple versions of the same term.

## Citation Research

This prompt extracts relevant citations for identified concepts:

### ## Task Summary

You are an extractor of knowledge on particular terminology. Your goal is to take a given context, which is a collection of excerpts from documentation, and a given term, and extract sources from the context that provide facts about the term. The sources must be direct quotes from the context; do not paraphrase.

Requirements:

- Do NOT make up sources. Only use sources that are directly quoted from the context
- The sources you provide must be directly quoted from the context. Do not paraphrase.
- The sources you provide must be relevant to the term you are defining.
- The sources you provide should be no more than a few sentences each.

## Definition Generation

This prompt synthesizes citations into concise definitions:

### ## Task Summary

Your goal is to synthesize given sources about [TARGET\_TERM] into a concise definition. You should include only highly relevant information about [TARGET\_TERM]. Consider a definition this way:

What you are given is all the facts that an expert on [TARGET\_TERM] would know. The definition contains only the information one would need in order to gain a \_functional\_ understanding of what [TARGET\_TERM] is, not to become an expert on it.

To liken it to an API, the sources you are given are the implementation details, and the definition you are creating is the public interface. Someone can read the definition you provide and understand what [TARGET\_TERM] is at a high level, without necessarily learning all the details.

Your definition should not typically be longer than 1-2 sentences.

## 3. Implementation Components

The frontend implementation, built using Streamlit, provides three core interfaces that mirror the structural layers of the CUL. The application is organized into distinct pages that represent each layer of the language specification. The Concepts page handles the foundational knowledge structures, the Relationships page manages the connections and hierarchies between concepts, and the Operational Elements page bridges the gap between abstract knowledge representation and practical implementation. Each page implements specific aspects of the CUL's requirements while maintaining consistency across the system.

### Concepts Management

- Direct implementation of CUL's Conceptual Layer
- Supports concept creation, definition, and citation management
- Foundation for automated knowledge extraction

### Relationships Visualization

- Interactive graph visualization of concept relationships
- Implements CUL's Relationship Layer structure
- Supports relationship types from None through Equivalence
- Calculates and displays relationship strengths

### Operational Elements

The system demonstrates how concepts are realized in practice through:

- Implementations: Concrete realizations of concepts
- Procedures: Step-by-step process definitions
- Tools: Software and platforms used in implementation

Examples shown in the current build include:

- Brand color implementations
- Sales pipeline configurations
- Lead qualification procedures
- Design tool integrations

## 4. Next Steps

Future development will focus on expanding the prompting strategy to fully implement the CUL's relationship and extended conceptual layers. The immediate priority is developing a suite of prompts that can automatically identify and validate relationship types between concepts, calculate relationship strengths, and maintain the complex hierarchy defined by the CUL. This work will be complemented by refinements to the existing conceptual layer prompts, incorporating the updated CUL specifications and adding capabilities for mention detection, knowledge dependency identification, and definition dependency management. In parallel, I will develop comprehensive validation and testing frameworks to ensure the accuracy and effectiveness of the prompt-based approach. These frameworks will include automated validation of prompt outputs, relationship consistency testing, and quantitative metrics for evaluating prompt effectiveness.

### Relationship Layer Implementation

- Develop prompts for relationship type identification
- Implement validation of relationship constraints
- Create prompts for relationship strength calculation
- Build relationship hierarchy management

### Extended Conceptual Layer

- Refine existing prompts based on updated CUL
- Develop additional prompts for:
  - Mention detection and processing
  - Knowledge dependency identification
  - Definition dependency management

### Validation and Testing

- Implement automated validation of prompt outputs
- Create test suite for relationship consistency
- Develop metrics for prompt effectiveness

## Appendix: Code Structure

The codebase is organized into several key components, each implementing specific aspects of the CUL:

### API Layer

**api/routers/chat.py** Handles LLM integration and prompt management

- Configures Claude-3-Sonnet
- Manages prompt routing and responses
- Implements structured output parsing

### Frontend Views

**frontend/pages/\*.py** Three main views implementing CUL layers:

- **1\_Concepts.py** - Concept management and definition
- **2\_Relationships.py** - Relationship visualization and management
- **3\_Operational\_Elements.py** - Implementation tracking

### Infrastructure

**docker/Dockerfile** Ensures consistent environment across deployments:

- Includes all required dependencies for LLM integration
- Configures development and testing environment

The system is designed to be modular and extensible, allowing for future implementation of additional CUL components as the project progresses.

## Appendix: Chelle Ubiquitous Language

### Purpose

Chelle manages an organization's information via construction of a ubiquitous language (UL) for that organization.

A UL is used for:

- Encoding an organization's knowledge (concepts, definitions, facts, relationships)
- Organizing an organization's documents (documents are tagged by concept)

### Document Conventions

In this document, the following conventions apply, which should help in navigating:

- The sub-headings are the terms contained in the Glossary, and the content within is the complete definition of the term, followed by the facts we know about the term.
- When a definition or fact makes reference to another term in the Glossary, the term is *italicized*. A term is **bolded** in its own definition.
  - Example: '**Understanding** is a degree of demonstrated retention of a *Concept*.' *Concept* is italicized here because it is a term belonging to the Glossary. **Understanding** is bolded because this is its own definition.
- When a definition makes use of customized language in defining a property of the term, the customized language is put in "quotes". For example, if a term from the Glossary defines an entity which has a property that may only contain certain values, and those values are enumerated, they would be quoted. Customized language from the definition of one term may appear in the definition of another term.
  - An example of this can be found in the *Concept* term, where we use the word "lexeme". That word is a child of the term *Concept*, so it's not big enough to be its own term in the Glossary, but it may need to be referenced elsewhere, so the quotes help identify it.

## Core Principles

### Dependency Ordering

A term  $T_1$  must be defined before term  $T_2$  if and only if  $T_2$ 's definition contains  $T_1$ . This creates a directed acyclic graph (DAG) of definitional dependencies.

### Term Structure

Each term entry consists of two parts:

- Definition: A precise statement that uniquely identifies the term using only previously defined terms
- Citations: Verified statements about the term that document its properties, behaviors, and relationships

## Completeness Rule

The glossary  $G$  is complete if and only if:

- Every term used in any definition is itself defined earlier in  $G$
- Every term has exactly one definition
- No definition creates a circular dependency
- Every citation has a verifiable source

## Relationship Uniqueness Rule

The system maintains exactly one explicit relationship between any pair of Concepts:

Every pair of Concepts  $(A, B)$  must have exactly one Relationship Type  $R$

Relationship Types are mutually exclusive

No implicit or derived relationships are stored

## Mathematical Foundations

**Lattice Operations** The relationship hierarchy forms a complete lattice with the following operations:

**Join Operation** ( $\cup$ ) Definition: For any two relationship types  $R_1$  and  $R_2$ , their join  $R_1 \cup R_2$  is the least upper bound that satisfies both relationships

Formally:  $R_1 \cup R_2 = \min\{R | R \geq R_1 \text{ and } R \geq R_2\}$

**Meet Operation** ( $\cap$ ) Definition: For any two relationship types  $R_1$  and  $R_2$ , their meet  $R_1 \cap R_2$  is the greatest lower bound that is satisfied by both relationships

Formally:  $R_1 \cap R_2 = \max\{R | R \leq R_1 \text{ and } R \leq R_2\}$

## Knowledge System

**Knowledge** Definition: The fundamental unit of transferable information.

Citations:

- Has identifiable source
- Can be validated
- Can be versioned
- Can be structured
- Can be referenced

**Knowledge Structure** Definition: A Knowledge Structure is a formal representation system for storing and managing *Knowledge*.

Citations:

- Must be versioned with unique identifier
- Must maintain referential integrity
- Must be computably verifiable



Properties:

- Consistency: No contradictory statements allowed
- Completeness: All referenced Concepts must exist
- Computability: All operations must terminate in finite time

Validation Rules:

- All references must be resolvable
- Version changes must maintain dependency consistency
- All operations must have specified complexity bounds

**Knowledge Operations** Definition: A set of permitted operations for manipulating *Knowledge Structures*.

Operations:

- Create: Initialize new Knowledge Structure
- Update: Modify existing Knowledge Structure
- Compose: Combine multiple Knowledge Structures
- Validate: Verify Knowledge Structure consistency

Constraints:

- All operations must preserve Knowledge Structure properties
- All operations must be reversible
- All operations must maintain audit trail

## Term Categories

### Primitive Terms

- Terms that require no other terms in their definition
- Must appear first in the glossary
- Examples: “Lexeme”, “Citation”

### Composite Terms

- Terms that require other terms to be defined first
- Must appear after all terms referenced in their definition
- Example: “Concept” - requires both “Lexeme” and “Citation” in its definition

### Relationship Terms

- Terms that define connections between other terms
- Must appear after all terms they connect
- Example: “Knowledge Dependency” - requires both “Concept” and “Understanding”

## Primitive Terms

**Lexeme** Definition: A **Lexeme** is a unit of meaning in a language, consisting of a word or group of words.

**Citation** Definition: A **Citation** is a verifiable statement that documents a property, behavior, or relationship of a term within its organizational context.

Validations:

- Every Citation must: a. Reference only defined terms b. Be computably verifiable c. Have an explicit source or derivation

**Property** Definition: A Property is a verifiable characteristic or attribute that can be tested for presence or absence.

Classifications:

- Primitive Properties:
  - Cannot be derived from other properties
  - Must have direct verification procedure
  - Must be atomic (cannot be decomposed)
- Derived Properties:
  - Must specify source properties explicitly
  - Must provide derivation function
  - Must maintain traceable derivation chain

Computably Verifiable Requirements:

- Must have an algorithm  $A$  that terminates in finite time
- $A(x)$  returns true  $\iff$   $x$  has property  $P$
- $A(x)$  returns false  $\iff$   $x$  does not have property  $P$
- Must have specified worst-case complexity bound  $\mathcal{O}(f(n))$

**Equivalence** Definition: A binary relation that is reflexive, symmetric, and transitive.

Citations:

- For any  $x$ ,  $x$  is equivalent to itself (reflexivity)
- If  $x$  is equivalent to  $y$ , then  $y$  is equivalent to  $x$  (symmetry)
- If  $x$  is equivalent to  $y$  and  $y$  is equivalent to  $z$ , then  $x$  is equivalent to  $z$  (transitivity)

## Basic Terms

**User** Definition: A **User** is an individual person who is using the product via a logged in account.

- **Users** are managed externally to the product, via Clerk.
- **Users** can be either *Mentors* or *Learners*.
- what about admins/users in Clerk?

**Organization** Definition: An **Organization** is a group of Users that use the product in tandem.

- **Organizations** are managed externally to the product, via Clerk.
- It is not possible to use Chelle and not belong to an **Organization**

**Integration** Definition: An **Integration** is an authorized connection between the product and an *Organization's* account on a third party platform for the purpose of ingesting data.

- **Integrations** fall into categories: documentation, codebases, conversations, and meets.
- Connecting an **Integration** does not automatically “register” all *Assets* it provides access to.

**Asset** Definition: An **Asset** is a raw source of information.

- An **Asset** may be text, image, video, audio, or structured data.
- It may be uploaded directly by a *User* or acquired via an *Integration*.

**Definition** Definition: A **Definition** is a statement composed of a *Lexeme* and its essential *Citations* that uniquely identify the term.

Citations:

- A Definition must be expressed using only previously defined terms or primitive concepts
- A Definition must be decidable (can be verified in finite time)
- A Definition must be unambiguous
- A Definition must use only defined terms or primitive concepts

Validations:

- Every term used must either be: a. A primitive term b. Previously defined in the glossary c. Explicitly marked as external to the system
- All Citations must be verifiable
- Citations must be minimal (contain only essential distinguishing characteristics)

## Core Concept Structure

**Concept** Definition: A **Concept** is a *Lexeme* with an associated *Definition* owned by an *Organization*.

Citations:

- A **Concept** is identified either manually by a *Mentor*, or automatically via AI.
- A **Concept** contains *Knowledge*.
- A **Concept** is associated with other Concepts through *Mentions* in its *Knowledge*.
- A **Concept** has an associated “Summative” *Assessment*.
- A **Concept** may have a “Formative” *Assessment* constructed for it ad-hoc.

- Two **Concepts** from different *Organizations* may have identical “lexemes”, but they must have different identities.
- A **Concept** may have “synonyms”, which are alternative “lexemes”.

**Glossary** Definition: A **Glossary** is the set of *Concepts* owned by a particular *Organization*.

Citations:

- The **Glossary** is the primary interface through which *Users* interact with *Knowledge*.

## Relationship Foundation

**Relationship Type** Definition: A **Relationship Type** is a formal classification of how two *Concepts* relate to each other within a *Glossary*.

Citations:

- Every relationship type has a precise mathematical definition
- Relationship types form a complete lattice
- Relationship types are mutually exclusive
- Any two Concepts must have exactly one Relationship Type between them

Validation Rules:

- Attempting to add a relationship when one exists must be rejected
- Relationship changes must be atomic operations
- All relationship changes must maintain audit trail

**Relationship Category** Definition: A **Relationship Category** is a grouping of *Relationship Types* that share common mathematical properties.

Citations:

- “Equivalence Category” contains relationships that are reflexive, symmetric, and transitive
- “Order Category” contains relationships that are antisymmetric and transitive
- “Similarity Category” contains relationships that are symmetric but not necessarily transitive
- “Distinction Category” contains relationships that are symmetric and anti-transitive

**Relationship Classification** Definition: A **Relationship Classification** is a formal system that categorizes all possible *Relationship Types* between *Concepts* in a *Glossary*.

Citations:

- Forms a complete lattice with well-defined meet and join operations
- Preserves the relationship hierarchy
- Enforces mutual exclusivity between classifications

## Relationship Types

**None** Definition: A **None** is a *Relationship Type* indicating the absence of any semantic connection between two *Concepts*.

Citations:

- None is symmetric
- None is the minimal element in the relationship lattice
- If A has None relationship with B, then A and B share no properties

**Disjoint** Definition: A **Disjoint** is a *Relationship Type* where two *Concepts* have no properties in common.

Citations:

- Disjoint is symmetric
- If A is disjoint with B, and B subsumes C, then A is disjoint with C
- Disjoint *Concepts* cannot have common instances

**Overlap** Definition: An **Overlap** is a *Relationship Type* where two *Concepts* share some but not all properties.

Citations:

- Overlap is symmetric
- Overlap is not transitive
- Two overlapping *Concepts* must have at least one common property

**Related** Definition: A **Related** is a *Relationship Type* where two *Concepts* have a meaningful semantic connection that is weaker than *Overlap* but stronger than *Disjoint*.

Citations:

- Related is symmetric
- Related is not transitive
- Related requires explicit categorization via a “Connection Type”

Interaction Rules:

- Related relationships can coexist with stronger relationships
- If A Subsumes B, then A Related B is implied with Connection Type “Functional”
- If A Overlaps B, then A Related B is implied with Connection Type “Spatial”
- Related relationships cannot contradict stronger relationships
- Multiple Related relationships with different Connection Types are allowed if consistent

## Connection Type

Definition: A **Connection Type** is a classification that specifies the semantic nature of a *Related* relationship between *Concepts*.

Citations:

- Valid “Connection Types” are: “Causal”, “Temporal”, “Spatial”, “Functional”
- Each Connection Type has specific validation rules
- Connection Types are enumerable and extensible per *Organization*

Axioms:

- Distinctness: Each Connection Type is distinct and mutually exclusive
- Finite Enumeration: The set of Connection Types is finite and enumerable
- Extensibility: New Connection Types can be added but must satisfy all axioms
- Verification: Each Connection Type must have a decidable verification procedure

Dependencies:

- Connection Types must form a directed acyclic graph (DAG)
- Each Connection Type must declare its dependencies explicitly
- No circular dependencies are permitted

**Subsumption** Definition: A **Subsumption** is a *Relationship Type* where the source *Concept* (parent) completely contains all properties of the target *Concept* (child).

Definition: A Subsumption is a directed Relationship Type where the source Concept (parent) completely contains all properties of the target Concept (child).

Citations:

- Direction is explicit: A Subsumes B means A is parent, B is child
- When A Subsumes B, B is subsumed by A (but we don’t store this as a separate relationship)
- The relationship is stored only once with direction indicator
- The relationship strength is always calculated from parent to child

Storage Requirements:

- Store as directed edge: (source, target, SUBSUMES)
- Query support must handle both directions:
  - Get all concepts that subsume X
  - Get all concepts that are subsumed by X

Property Inheritance Rules:

- Complete Inheritance: If A Subsumes B, then B inherits all properties of A
- Local Override: B may strengthen but not weaken inherited properties
- Transitivity: If A Subsumes B and B Subsumes C, then C inherits from both A and B

Validation Requirements:

- All properties from parent concept must exist in child concept
- Child concept properties must satisfy parent concept constraints
- Child may add properties but cannot remove inherited ones
- Child may strengthen but cannot weaken inherited properties

Validation Rules:

For any A Subsumes B:

- A is the parent/broader concept
- B is the child/narrower concept
- B inherits all properties from A
- B may add additional properties
- B may not remove A's properties

**Equivalence (as Relationship Type)** Definition: An **Equivalence** is a *Relationship Type* where two *Concepts* share exactly the same set of essential *Citations* in their *Definitions*.

Citations:

- Inherits mathematical properties from primitive Equivalence relation:
  - Reflexive: Every Concept is equivalent to itself
  - Symmetric: If A is equivalent to B, then B is equivalent to A
  - Transitive: If A is equivalent to B and B is equivalent to C, then A is equivalent to C
- Has *Relationship Strength* of 1.0
- Is the maximal element in the *Relationship Hierarchy*
- Creates equivalence classes within a *Glossary*
- Cannot exist between Concepts from different Organizations (by definition of Concept)
- Must satisfy strict Citation matching (not just semantic similarity)

Validations:

- All essential Citations must match exactly
- Both Concepts must belong to same Organization
- Equivalence must be computably verifiable

## Relationship Properties

**Relationship Strength** Definition: A **Relationship Strength** is a measure of the semantic coupling between two *Concepts* connected by a *Relationship Type*.

Citations:

- Strength is normalized on a [0,1] scale where 1 represents *Equivalence* and 0 represents *None*
- Strength is monotonically decreasing along the relationship hierarchy
- Strength can be computed automatically or specified manually by a *Mentor*

**Relationship Constraint** Definition: A **Relationship Constraint** is a rule that must be satisfied for a particular *Relationship Type* to be valid between two *Concepts*.

Citations:

- Constraints are validated during relationship creation and modification

- Constraints may reference *Citations* as evidence
- Constraints must be computably verifiable

**Relationship Hierarchy** Definition: A **Relationship Hierarchy** is an ordered structure that defines the relative strength and implications of all *Relationship Types* within a *Glossary*.

Citations:

- Forms a complete lattice from *Equivalence* (maximal) to *None* (minimal)
- Each level implies all weaker relationships below it
- Preserves transitive properties across levels

The relationship types form a complete lattice:

1. Equivalence
2. Subsumption
3. Overlap
4. Related
5. Disjoint
6. None

Each level in this hierarchy is strictly weaker than the ones above it and strictly stronger than the ones below it. This forms a total order over relationship types.

**Cross-Reference Rules** All relationships must be:

- Single (Exactly one relationship between any two Concepts)
- Well-defined (have precise mathematical semantics)
- Decidable (can be computed in finite time)
- Consistent (preserve logical constraints)

Relationship composition must preserve:

- Transitivity where applicable
- Symmetry properties
- Hierarchical constraints

Validation requirements:

- Every relationship must be validated
- Validation must be deterministic
- Failed validations must be actionable

**Relationship Strength Calculations** Definition: A formal system for computing and resolving relationship strengths between *Concepts*.

Base Strength Values:

- Equivalent: 1.0
- Subsumes: 0.8



- Overlap: 0.6
- Related: 0.4
- Disjoint: 0.2
- None: 0.0

Modifier Rules:

- Property overlap ratio:  $\text{strength} *= (\text{shared\_properties} / \text{total\_properties})$
- Connection Type weight: if Related, apply weight based on Connection Type
- Citation overlap:  $\text{strength} *= (\text{shared\_citations} / \text{total\_citations})$

Transitivity Rules:

- Direct Rule:  $\text{strength}(A, C) \geq \text{strength}(A, B) \text{ strength}(B, C)$
- Path Rule: strength along any path  $\leq$  minimum strength of any edge
- Resolution Rule: actual strength = maximum strength across all paths

## Mention System

**Mention Resolution** Definition: A formal system for identifying and managing references between *Concepts*.

Resolution Rules:

- Exact match: Direct lexeme correspondence
- Synonym match: Alternative lexeme forms
- Context match: Semantic equivalence in context

Validation Requirements:

- All mentions must resolve to exactly one Concept
- Ambiguous mentions must be explicitly resolved
- Circular mentions must be prevented

**Mention Processing** Definition: The system for processing and maintaining *Mentions* between *Concepts*.

Processing Steps:

1. Detection
  1. Identify potential mentions in text
  2. Validate against known lexemes
  3. Record context and metadata
2. Resolution
3. Maintenance
4. Resolve to specific Concept
5. Handle ambiguity
6. Validate consistency
7. Track mention validity
8. Update on Concept changes
9. Maintain dependency graph

Validation States:

- “Resolved”: Unique Concept identified
- “Ambiguous”: Multiple possible Concepts
- “Invalid”: No matching Concept

**Relationship Validation** Definition: Formal procedures for validating relationship operations.

Required Checks:

1. Uniqueness Validation
  1. Verify no existing relationship before creation
  2. Verify exactly one relationship exists after operation
  3. Verify no implicit relationships are stored
2. Consistency Validation
  4. Verify relationship is valid for both Concepts
  5. Verify all constraints are maintained
  6. Verify strength hierarchy is respected
3. Dependency Validation
  7. Verify all dependent relationships remain valid
  8. Verify no circular dependencies created
  9. Verify transitive properties maintained

Error States:

- “DuplicateRelationship”: Attempt to add relationship when one exists
- “InvalidTransition”: Invalid relationship change attempted
- “ConstraintViolation”: Operation violates system constraints

Resolution Requirements:

- All errors must be actionable
- Error messages must be specific
- Recovery procedures must be defined

## Mapping

**Mapping** Definition: A **Mapping** is a correspondence that associates elements of one set with elements of another set while preserving their *Relationship Types*.

Citations:

1. Structural Properties:
  - Every mapping has exactly one source and one target
  - A mapping may have associated metadata
  - A mapping preserves structural properties between sets
2. Relationship Consistency Rules:
  - If A maps to A' and B maps to B', then:
    - If A is *Equivalent* to B, then A' must be *Equivalent* to B'
    - If A *Subsumes* B, then A' must *Subsume* B'
    - If A *Overlaps* with B, then A' must *Overlap* with B'
    - If A is *Related* to B, then A' must be *Related* to B' with the same “Connection Type”
    - If A is *Disjoint* from B, then A' must be *Disjoint* from B'
    - If A has *None* relationship with B, then A' must have *None* relationship with B'
3. Validation Requirements:
  - All relationship consistency rules must be verifiable in finite time
  - Violations must be detected and reported with specific details
  - Mapping is invalid if any relationship consistency rule is violated
4. Relationship Strength Preservation:

- For any two mapped concepts, their *Relationship Strength* must be preserved within a specified tolerance
  - The tolerance must be explicitly defined in the mapping metadata
  - Violations of strength preservation must be logged and flagged
5. Categorical Constraints:
- Mappings must preserve *Relationship Category* memberships
  - If a concept belongs to a category in the source, its mapped counterpart must belong to the equivalent category in the target
  - Category preservation violations invalidate the mapping

## Mapping Validation

Definition: A **Mapping Validation** is a process that verifies the consistency of a *Mapping* with respect to all *Relationship Types* and constraints.

Citations:

1. Validation Process:
  - Checks all pairs of mapped concepts for relationship preservation
  - Verifies category membership preservation
  - Validates relationship strength preservation
  - Confirms connection type consistency
2. Validation Results:
  - Produces a detailed report of all violations
  - Categorizes violations by severity
  - Provides specific remediation suggestions
3. Validation States:
  - “Valid”: All relationship consistency rules are satisfied
  - “Warning”: Relationship strengths vary but within tolerance
  - “Invalid”: One or more relationship consistency rules are violated

## Operational Layer

**Operational Layer** Definition: The **Operational Layer** provides the implementation framework for transforming abstract *Concepts* and *Relationships* into concrete knowledge entities within an organization.

Citations:

- Must maintain bidirectional traceability between conceptual and operational elements
- Must enforce validation at every transformation step
- Must provide verifiable implementation patterns
- Must preserve semantic relationships during operationalization

## Operational Layer Components

**Implementations** Definition: An Implementation is a concrete realization of a *Concept*.

Citations:

- Must specify measurable properties
- Must define completion criteria
- Must have clear categorization
- Must track execution states

Examples:

- Concept: “Primary Brand Colors”, Slate Blue, #5B7C99; Apricot, #ED820E
- Concept: “Sales Pipeline” Implementation: Hubspot deal stages configuration with specific probability percentages
- Concept: “Customer Segmentation” Implementation: SQL views defining customer cohorts based on revenue/engagement
- Concept: “Sales Territory” Implementation: Geospatial mapping rules in CRM with assigned representatives
- Concept: “Brand Voice” Implementation: GPT prompt template with specific tone parameters
- Concept: “Campaign Performance” Implementation: Real-time dashboard with defined KPI calculations
- Concept: “Content Calendar” Implementation: Structured JSON schema for content planning
- Concept: “Learning Outcome” Implementation: Bloom’s Taxonomy classifier with success criteria
- Concept: “Student Progress” Implementation: Weighted scoring algorithm across assessment types
- Concept: “Curriculum Sequence” Implementation: Directed acyclic graph of prerequisite relationships
- Concept: “Code Quality” Implementation: SonarQube ruleset configuration
- Concept: “System Architecture” Implementation: Infrastructure-as-code templates with security policies
- Concept: “Release Process” Implementation: GitHub Actions workflow definitions

**Procedures** Definition: A Procedure is a defined sequence of steps that implements a *Concept*.

Citations:

- Must be step-by-step reproducible
- Must have defined outcomes
- Must have verifiable steps
- Must capture exceptions handling

Examples:

- Concept: “Lead Qualification” Procedure:
  1. Check company size and industry match
  2. Verify budget authority
  3. Score against ideal customer profile
  4. Route to appropriate sales team
- Concept: “Campaign Launch” Procedure:
  1. Validate creative assets against brand guidelines

- 2. Configure audience targeting parameters
- 3. Set up tracking pixels and conversion events
- 4. Schedule coordinated content distribution
- Concept: “Assessment Creation” Procedure:
  - 1. Map questions to learning objectives
  - 2. Generate difficulty ratings
  - 3. Create rubric with scoring criteria
  - 4. Validate with peer review
- Concept: “Service Deployment” Procedure:
  - 1. Run security scan on dependencies
  - 2. Execute integration test suite
  - 3. Update configuration in target environment
  - 4. Perform canary deployment

**Constraints** Definition: A Constraint is an operational rule that enforces integrity when working with a concept.

Citations:

- Must have stated limitations
- Must be explicitly bounded
- Must have verifiable conditions
- Must specify scope

Examples:

- Concept: “Discount Authorization” Constraints:
  - Maximum discount percentage by deal size
  - Required approvals above thresholds
  - Margin preservation rules
- Concept: “Ad Spend” Constraints:
  - Budget allocation limits by channel
  - Minimum ROAS thresholds
  - Frequency capping rules
- Concept: “Class Size” Constraints:
  - Maximum student-to-teacher ratios
  - Required support staff thresholds
  - Physical space requirements
- Concept: “API Performance” Constraints:
  - Maximum response time limits
  - Rate limiting rules
  - Concurrent connection caps

**Validations** Definition: A Validation is a process that verifies the correctness of operational elements against their conceptual definitions.

Citations:

- Must define test cases
- Must have completion states
- Must track acceptance criteria

- Must verify against source material

Examples:

- Concept: “Revenue Recognition” Validations:
  - Contract terms completeness check
  - Payment schedule verification
  - Multi-currency conversion accuracy
- Concept: “Attribution Model” Validations:
  - Touch point data integrity
  - Channel classification accuracy
  - Conversion path completeness
- Concept: “Student Achievement” Validations:
  - Assessment reliability metrics
  - Progress trend analysis
  - Comparative cohort performance
- Concept: “Code Review” Validations:
  - Test coverage metrics
  - Static analysis results
  - Performance benchmark compliance

**Standards** Definition: A Standard is a formalized set of requirements that operational elements must satisfy.

Citations:

- Must specify quality metrics
- Must define acceptance levels
- Must have measurable outcomes
- Must maintain traceability

Examples:

- Concept: “Sales Documentation” Standards:
  - Required fields for opportunity records
  - Meeting note format and storage
  - Deal stage transition criteria
- Concept: “Digital Assets” Standards:
  - Image resolution and format specifications
  - Metadata tagging requirements
  - File naming conventions
- Concept: “Course Material” Standards:
  - Accessibility compliance requirements
  - Learning objective format
  - Content review cycle
- Concept: “Code Style” Standards:
  - Language-specific formatting rules
  - Documentation requirements
  - Commit message format

**Templates** Definition: A Template is a standardized pattern for implementing conceptual elements consistently.

Citations:

- Must have reusable elements
- Must be parameterized
- Must maintain consistency
- Must track variations

Examples:

- Concept: “Proposal” Templates:
  - Solution architecture diagrams
  - Pricing configuration tables
  - Implementation timeline
- Concept: “Email Campaign” Templates:
  - Responsive email layouts
  - A/B test configuration
  - Performance report format
- Concept: “Lesson Plan” Templates:
  - Activity sequence structure
  - Resource requirement checklist
  - Assessment rubric format
- Concept: “Technical Design” Templates:
  - Architecture decision records
  - API documentation format
  - Deployment runbook structure

**Data** Definition: Definition: Data is the literal content extracted from organizational Assets, stored within the system in a structured format.

Citations:

- Must be versioned with source Asset reference
- Must maintain original fidelity
- Must be queryable
- Must track extraction confidence

Examples:

1. The complete semantic layering model extracted from our UX team’s accessibility standards documentation
2. A mapping of regulatory compliance requirements to specific engineering design patterns from our security handbook
3. The statistical correlation between customer support response times and renewal rates from our quarterly business review
4. Student progression patterns through prerequisite chains extracted from three years of course completion data
5. The decision matrix for market segment targeting derived from our annual strategic planning session

6. Manufacturing tolerance specifications and their relationship to customer satisfaction metrics from product quality reports

**Data Store** Definition: A Data Store defines connection patterns to external knowledge repositories where organizational concepts are maintained.

Examples:

1. Connection to the design system repository where brand identity concepts are maintained as living documentation
2. Integration with the HR knowledge base that maintains our evolving organizational structure and role definitions
3. Link to our research department's experiment tracking system containing hypothesis validation patterns
4. Reference to the sales enablement platform where our value proposition frameworks are continuously refined
5. Connection to our medical knowledge graph maintaining symptom-treatment relationship patterns
6. Integration with our legal team's compliance database tracking regulatory requirement interpretations

**Figure** Definition: A Figure is a visual representation of operational elements and their relationships.

Citations:

- Must have clear notation
- Must be consistent
- Must show relationships
- Must maintain accuracy

Examples:

- Concept: "Sales Process" Figures:
  - Pipeline stage flow diagram
  - Territory mapping visualization
  - Forecast modeling charts
- Concept: "User Journey" Figures:
  - Touchpoint sequence diagram
  - Channel attribution sankey
  - Engagement funnel visualization
- Concept: "Learning Path" Figures:
  - Prerequisite relationship graph
  - Skill progression tree
  - Mastery level indicators
- Concept: "System Architecture" Figures:
  - Component relationship diagram
  - Data flow visualization
  - Infrastructure topology map



## Tools

Definition: A Tool is a software application or platform used to implement and manage operational elements.

Examples

- Figma: UI/UX implementation
- Miro: System visualization
- LucidChart: Process mapping
- NextJS: Web application framework
- GitHub: Code and configuration management
- Docker: Containerization platform
- HuggingFace: Model deployment
- LangChain: LLM orchestration
- Weights & Biases: Experiment tracking
- HubSpot: CRM implementation
- Salesforce: Sales process automation
- Marketo: Marketing automation

## Operational Element

Definition: An Operational Element is any concrete implementation component that realizes a conceptual or relationship layer construct.

Properties:

- Type classification
- Source reference
- Extraction confidence
- Related concepts
- Version information
- Usage context
- Review status

## Cross-Reference Phase

Definition: A Cross-Reference Phase is a systematic process for ensuring consistency between operational elements.

Steps:

1. Check for dependencies between operational elements
2. Verify consistency across components
3. Identify conflicts or overlaps
4. Document relationships
5. Update related concept definitions if needed

## Validation Phase

Definition: A Validation Phase is a structured process for verifying operational integrity.

Steps:

1. Verify completeness of extraction
2. Check consistency with concept definition
3. Validate against source material
4. Review with subject matter experts
5. Document confidence level

## Implementation Rules

### Completeness Requirements:

- All conceptual elements must have corresponding implementations
- All relationships must have operational representations
- All validations must be executable
- All constraints must be enforceable

### Consistency Requirements:

- Implementations must preserve conceptual semantics
- Operational relationships must mirror conceptual relationships
- Validation results must be reproducible
- Standards must be uniformly applied

### Traceability Requirements:

- Every operational element must link to its conceptual source
- All transformations must be documented
- Implementation decisions must be justified
- Validation results must be preserved

## Instruction Layer

### Core System Roles

**Role Definition:** A Role is a constrained set of permissions and capabilities that a User may assume within the system.

Citations:

- Two fundamental types: Mentor and Learner
- Determines available operations and access rights
- Must be explicitly tracked and validated
- Has defined transition rules between types

**Active Role Definition:** An Active Role represents the current Role assumed by a User within their App Session.

Citations:

- Must be explicitly set within each App Session
- Can be changed based on User permissions
- Determines available operations and interfaces
- Must maintain audit trail of changes
- Must enforce type-specific constraints

**Job Definition:** A Job is the professional position and responsibilities of a User within their Organization.

Citations:

- Must be formally specified with measurable outcomes
- Creates context for work assignments
- Must be validated against Organization structure
- Determines required capabilities and knowledge access

### Assessment Framework

**Assessment Type Definition:** An Assessment Type is a classification of evaluation methods used to measure capability.

Citations:

- Two distinct categories:
  - “Formative”: Ongoing learning measurement
  - “Summative”: Final capability validation
- Question format limited to multiple choice
- Results must be computably verifiable
- Must maintain validation trace to source material

**Assessment Definition:** An Assessment is a structured collection of questions of a specific Assessment Type used to measure capability.

Citations:

- Must map to specific measurement levels
- Must maintain validation trace to source material
- Results must be computably verifiable
- Must have clear completion criteria
- Must track attempt history

**Understanding Level Definition:** An Understanding Level is a quantified measure of demonstrated capability.

Citations:

- Two classification schemes:
  - “Practical”: None, Functional, Practical, Proficient
  - “Theoretical”: None, Basic, Intermediate, Mastery
- Levels must form a strict total order
- Each level must have verifiable criteria
- Transitions between levels must be validated

**Understanding Definition:** Understanding is a measured level of demonstrated capability regarding a Concept, validated through Assessment.

Citations:

- Must be quantifiably measurable via Assessment

- Level determined by Assessment Type and performance
- Must have clear progression criteria
- Must track historical development
- Must be regularly validated

**Knowledge Definition:** Knowledge is the complete set of validated instructional materials associated with a Concept.

Citations:

- Must maintain versioning and provenance
- May be AI-generated or User-created
- Storage requirements:
  - Primary storage as structured text
  - Support for multimedia delivery
- Must track content relationships
- Must maintain consistency with source material

**Evaluation Definition:** An Evaluation is a formal measurement using Assessment to quantify current Understanding.

Citations:

- Distinct from regular Assessment in:
  - Focus on measurement rather than learning
  - Can be administered at any time
  - Results affect progression tracking
- Must provide quantifiable results
- Must track historical progression

**Task Definition:** A Task is a Job-relevant work assignment that demonstrates Understanding.

Citations:

- Must have clear completion criteria
- Must be relevant to assigned Job
- Must maintain completion audit trail
- Must have validation requirements
- Requires specific evidence for completion

## Role Implementations

**Learner Definition:** A Learner is a Role focused on Knowledge acquisition and Understanding demonstration.

Citations:

- Primary interaction through structured learning
- Progress tracked via Understanding Levels
- Access to Knowledge consumption interfaces
- Can attempt Assessments
- Must complete assigned Tasks

**Mentor Definition:** A Mentor is a Role responsible for Knowledge curation and learning oversight.

Citations:

- Manages learning materials
- Creates and maintains learning structures
- Assigns Tasks to Learners
- Monitors Understanding progression
- Can modify content relationships

**Authority Definition:** The Authority is a designation given to a User who owns specific Knowledge.

Citations:

- Can be assigned by any Mentor
- Responsible for Knowledge validation
- Maintains approval on Understanding criteria
- Must have relevant Job context

## Learning Structure

**Guide Definition:** A Guide is a structured learning unit that packages related Concepts, Assessments, and Tasks into a cohesive learning experience.

Citations:

- Required components:
  - Objective statement
  - Concept set with explicit relationships
  - Assessment sequence
  - Practical Task
- Completion requires:
  - Certification for all included Concepts
  - Task completion validation
- Must maintain referential integrity with Concepts
- Must preserve Relationship constraints

**Learning Path Definition:** A Learning Path is a directed acyclic graph of Concepts ordered by their Dependencies, terminating at a target Concept.

Citations:

- Construction via topological sort of Concept subgraph
- Must respect both Knowledge and Definitional Dependencies
- Must maintain consistency with Relationship Hierarchy
- Path validity must be computably verifiable

**Certification Definition:** A Certification is a verified record of achieved Understanding for a specific Concept.

Citations:

- Automatically awarded upon Summative Assessment completion
- Must maintain cryptographic proof of completion
- Must track Understanding level achieved
- Must preserve assessment evidence
- Must be verifiable against source Knowledge

## Dependencies

**Knowledge Dependency Definition:** A Knowledge Dependency represents the requirement for “Practical” or “Intermediate” Understanding of a prerequisite Concept.

Citations:

- Created from Knowledge Mentions
- Must respect Relationship Hierarchy
- Must maintain transitive closure
- Must be computably verifiable
- Cannot create circular dependencies

**Definitional Dependency Definition:** A Definitional Dependency represents the requirement for “Basic” or “Functional” Understanding of a prerequisite Concept.

Citations:

- Created from Definition Mentions
- Stronger constraint than Knowledge Dependency
- Must respect Relationship Hierarchy
- Must maintain transitive closure
- Must be computably verifiable

**Mention Definition:** A Mention is an occurrence of a Concept’s lexeme that creates an explicit Dependency.

Citations:

- Types determined by context:
  - Knowledge context creates Knowledge Dependency
  - Definition context creates Definitional Dependency
- Must be automatically detected and validated
- Must maintain referential integrity
- Must be updated when Knowledge changes
- Must respect Relationship constraints

## Content Organization

**Article Definition:** An Article is a narrative presentation of a Concept’s Knowledge optimized for learning engagement.

Citations:

- Must preserve all Knowledge relationships
- Must maintain consistent structure
- Must support multimedia integration
- Must track engagement metrics
- Must validate against source Knowledge

**Teaser Definition:** A Teaser is a condensed presentation of Concept Knowledge designed to rapidly establish basic Understanding.

Citations:

- Must target “Functional” or “Basic” Understanding level
- Must maintain accuracy with full Knowledge
- Must be automatically generated
- Must be validated against source material
- Must respect Dependencies

**Corpus Definition:** A Corpus is the organized collection of all Assets and their derived Knowledge within an Organization.

Citations:

- Organization-specific containment
- Must maintain Asset provenance
- Must track all derived Knowledge
- Must enforce Relationship constraints
- Must maintain referential integrity

**Norm Definition:** A Norm is a universally recognized lexeme that is automatically provisioned across Organizations as an empty Concept.

Citations:

- Created without associated Knowledge
- Can be populated differently per Organization
- Optional removal supported
- Must maintain lexeme consistency
- Must respect Relationship constraints