

# Augmented Knowledge Base Creation and Curation using Large Language Models: A Proof-of-Concept Implementation

Joshua Cook

December 8, 2024

**Abstract:** This paper demonstrates that while technical challenges exist in scaling LLM-based knowledge management systems, these challenges are primarily engineering problems rather than fundamental limitations. Through implementation of the Chelle Knowledge Model (CKM), we show that automated ontology creation can transform organizational knowledge management by enabling rapid, iterative knowledge structuring and deployment. Our proof-of-concept implementation validates the feasibility of zero-shot information extraction techniques and context-aware knowledge processing in organizational settings.

***Note:** The length of this paper is significantly longer than the 10-12 page limit, however, the bulk of this is the appendices and the Chelle Knowledge Model. The actual paper is about 12 pages.*

## I. Introduction

Organizations face substantial challenges in managing institutional knowledge. Studies show that employees spend 19% of their time searching for and gathering information (Chui et al., 2012), while knowledge silos impair organizational decision-making processes (Deloitte, 2020). The volume of organizational data continues to grow exponentially (Taylor, 2023; Gartner, 2023), with information scattered across departments, systems, versions, and documents (Deloitte, 2020). These inefficiencies represent both operational and productivity costs. Organizations must maintain knowledge currency in rapidly evolving fields (Tamayo et al., 2023). They must address departmental silos that prevent effective knowledge sharing (Cross et al., 2004). They must ensure consistent knowledge access across their workforce while managing domain-specific vocabulary and concepts (Otero et al., 2015). Traditional knowledge management approaches, based on manual curation and hierarchical structures, have limitations in addressing these needs (AIIM, 2023). Research indicates that knowledge management effective-

ness decreases as organizational complexity increases, with larger enterprises reporting lower satisfaction rates with their systems (Heisig et al., 2016).

These organizational challenges motivated our research into potential solutions, focusing on the intersection of Large Language Models (LLMs) and ontology management. While LLMs present opportunities for knowledge extraction and ontology development (Babaei Giglou et al., 2023), existing approaches lacked frameworks for organizational deployment. Our research revealed three fundamental knowledge processing operations essential for effective organizational knowledge management: 1) **entity extraction**, 2) **relationship extraction**, and 3) **ontological mapping**. Implementing these operations required addressing several technical challenges, including context limitations, potential hallucination issues (Bikeyev, 2023; Wang et al., 2023), worker load balancing, API management, and system scalability. These operations and their associated challenges emerged as the core considerations for automated knowledge structuring at scale.

## Problem Statement

The objective of this work can be formally stated as follows:

**Given:** A corpus of organizational documents  $\Omega = \{d_1, \dots, d_n\}$

**Goal:** Extract and formalize the latent knowledge  $\mathcal{K}_\ell(\Omega)$  into a structured ontology  $\mathcal{O}$

$$f : \Omega \xrightarrow{\text{extraction}} \mathcal{K}_\ell(\Omega) \xrightarrow{\text{formalization}} \mathcal{O} \quad (1)$$

where:

- $\mathcal{K}_\ell(\Omega)$  represents the latent knowledge embedded within  $\Omega$
- $\mathcal{O} = (C, R, A)$  is a formal ontology consisting of:
  - $C$ : Set of concepts
  - $R$ : Set of relations between concepts
  - $A$ : Set of axioms governing the ontology

## Contributions

Toward the solution of this problem we developed the **Chelle Knowledge Model (CKM)**, a formal description of knowledge extracted from a corpus. It is accompanied by a proof-of-concept application [link]. The CKM specifically addresses the challenges of knowledge currency, accessibility, and scalability through a structured approach to knowledge processing and organization. The CKM serves as a fundamental framework defining our application’s approach to knowledge management. Ontologies have historically been the province of

research institutions. The CKM represents an opinionated approach to making ontologies more accessible and practical for everyday organizational knowledge management. With its formalism, we can get down to the task of automating the creation of everyday ontologies.

The model is built around the three fundamental operations that work in sequence:

- **Entity extraction** from organizational documents
- **Relationship extraction** between identified entities
- **Ontological mapping** to build a comprehensive knowledge structure

This framework builds on recent advances in zero-shot learning for knowledge base population (Caufield et al., 2024) and entity mapping (Matentzoglou et al., 2023).

The research and implementation demonstrate that while technical challenges exist in scaling LLM-based knowledge management systems, these challenges are primarily engineering problems rather than fundamental limitations of the approach. The successful implementation of core functionality supports the viability of using LLMs to transform organizational knowledge management through automated extraction and organization processes. The remainder of this paper explores this transformation in detail: Section II presents the Chelle Knowledge Model (CKM), including its design principles and core components. Section III describes the knowledge processing pipeline implementation and technical architecture. Section IV discusses implementation progress, challenges, and implications for future work. Finally, Section V concludes with an analysis of the transformative potential of automated ontology creation in organizational knowledge management.

## II. The Chelle Knowledge Model (CKM)

### Overview and Motivation

Traditional knowledge bases often fail to effectively manage and preserve context, leading to significant inefficiencies in knowledge extraction and application. According to AIIM (2023), 74% of organizational content systems remain disconnected from other business systems, creating persistent knowledge silos. The rapid growth of scientific and technical literature further compounds these challenges, outpacing traditional manual curation efforts (Caufield et al., 2024).

The Chelle Knowledge Model addresses these limitations through a structured approach to knowledge organization and processing. The model recognizes that knowledge meaning and relationships evolve with context, necessitating a formal framework for preserving semantic integrity while supporting natural knowledge evolution.

## Design Principles

The Chelle Knowledge Model (CKM) is founded on five fundamental design principles:

1. The **Semantic Integrity and Context Preservation** principle ensures all knowledge transformations maintain explicit context tracking, version control for knowledge elements, provenance tracking from raw assets through refined knowledge, and preservation of semantic relationships across organizational contexts (Cross et al., 2004, Otero et al., 2015, Matentzoglou et al., 2023).
2. The **Mathematical Formalism** principle implements a complete lattice structure for relationship hierarchies, formal relationship classification with precise mathematical semantics, computationally verifiable constraints and validations, and rigorous dependency management through directed acyclic graphs (Heisig et al., 2016, Wang et al., 2023, Caufield et al., 2024).
3. The **Operational Transparency** principle maintains bidirectional traceability between conceptual and operational elements, explicit validation requirements at every transformation step, complete audit trails for all knowledge operations, and clear separation between raw assets and refined knowledge (AIIM, 2023, Jackson et al., 2021, Jonquet et al., 2024).
4. The **Knowledge Evolution Management** principle provides structured approaches for handling concept relationships over time, formal procedures for relationship validation and strength calculations, systematic mapping validation across knowledge domains, and clear versioning and inheritance rules for knowledge structures (Tamayo et al., 2023, Gartner, 2023, Taylor, 2023).
5. The **Human-in-the-Loop Validation** principle establishes explicit roles for knowledge curation and oversight, authority designation for knowledge validation, systematic quality checks with human verification, and clear certification processes for knowledge understanding (Matthews et al., 2023, Deloitte, 2020, Qian et al., 2025).

## Core Components

The CKM architecture is structured in three interconnected layers, as shown in Figure 1. This diagram represents the complete transformation of organizational knowledge from initial concepts to operational implementation.

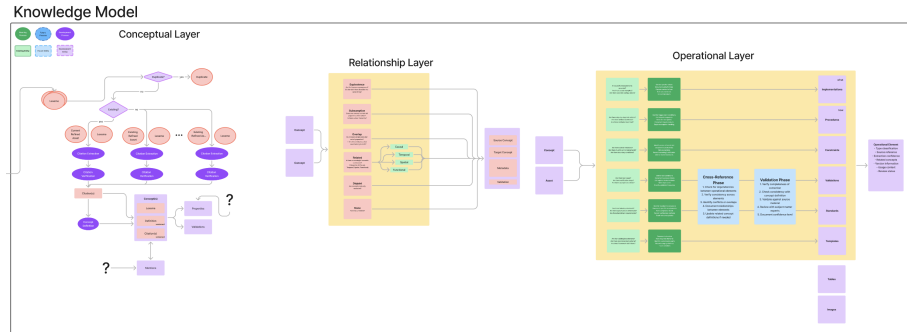


Figure 1: The Chelle Knowledge Model (CKM)

The three layers work in concert to process and structure organizational knowledge:

1. **Conceptual Layer**
  - Begins with lexeme identification and concept formation
  - Establishes concept definitions and validations
  - Manages knowledge progression through refinement stages
  - Forms the foundation for all knowledge representation
2. **Relationship Layer**
  - Implements a formal classification system for concept relationships
  - Processes relationships through defined types (equivalence to none)
  - Maintains semantic consistency across concept connections
  - Ensures relationship validity and strength calculations
3. **Operational Layer**
  - Transforms abstract concepts into concrete implementations
  - Provides templates and tools for knowledge operationalization
  - Implements cross-reference and validation phases
  - Ensures practical applicability of theoretical knowledge

### From Theory to Practice

The CKM's theoretical framework, with its three-layer architecture and formal design principles, necessitates a carefully structured technical implementation. The transition from theoretical model to operational system requires addressing complex requirements that span all layers of the architecture.

**Theoretical Requirements** The implementation of CKM demands technical solutions across three critical areas, each of which intersects with all layers of the model:

1. Document Processing requirements address how the system ingests and processes organizational knowledge:

- Context-aware lexeme<sup>1</sup> identification and extraction
  - Relationship detection between concepts within and across documents
  - Validation of extracted knowledge against CKM rules
  - Preservation of semantic meaning throughout processing
  - Transformation of raw content into operational knowledge
2. Storage Architecture requirements define how the system maintains and manages knowledge:
    - Schema design supporting all three CKM layers simultaneously
    - Version control and change tracking across the entire knowledge hierarchy
    - Efficient indexing for multi-dimensional knowledge retrieval
    - Context preservation across organizational boundaries
    - Support for evolving knowledge structures and relationships
  3. Integration Requirements specify how system components interact:
    - Coordinated LLM processing across conceptual, relationship, and operational layers
    - Distributed worker architecture for scalable processing
    - Human validation interfaces at all levels of knowledge processing
    - Cross-reference and validation mechanisms spanning the entire model

## Proof-of-Concept Implementation

1. **Core Infrastructure**
  - Docker-based containerization enabling modular service deployment
  - NGINX reverse proxy managing request routing and load distribution
  - Redis providing distributed queue management and caching
  - MongoDB offering flexible document storage with versioning
  - Horizontal scaling with 20 API replicas and 4 worker nodes
2. **Processing Pipeline**
  - FastAPI framework implementing RESTful endpoints for knowledge operations
  - Python RQ managing distributed task processing across layers
  - Langfuse providing comprehensive LLM operation monitoring
  - Specialized processor classes handling context-aware operations:
    - Lexeme identification and extraction
    - Relationship detection via structured prompts
    - Validation chains enforcing CKM rules
    - Document versioning and state tracking
3. **Knowledge Management**
  - MongoDB schema supporting multi-layered knowledge representation
  - Metadata enrichment maintaining contextual relationships
  - Immutable document patterns enabling reliable version control
  - Flexible indexing supporting cross-cutting queries
  - State tracking across processing stages

---

<sup>1</sup>A lexeme is defined in CKM as 'a unit of meaning in a language, consisting of a word or group of words.'

#### 4. Integration Framework

- AWS Bedrock integration for Claude API access
- Structured prompt templates managing LLM interactions
- Redis Queue coordinating distributed processing
- Next.js frontend enabling:
  - Human-in-the-loop validation interfaces
  - Interactive knowledge exploration
  - Real-time processing feedback

#### 5. Development Support

- Jupyter Lab providing interactive development environment
- RQ Dashboard offering queue visualization and monitoring
- Hot-reload capability enabling rapid iteration
- Comprehensive logging across all system components

The efficacy of this architecture will be discussed in Section IV.

### III. Implementation of the Knowledge Processing Pipeline

Our implementation of the CKM is structured around the model’s three-layer architecture:

- A **Conceptual Layer** implementation for lexeme identification and concept formation
- A **Relationship Layer** implementation for managing the relationship hierarchy and validation
- An **Operational Layer** implementation for transforming abstract concepts into concrete knowledge entities

#### Architecture Overview

The system implementation translates these architectural components into a processor-based design, with distinct components handling specific aspects of document analysis and knowledge extraction. The core processors are organized in a hierarchical structure:

```
api/processors/assets/  
- process_citations.py  
- process_definitions.py  
- process_images.py  
- process_lexemes.py  
- process_refined.py  
- process_refined_metadata.py  
- process_refined_splitting.py  
- process_tables.py
```

Each processor implements a specific stage in the knowledge extraction pipeline, with processing flows managed through an asynchronous task system.

## Document Processing Pipeline

Appendix I describes a detailed example of document processing flow.

**Document Classification and Metadata Extraction** The system employs domain-specific prompts to analyze incoming documents, classifying them into categories including Technical, Educational/Academic, Commercial/Business, Legal/Compliance, Research/Scientific, Administrative/Operational, and General/Mixed. This classification determines subsequent processing strategies.

**Lexeme Extraction** The system implements both general-purpose and domain-specific lexeme extraction. Domain-specific extractors use carefully crafted prompts targeting particular document types. For example, academic document processing focuses on identifying learning components, pedagogical elements, and domain-specific terminology. These prompts are organized in a structured repository:

```
prompts/assets/lexeme/  
- academic.txt  
- administrative.txt  
- business.txt  
- general.txt  
- legal.txt  
- research.txt  
- technical.txt
```

**Citation Processing** For each identified lexeme, the system extracts supporting citations from the source document. This component currently processes lexemes sequentially, which has revealed significant scalability challenges when handling documents with high lexeme density (>25 lexemes).

**Concept Formation** The system synthesizes lexemes and their supporting citations into structured concept definitions using a JSON-based prompt structure. This ensures consistent concept representation while maintaining traceability between concepts and their supporting evidence.



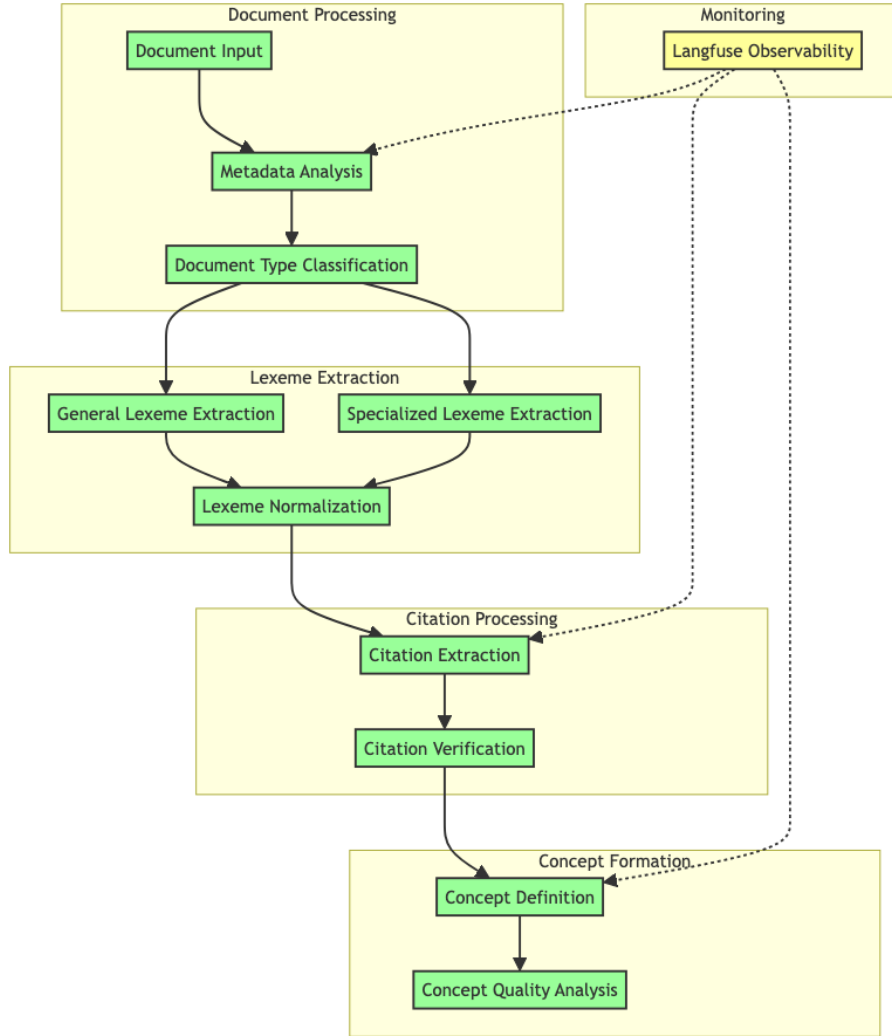


Figure 2: Knowledge Processing Pipeline Architecture

### Technical Challenges and Solutions

Several significant technical challenges emerged during implementation:

1. **Asynchronous Processing:** The system's initial design revealed limitations in handling multiple concurrent lexeme processing tasks. Integration of Langfuse monitoring provided crucial insights into processing bottlenecks and component interaction patterns.
2. **Worker Load Balancing:** Initial attempts to address performance issues

through API server load balancing proved insufficient. Subsequent implementation of worker-level load balancing demonstrated more promising results, though requiring substantial architectural refactoring.

3. **Debug Complexity:** The asynchronous nature of the processing pipeline complicated debugging efforts, particularly for delayed processing functions. This challenge led to the implementation of comprehensive trace logging and monitoring.

## Monitoring and Observability

The integration of Langfuse provided essential observability into the LLM-based processing pipeline. The monitoring system tracks:

- Processing duration for each pipeline stage
- Success/failure rates for processing tasks
- Resource utilization patterns
- Component interaction sequences

This monitoring infrastructure has proved invaluable for both development iteration and system optimization.

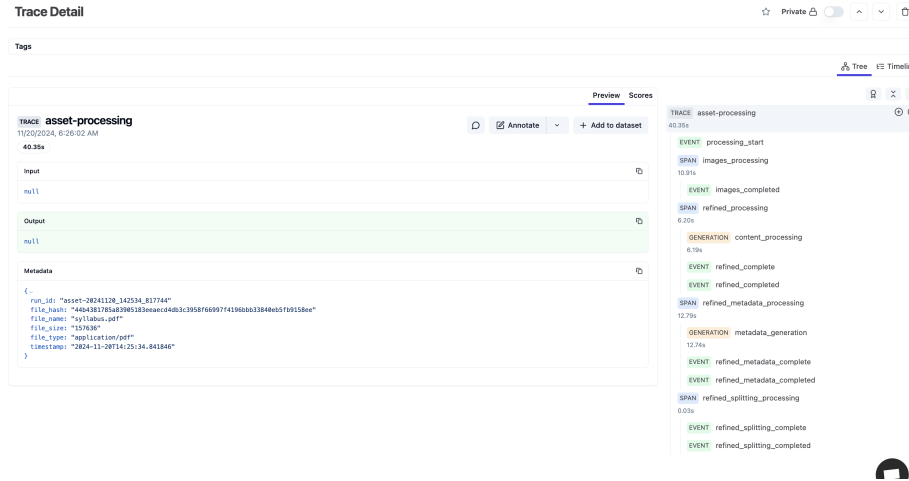


Figure 3: An Example Langfuse Trace

I'll help you merge those sections into a cohesive opening for Section IV. Here's how I would combine them:

## IV. Discussion

The implementation of the knowledge processing pipeline demonstrated both promising results and revealed technical challenges that impact system scalability.

Core functionality was successfully implemented. But limitations emerged. We had issues with sequential processing of documents, especially when they grew to be more complex. It was also difficult to debug code in asynchronous operations i.e. the `rq` delayed job system. In my opinion, these are engineering challenges rather than a conceptual flaws in the approach.

### Implementation Progress and Challenges

When processing rich documents containing numerous concepts (>25 lexemes) we saw performance degradation. This bottleneck led to exploring two distinct solution paths:

1. Initial attempts focused on load balancing the API server to better utilize system resources
2. Subsequently, attention shifted to worker load balancing, which proved to be the correct approach but required substantial code refactoring

### Scope Management Insights

A key learning was the importance of careful scope management in research projects. While the theoretical framework of the CKM was well-developed and remains sound, the implementation would have benefited from a more targeted initial approach focusing on:

1. Small, representative document samples rather than full-scale processing
2. Core functionality demonstration over scalability optimization
3. Complete implementation of basic features before addressing performance challenges

Starting with minimal viable examples can provide faster validation of core concepts while avoiding premature optimization challenges.

In terms of completed components, the following table summarizes the status of each component:

Component	Status
Document classification and metadata extraction	Implemented
Basic lexeme extraction	Implemented
Core document processing pipeline	Implemented
API server and worker architecture	Implemented
MongoDB storage implementation	Implemented
Redis queue management	Implemented
Human-in-the-loop validation	Theoretical
Relationship extraction	Theoretical
Citation verification	Theoretical
Concept formation	Theoretical
Assessment framework	Theoretical
Complete operational layer	Theoretical

Component	Status
Learning structure implementation	Theoretical
Mentions system	Theoretical

Table 2: Implementation Status of System Components

### Preliminary Results

Initial results from concept extraction and definition were promising, based on ad hoc analysis of system logs. The LLM-based approach demonstrated effectiveness in identifying and defining concepts from input documents. It should be noted that due to development focusing on core functionality, the system’s output capture mechanisms were not implemented in a way that would support rigorous research validation and presentation.

### Time Constraints and Future Work

The primary limiting factor in this implementation was time rather than fundamental issues with the system architecture or approach. The underlying architectural decisions appear sound based on the implemented components’ performance. Development of the system will continue, with particular focus on:

1. Completing the relationship extraction functionality
2. Implementing robust output capture for proper evaluation
3. Leveraging the refactored worker load balancing system for improved scalability
4. Expanding the use of Langfuse for monitoring and optimization of LLM interactions

Manual experiments with knowledge model construction and LLM interaction have provided encouraging indicators of the system’s potential effectiveness, though formal validation remains to be completed. The successful implementation of core functionality, despite scalability challenges, supports the viability of the overall architectural approach. Future work will build upon these foundations to realize the full potential of the system.

## V. Conclusion

### Transformative Potential of Automated Ontology Creation

The most significant insight from this research is not merely the automation of ontology creation through LLMs, but rather how this automation fundamentally transforms the role of ontologies in knowledge management. When ontology creation shifts from a months-long expert endeavor to a rapid, iterative process,

it enables ontologies to be deployed for previously impractical use cases - much like how the accessibility of databases or notebooks has enabled their widespread adoption in diverse contexts.

## **Implementation Insights**

While the proof-of-concept implementation faced expected challenges in scaling complex machine learning systems, these challenges proved to be primarily engineering hurdles rather than fundamental limitations. The core functionality demonstrated the viability of the architectural approach, particularly in concept extraction and definition, though full relationship extraction capabilities remain to be implemented.

The development of the Chelle Knowledge Model (CKM) emerged as a foundational framework - akin to a constitutional document - providing guiding principles for knowledge management implementation. This theoretical foundation, combined with consistently promising outputs from experimental work, suggests strong potential for the overall approach.

## **Practical Applications**

The immediate practical application of this work focuses on educational contexts, specifically facilitating connections between students, teachers, and knowledge resources. This narrow but concrete focus provides a clear path for validating the approach while avoiding overextension of the system's capabilities.

## **Future Work**

Two critical areas require attention for advancing this work:

1. Development of a complete working system implementation that demonstrates the full potential of the theoretical framework
2. Rigorous validation of the system's effectiveness in real-world applications

The successful implementation of core functionality, despite various technical challenges, supports the fundamental viability of using LLMs to transform knowledge management through rapid ontology creation and deployment. While significant work remains, particularly in validation and scaling, the potential to fundamentally change how organizations approach knowledge structuring and management appears promising.

# Appendix

## Appendix A: Example – Technical Document Processing Flow

### Input Document

Title: Best Practices for API Security

Section 1: Authentication

OAuth2 is the recommended protocol for API authentication. It provides secure delegated access through token-based authorization.

Section 2: Rate Limiting

Implement rate limiting to prevent API abuse. Consider using token bucket algorithms for flexible rate control.

### Processing Pipeline Steps

#### 1. Document Classification & Metadata

```
metadata = {  
    "type": "Technical",  
    "category": "Security",  
    "confidence": 0.95,  
    "source": "internal_documentation",  
    "timestamp": "2024-12-09T10:30:00Z"  
}
```

#### 2. Lexeme Extraction    Identified lexemes:

- “OAuth2”
- “API authentication”
- “token-based authorization”
- “rate limiting”
- “token bucket algorithms”

#### 3. Citation Processing    Example citation extracted:

```
{  
    "lexeme": "OAuth2",  
    "context": "recommended protocol for API authentication",  
    "source_location": "Section 1",  
    "confidence": 0.92  
}
```

#### 4. Concept Formation    Example concept generated:

```
{
  "lexeme": "API authentication",
  "definition": "The process of verifying identity and access rights for API requests",
  "related_concepts": ["OAuth2", "token-based authorization"],
  "citations": [
    { "source": "Section 1", "text": "OAuth2 is the recommended protocol..." }
  ],
  "confidence": 0.89
}
```

## Appendix B: References

- Association for Intelligent Information Management (AIIM). (2023, April 20). *2023 State of the Intelligent Information Management Industry*. AIIM. <https://www.aiim.org/industrywatch2023>
- Babaei Giglou, H., D’Souza, J., & Auer, S. (2023, October). LLMs4OL: Large language models for ontology learning. In *International Semantic Web Conference* (pp. 408-427). Cham: Springer Nature Switzerland.
- Bikeyev, A. (2023). Synthetic ontologies: A hypothesis. Available at SSRN 4373537.
- Caufield, J. H., Hegde, H., Emonet, V., Harris, N. L., Joachimiak, M. P., Matentzoglou, N., ... & Mungall, C. J. (2024). Structured prompt interrogation and recursive extraction of semantics (SPIRES): A method for populating knowledge bases using zero-shot learning. *Bioinformatics*, 40(3), btae104.
- Chui, M., Manyika, J., Bughin, J., Dobbs, R., & Roxburgh, C. (2012). *The social economy: Unlocking value and productivity through social technologies*.
- Cross, R., & Sproull, L. (2004). More than an answer: Information relationships for actionable knowledge. *Organization science*, 15(4), 446-462.
- Deloitte. (2020). *2020 Deloitte Global Human Capital Trends: The social enterprise at work: Paradox as a path forward*. Deloitte Development LLC. <https://www2.deloitte.com/content/dam/Deloitte/us/Documents/human-capital/us-2020-deloitte-global-human-capital-trends.pdf>
- Gartner, Inc. (2023, May 10). *Gartner survey reveals 47% of digital workers struggle to find the information needed to effectively perform their jobs*. <https://www.gartner.com/en/newsroom/press-releases/2023-05-10-gartner-survey-reveals-47-percent-of-digital-workers-struggle-to-find-the-information-needed-to-effectively-perform-their-jobs>
- Heisig, P., Suraj, O. A., Kianto, A., Kemboi, C., Perez Arrau, G., & Fathi Easa, N. (2016). Knowledge management and business performance: global experts’ views on future research needs. *Journal of Knowledge Management*, 20(6), 1169-1198.
- Jackson, R., Matentzoglou, N., Overton, J. A., Vita, R., Balhoff, J. P., Buttigieg, P. L., ... & Peters, B. (2021). OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies. *Database*, 2021, baab069.
- Jonquet, C., & Grau, N. (2024). M4. 4-Review of Semantic Artefact Catalogues and guidelines for serving FAIR semantic artefacts in EOSC.
- Matentzoglou, N., Caufield, J. H., Hegde, H. B., Reese, J. T., Moxon, S., Kim, H., ... & Mungall, C. J. (2023). Mappergpt: Large language models for linking and mapping entities. *arXiv preprint arXiv:2310.03666*.
- Matthews, J., Love, P. E. D., Porter, S., & Fang, W. (2023). Curating a domain ontology for rework in construction: challenges and learnings from practice. *Production Planning & Control*.



<https://doi.org/10.1080/09537287.2023.2223566>

- Otero-Cerdeira, L., Rodríguez-Martínez, F. J., & Gómez-Rodríguez, A. (2015). Ontology matching: A literature review. *Expert Systems with Applications*, 42(2), 949-971.
- Qiang, Z., Wang, W., & Taylor, K. (2025). Agent-OM: Leveraging LLM Agents for Ontology Matching. *Proceedings of the VLDB Endowment*, 18(1). <https://doi.org/10.48550/arXiv.2312.00326>
- Tamayo, J., Doumi, L., Goel, S., Kovács-Ondrejko, O., & Sadun, R. (2023). Reskilling in the age of AI: Five new paradigms for leaders—and employees. *Harvard Business Review*, 101(5), 86–95. <https://hbr.org/2023/09/reskilling-in-the-age-of-ai>
- Taylor, P. (2023, November 16). *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (in zettabytes)*. Statista. <https://www.statista.com/statistics/871513/worldwide-data-created/>
- Wang, S., Sun, X., Li, X., Ouyang, R., Wu, F., Zhang, T., Li, J., & Wang, G. (2023). GPT-NER: Named Entity Recognition via Large Language Models. arXiv preprint arXiv:2301.13294. <https://arxiv.org/abs/2301.13294>
- Wei, X., Cui, X., Cheng, N., Wang, X., Zhang, X., Huang, S., Xie, P., Xu, J., Chen, Y.,
- Zhang, M., Jiang, Y., & Han, W. (2024). ChatIE: Zero-Shot Information Extraction via Chatting with ChatGPT. Retrieved from <https://arxiv.org/abs/2302.10205>

## Appendix C: Chelle Knowledge Model

### Purpose

Chelle manages an organization’s information via construction of a knowledge model (KM) for that organization.

A KM 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*.

**Raw Asset** Definition: A **Raw Asset** is an unprocessed unit of information from an Integration or direct upload that has not yet undergone refinement or classification.

Citations:

- Must have a unique identifier
- Must maintain provenance data (source, timestamp, uploader)
- Must track original format and encoding
- Can be text, image, video, audio, or structured data
- Cannot be modified after ingestion
- Must be versioned

**Asset Metadata** Definition: **Asset Metadata** is the collection of properties that describe and classify an Asset's characteristics, provenance, and processing state.

Citations:

- Must include content type identification

- Must track processing history
- Must maintain provenance chain
- Must include version information
- Must track modification timestamps
- Must reference source Asset

**Refined Asset** Definition: A **Refined Asset** is a processed version of a Raw Asset that has been normalized, classified, and enriched with Asset Metadata.

Citations:

- Must maintain reference to source Raw Asset
- Must track all processing steps
- Must include confidence scores for processing
- Must be re-generatable from Raw Asset
- Must maintain version lineage
- Must track refinement timestamp

**Markdown File** Definition: A **Markdown File** is a Refined Asset containing structured text with semantic formatting that follows the Markdown specification.

Citations:

- Must preserve original Markdown syntax
- Must identify document structure:
  - Headers and hierarchy
  - Lists and nesting
  - Code blocks and language hints
  - Block quotes
  - Tables
  - Links and references
- Must track inline formatting:
  - Emphasis (bold, italic)
  - Code spans
  - Links
- Must maintain internal reference resolution
- Must include parsing confidence scores
- Must preserve raw source mapping
- Must track frontmatter metadata if present

**Table** Definition: A **Table** is a structured Refined Asset containing data organized in rows and columns with explicit relationships between elements.

Citations:

- Must have defined column schema
- Must maintain cell data types
- Must preserve structural relationships

- Must track header information
- Must maintain source cell positions
- Must include parsing confidence scores

**Image** Definition: An **Image** is a visual Refined Asset that has been processed for content extraction and analysis.

Citations:

- Must maintain original resolution
- Must track visual element detection
- Must preserve text extraction results
- Must include layout analysis
- Must maintain spatial relationships
- Must track processing confidence scores

**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 primitive term
  - Previously defined in the glossary
  - 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:  $\text{strength along any path} \leq \text{minimum strength of any edge}$
- Resolution Rule:  $\text{actual strength} = \text{maximum strength across all paths}$

## 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 4. Resolve to specific Concept 5. Handle ambiguity 6. Validate consistency
3. Maintenance 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