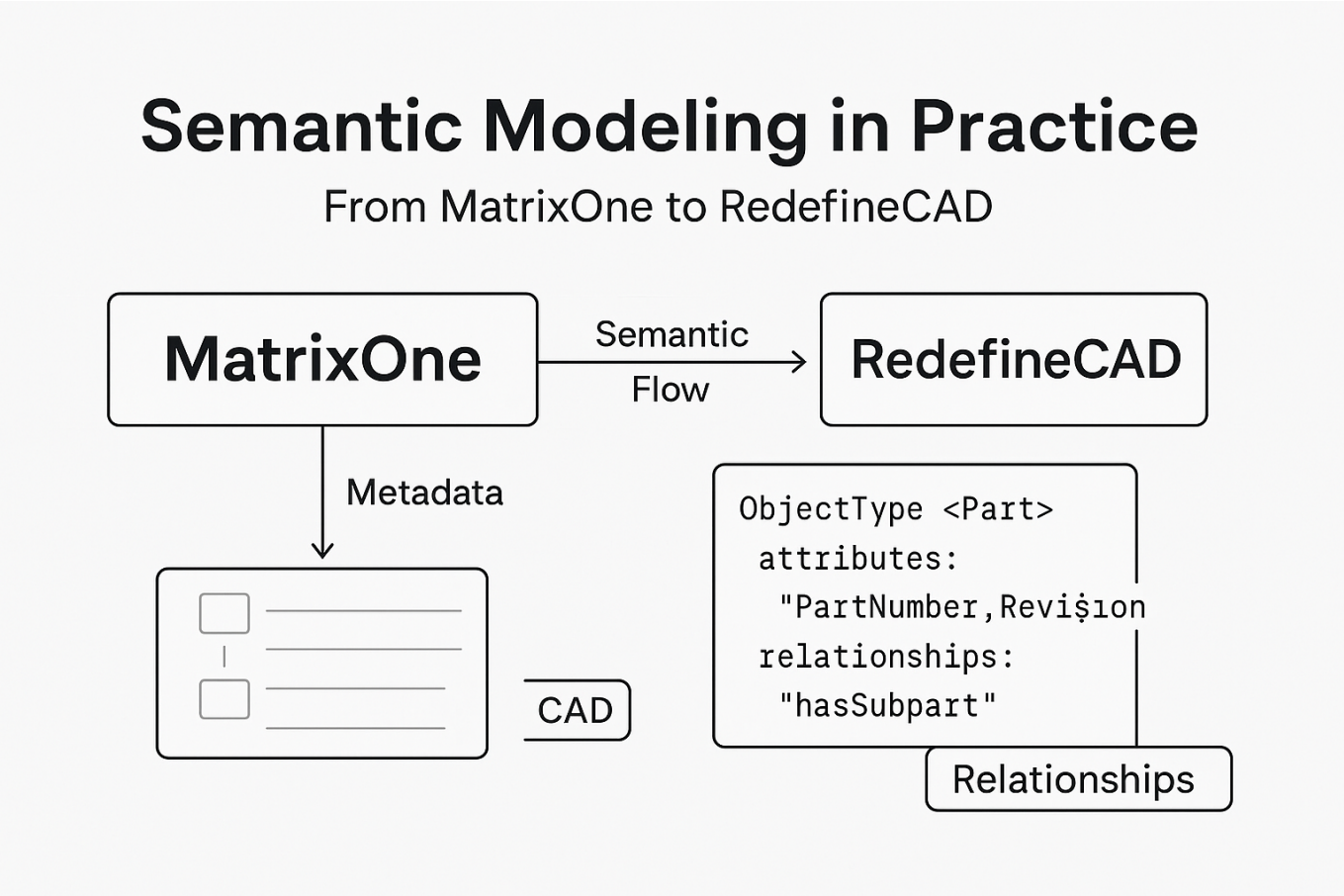
MMSA: Modular Metadata Structure Architecture



Author: Beiji Ma
Created: Summer 2025, Höör, Sweden
Version: v2.0 (Rebuild Edition)

> This white paper is a working draft intended for internal preview. For citation or distribution, please include author attribution.

Abstract

MMSA (Modular Metadata Structure Architecture) introduces a lightweight, composable modeling framework for structured metadata. It addresses challenges found in traditional PLM/PDM systems by emphasizing modularity, lifecycle binding, and semantic traceability. This white paper outlines the rationale, structure, use cases, and governance model of MMSA, intended for architects, developers, and digital transformation strategists.

Overview

MMSA (Modular Metadata Structure Architecture) is a modeling methodology designed for representing, managing, and evolving structured metadata in modular systems. It was conceived to address the limitations of rigid PLM/PDM systems, enabling expressive, lifecycle-aware, and semantically robust metadata modeling.

Table of Contents

Chapter 1: Introduction

MMSA is a framework born out of necessity. As traditional PLM systems fail to adapt to the needs of modern engineering—flexibility, traceability, semantic coherence—the idea of a modular, metadata-first architecture emerged.

MMSA seeks to generalize how structured metadata is defined, composed, versioned, and queried. Whether you're describing a bill of materials (BOM), a digital twin configuration, or a document management lifecycle, MMSA offers a composable and verifiable foundation.

Chapter 2: Motivation & Background

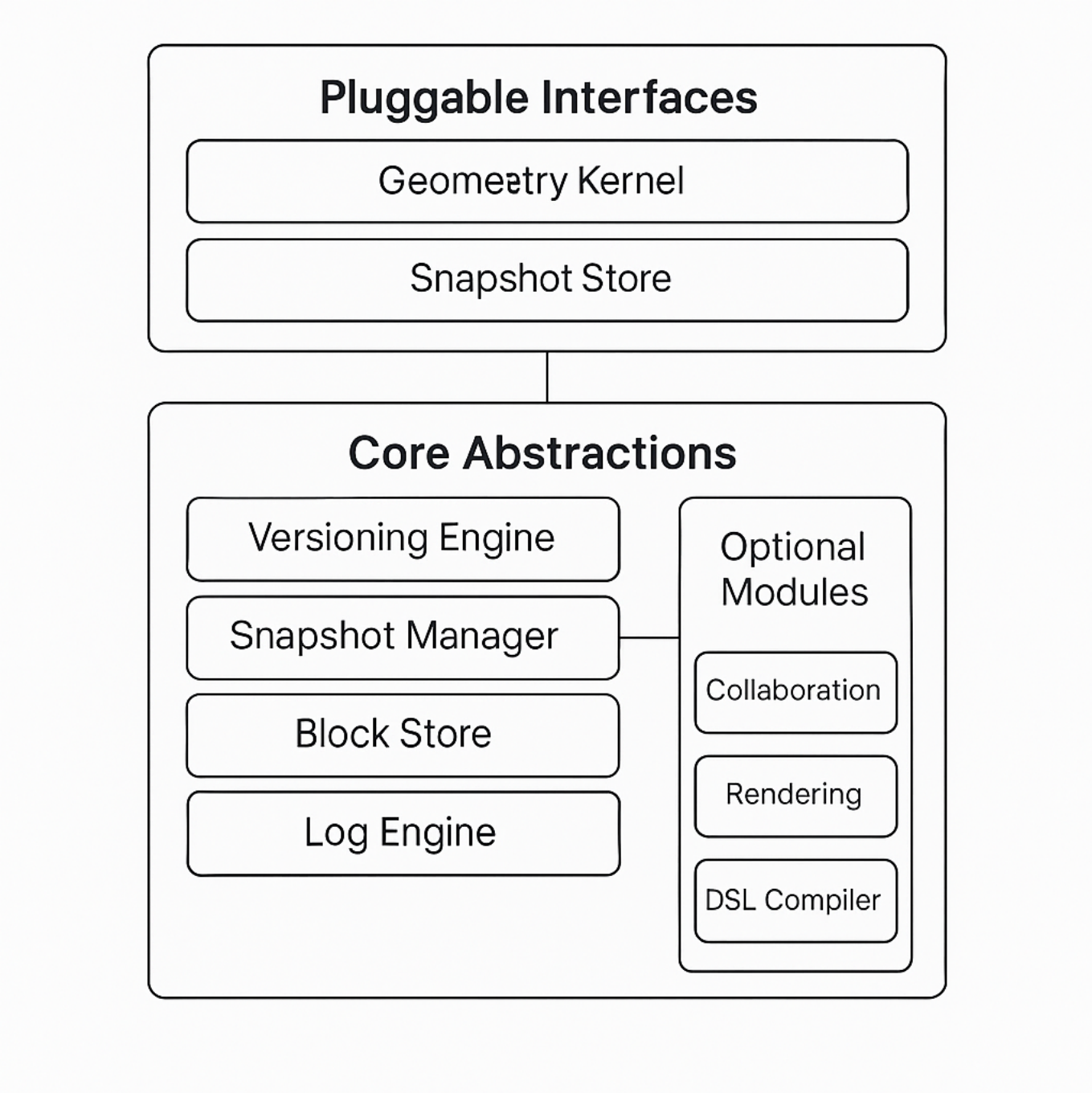
Modern enterprises struggle with:

- Inflexible metadata schemas
- Poor lifecycle modeling
- Siloed applications unable to communicate structure
- Lack of versioned semantics

Legacy PLM systems embed structure within UI logic, or force-fit metadata into rigid object models. MMSA breaks this pattern, offering a DSL-backed metadata representation with lifecycle and snapshot fidelity.

The architecture is influenced by years of frustration across CAD/PLM implementations, where structure had no standalone identity. MMSA inverts the model: structure comes first, and apps follow.

Chapter 3: Core Principles

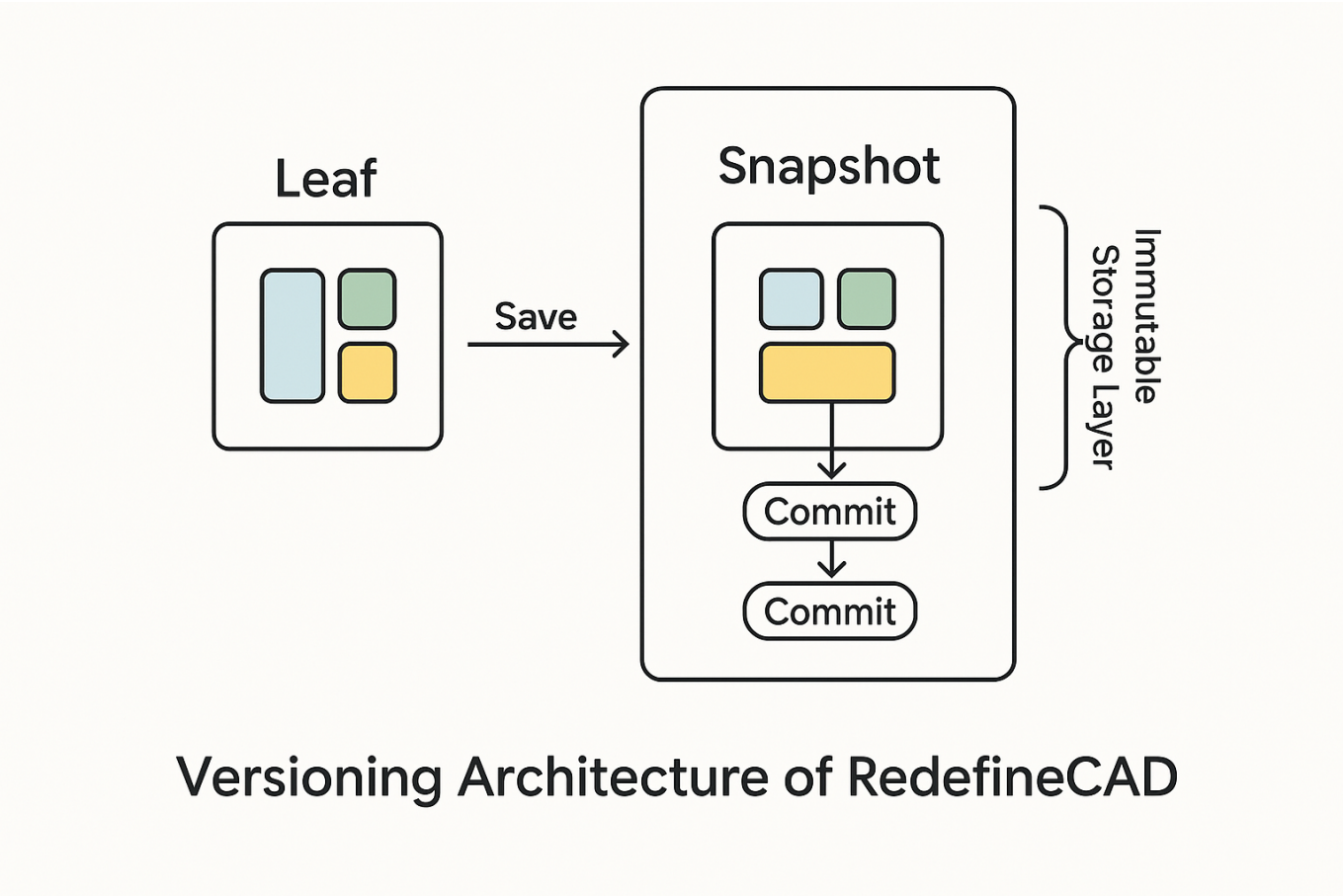


MMSA is grounded in the following principles:

- **Modularity**: Each unit of metadata is independently definable, testable, and versionable.
- **Composability**: Structures are formed by composing smaller blocks. No hidden dependencies.
- **Lifecycle Awareness**: Each structure is bound to state models. You can ask: "what does this look like in `Released` stage?"
- **Semantic Anchoring**: Every structural edge has meaning, whether it's `uses`, `contains`, or `overrides`.

These principles allow MMSA to power collaborative editing, semantic diffs, and permission-aware traversal.

Chapter 4: MMSA Model Structure



4.1 Modules

Modules define bounded metadata domains. Each module has:

- One or more Metadata Types
- Lifecycle model(s)
- Constraints and interpretation rules

Modules can import or extend other modules.

4.2 Metadata Units

Units represent things like `Part`, `Assembly`, `View`, `Document`, or custom definitions. Each unit has:

- Unique identifier (UUID or Part Number)
- Set of attributes
- Optional geometry or reference

4.3 Structural Relations

Metadata units may relate via:

- **Composition** (`has-children`)
- **Dependency** (`uses`, `mounts`)
- **Version Derivation** (`derived-from`, `replaces`)

Each edge may be typed and carry metadata itself (e.g., quantity).

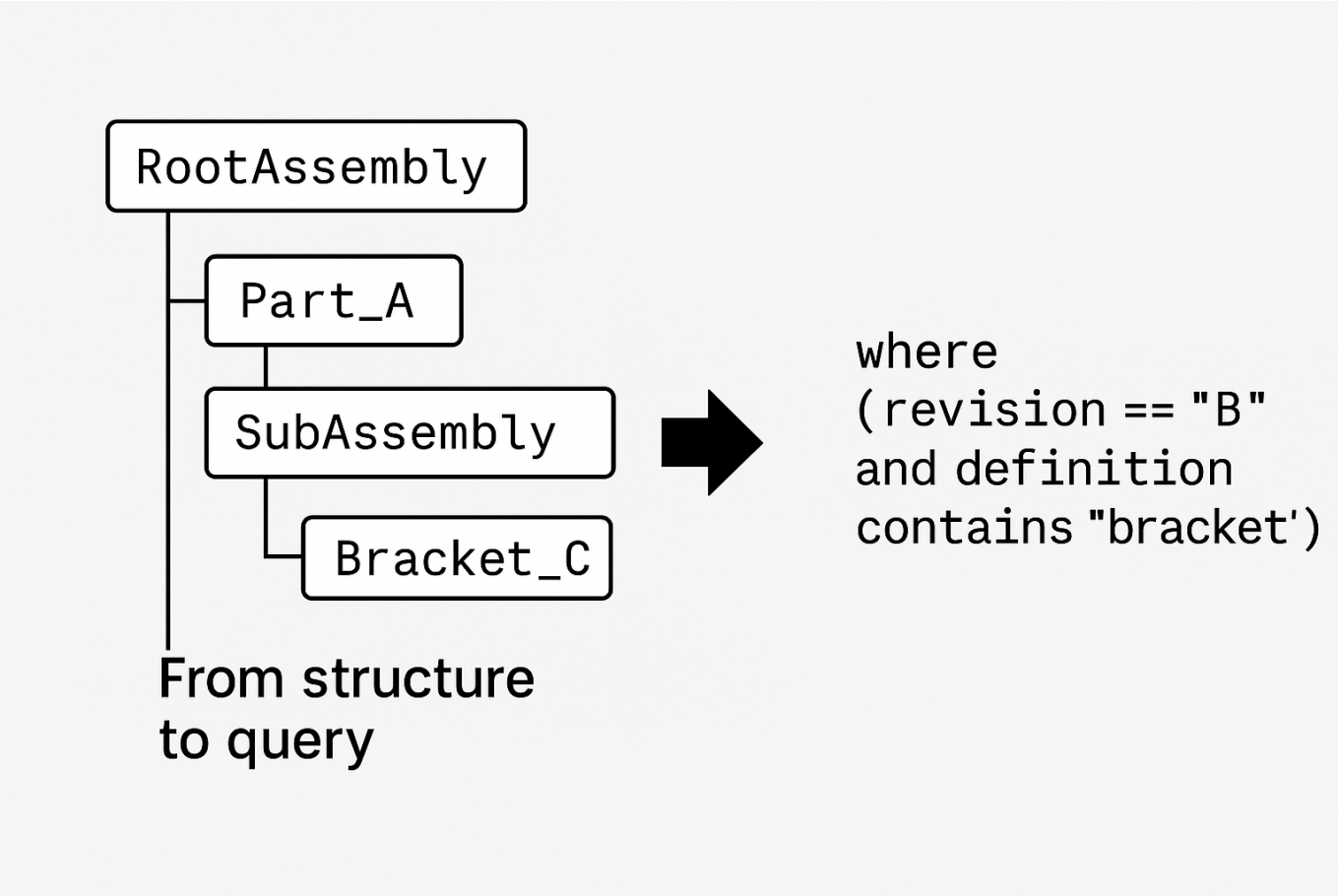
4.4 Lifecycle Bindings

A structure is bound to one or more lifecycles:

- Draft → Review → Release → Obsolete

Snapshots are taken at boundaries. You can query across lifecycles.

Chapter 5: Semantic Layers & Interpretation



MMSA introduces semantic layers to separate concerns and enable clear interpretation at different abstraction levels. These layers include:

- **Data Layer**: Raw metadata attributes and relationships.
- **Schema Layer**: Definitions of metadata types, their fields, constraints, and structural rules.
- **Lifecycle Layer**: Temporal or state-based transitions and constraints.
- **Query Layer**: Filtering, projection, and semantic composition using DSL expressions.
- **View Layer**: Rendered representations (UI, graphs, documents) that depend on metadata interpretation.

Each layer can evolve independently, and the system supports re-interpretation when moving from one layer to another.

Chapter 6: Application Scenarios

6.1 PLM BOM Systems

MMSA redefines how bills of materials (BOMs) are represented:

- Structures are natively composable.
- Each node carries its own metadata and history.
- Snapshots allow precise version tracing.

Supports cross-product variant modeling, assembly change detection, and structural diffing.

6.2 Digital Twins

In IoT or simulation-heavy environments:

- MMSA provides stateful modeling of component evolution.
- DSL allows querying sensor linkages, historical states, or sub-system lineage.
- Semantic diff helps compare twin states over time.

6.3 Multi-view Model Integration

Combining different model representations (3D geometry, requirements, manufacturing processes):

- MMSA supports multi-view alignment.
- View-to-view traceability is enforced semantically.
- Enables traceable relationships between design intent and implementation.

Chapter 7: DSL & Query Layer Integration

MMSA ships with a domain-specific query language to express:

- Structure traversal
- Attribute filtering
- Lifecycle filtering

Sample DSL Query:

```
where (revision == "B" and definition contains "bracket")
```

Execution Semantics:

- Query paths are compiled into a logical plan.
- Filters are matched against metadata and structure.
- Lifecycles determine snapshot scope.

The DSL engine can support indexes, precompiled paths, and secure filtering across boundaries.

Chapter 8: MMSA vs Traditional Architectures

Feature	Traditional PLM	MMSA
Metadata Schema	Hardcoded, fixed	Modular, extensible
Structure Model	Embedded in UI	Decoupled, DSL-driven

Feature	Traditional PLM	MMSA
Versioning	File-based, or ad hoc	Snapshot DAGs
Query	Limited or SQL-based	DSL with structural semantics
Lifecycle	Part of workflow	First-class binding

MMSA promotes structure-first thinking. It empowers developers, architects, and analysts to operate on structure without relying on application code.

Chapter 9: Evolution & Governance

MMSA is designed to evolve safely:

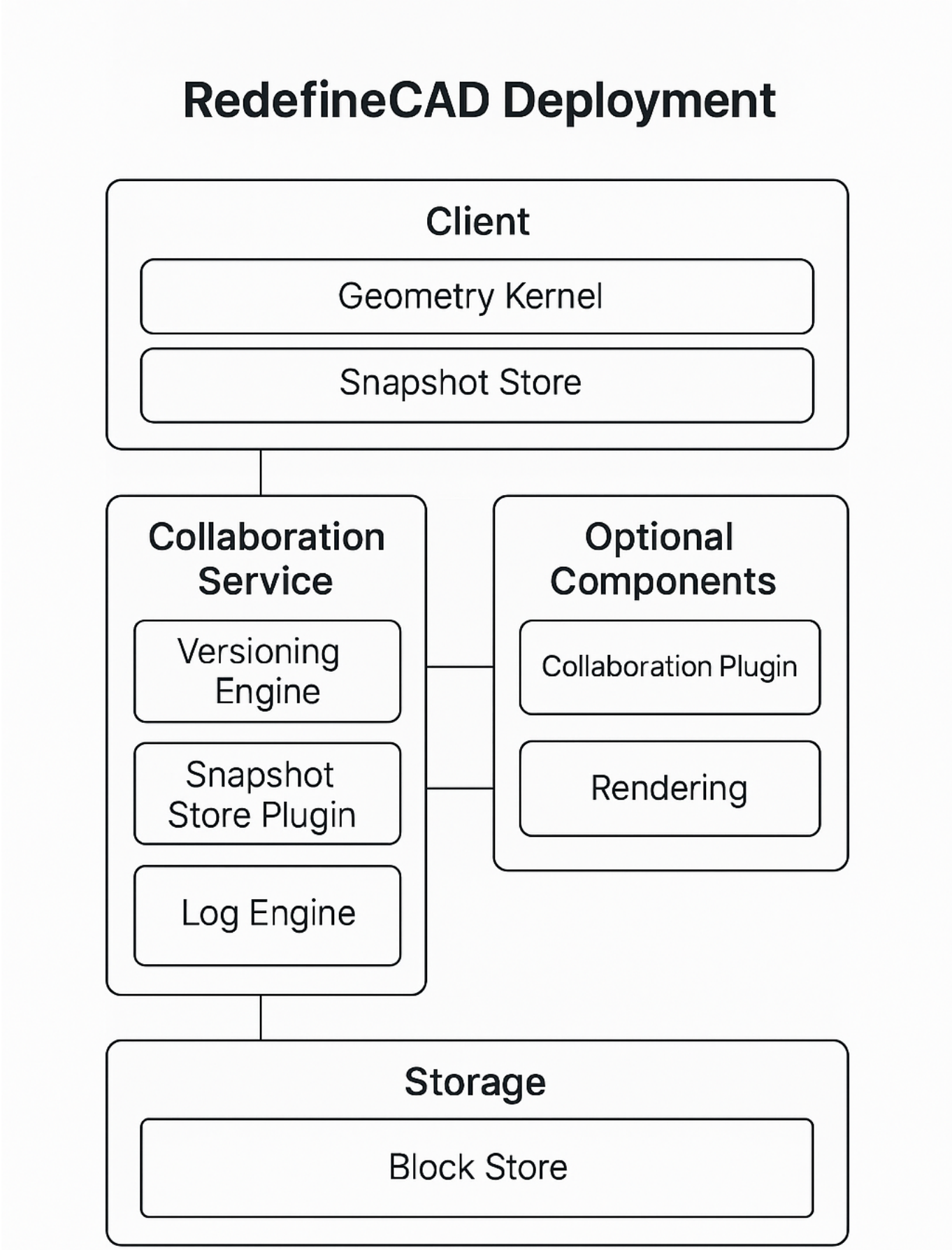
- **Schema Versioning**: Modules can declare versions; migration rules can be defined.
- **Structural Diffs**: Compare two snapshots to detect changes.
- **Governance Hooks**: Pre-save, post-publish rules and approvals.
- **Audit Trail**: Lifecycle events, DSL queries, and mutation histories are traceable.

Governance Roles:

- **Author**: Defines modules and metadata.
- **Reviewer**: Validates lifecycles and usage.
- **Integrator**: Merges and promotes to higher stages.

Future versions may incorporate policy engines, semantic validation, and provenance tracking.

Chapter 10: Implementation Guidelines


RedefineCAD Deployment
Client
Geometry Kernel
Snapshot Store
Collaboration Service
Versioning Engine
Snapshot Store Plugin
Log Engine
Optional Components
Collaboration Plugin
Rendering
Storage
Block Store

Recommended Technology Stack

- **Language**: Python, JavaScript/TypeScript (for DSL and metadata engine)
- **Storage**: JSON, SQLite, or document databases (MongoDB, CouchDB)
- **Query Layer**: Custom DSL parser and executor (can be implemented using ANTLR or PEG parser)
- **Snapshot Manager**: Git-style commit DAG

Deployment Suggestions

- Start with CLI tools for validation, diff, and structure check
- Extend to web-based viewers for visualizing metadata structure
- Use GitHub or GitLab-style review for governance integration

Suggested Licenses

- Open source: MIT, Apache 2.0
- Commercial use: Dual licensing optional

Chapter 11: Glossary of Terms

Term	Definition
Metadata Unit	A structured item (Part, Document, View, etc.) with attributes and identity
Module	A group of metadata types and rules packaged together
Structural Relation	A typed edge between metadata units (e.g., `contains`, `uses`)
Snapshot	A state-captured version of a structure at a lifecycle boundary
Lifecycle Binding	Association of a structure to a lifecycle state model
Semantic Diff	Comparison between snapshots based on metadata and structure
DSL	Domain-Specific Language used to express queries over structure and attributes

Chapter 12: Appendices

- 📜 DSL Grammar Reference (See separate doc `dsl_grammar.md`)
- 🔁 Common Lifecycle Templates (`Draft → Review → Release → Obsolete`)
- 📄 JSON Metadata Serialization Example:

```
{
  "uuid": "part-001",
  "type": "Part",
  "attributes": {"revision": "B", "definition": "bracket"},
  "children": []
}
```

- 🎨 Diagram Legend & Color Mapping:
  - Blue = Metadata Unit

- Green = Structural Link
- Red = Lifecycle State

Acknowledgements

This white paper draws inspiration from real-world limitations in existing PLM solutions and decades of domain experience in metadata lifecycle governance. The author extends thanks to all collaborators and early readers across the RedefineCAD project, especially those contributing insights on semantic modeling, modular design, and DSL-based architecture. Indirect references include:

- Early MatrixOne architecture
- Git and Graph-structured version control paradigms
- ENOVIA V6 and collaborative metadata authoring workflows
- Research on semantic diff, DSL runtime systems, and modular PLM schemas

---
