Dieses Dokument ist eine Zweitveröffentlichung (Verlagsversion)

This is a self-archiving document (published version)

Kim Feldhoff, Martin Zinner, Hajo Wiemer et al.

State-Centric Framework for Canonical Representation and Change Analysis of Evolving Documents in Information Systems

Erstveröffentlichung in / First published in:

Applied Sciences. 2026, 16(15). Art-Nr. 7644. MDPI. ISSN: 2076-3417

DOI: https://doi.org/10.3390/app16157644

Diese Version ist verfügbar / This version is available on:

https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-1065050

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung 4.0 International Lizenz.
This work is licensed under a Creative Commons Attribution 4.0 International License.

SLUB
Wir führen Wissen.

TECHNISCHE UNIVERSITÄT DRESDEN

QUCOSA
Quality Content of Saxony

applied sciences

MDPI

Article

State-Centric Framework for Canonical Representation and Change Analysis of Evolving Documents in Information Systems

Kim Feldhoff [1,*], Martin Zinner [1], Hajo Wiemer [1] and Steffen Ihlenfeldt [1,2]

[1] Faculty of Mechanical Science and Engineering, Institute of Mechatronic Engineering (IMD), TUD Dresden University of Technology, 01062 Dresden, Germany
[2] Fraunhofer Institute for Machine Tools and Forming Technology (IWU), 01187 Dresden, Germany
* Correspondence: kim.feldhoff@tu-dresden.de

Featured Application

The proposed state-centric framework supports traceable change handling in evolving structured-document collections through canonical state representation, explicit predecessor linkage, deterministic comparison, and hierarchical localization of structural modifications. A reproducible operational case study demonstrates one executable realization of these principles. The framework is applicable to engineering documentation, maintenance procedures, compliance records, enterprise knowledge repositories, and related information systems. Computational efficiency, scalability, and robustness remain implementation-dependent and require further empirical evaluation.

Abstract

Information systems increasingly operate on document collections that evolve continuously over time, are queried repeatedly, and must support traceable propagation of relevant changes. Despite this dynamic nature, documents are still commonly treated as static artifacts, which limits the explicit modeling of document evolution and may constrain meaningful comparison. This paper proposes a state-centric framework that reinterprets documents as structured and versioned states within an information space. Each document instance is represented as a canonical, hierarchically decomposed state linked to its contextually valid predecessor. On this basis, change is defined as a formally specified transition between comparable states rather than as an implicit difference between external artifacts. The framework integrates three interdependent layers: deterministic representation through canonicalization and hierarchical decomposition, temporal organization through explicit predecessor relations, and comparison mechanisms operating on structured states. A hierarchical digest-based realization is introduced as one computational instantiation of the framework. Its operational realizability is examined through a controlled and reproducible case study based on successive JSON document states, predefined correspondence information, and an executable reference implementation. Within this setting, equivalent document representations produced identical canonical digests, predefined structural changes were correctly detected and localized where valid correspondence information was available, and ambiguous component identities illustrated the limits of localization without persistent identifiers. Together, the analytical assessment and the reproducible operational case study provide complementary evidence for the conceptual coherence and operational realizability of the proposed framework under the stated representation assumptions.

Check for updates

Academic Editor: Juan-Carlos Cano

Received: 16 June 2026
Revised: 27 July 2026
Accepted: 29 July 2026
Published: 1 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Keywords: change-aware information systems; document comparison; document state representation; evolving documents; canonicalization; deterministic comparability; hierarchical hashing; change detection; traceability; reproducible case study

1. Introduction

1.1. Motivation

Modern information systems increasingly depend on document collections that evolve continuously over time [1–3]. In many application domains, documents are no longer passive records but active information objects that are repeatedly updated, queried, compared, redistributed, and interpreted across heterogeneous technical and organizational environments. Representative examples include engineering specifications, operating procedures, maintenance instructions, compliance records, and digital knowledge repositories. Such artifacts are frequently modified due to design iterations, regulatory updates, process optimization, localization, and context-dependent adaptation.

At the same time, digital transformation initiatives require information assets to remain reliable, traceable, and reusable throughout their lifecycle. This requirement is also reflected in research data management through the FAIR principles, which emphasize findability, accessibility, interoperability, and reusability [4]. These principles implicitly presuppose that evolving information can be represented consistently across versions and compared in a meaningful manner.

These requirements extend beyond information management itself. Recent research increasingly emphasizes that reliable engineering decisions depend on transparent, traceable, and reproducible information structures. For example, physics-informed decision frameworks for reclaimed structural components integrate heterogeneous engineering evidence into traceable decision processes whose reproducibility depends on consistent information representation and documentation [5]. Likewise, recent work on collaborative engineering decision-making highlights that complex engineering processes involve multiple stakeholders, evolving information, and iterative evaluation, making structured and auditable information management an essential prerequisite for consistent downstream decision support [6]. Although these studies focus primarily on engineering decision support rather than document representation, they reinforce the broader motivation of the present work: robust decision processes ultimately depend on well-defined representations of the underlying information.

Despite this dynamic reality, documents are still commonly handled as static files or loosely connected records in many systems and workflows [7–9]. As a result, change is often managed indirectly through manual inspection, ad hoc differencing, or repeated traversal of complete document contents. This mismatch between dynamic information processes and static representation models creates several recurring limitations.

First, comparison effort is repeatedly incurred even when no relevant change has occurred. Second, detected differences are often expressed at the level of raw representations rather than meaningful structural units [10,11]. Third, relations between successive document versions frequently remain weakly formalized, which complicates traceability, auditability, and systematic interpretation of change [12,13].

These issues are not merely technical inefficiencies. They indicate a deeper conceptual gap: many systems still lack a coherent model that treats documents as evolving and comparable information states. Without such a model, efficient comparison, robust traceability, and meaningful interpretation remain fragmented and implementation-dependent.

https://doi.org/10.3390/app16157644

1.2. Problem Statement

The central problem addressed in this paper is therefore broader than improving isolated comparison algorithms. It concerns the representational foundations of change-aware information systems:

> *How should documents be represented such that change becomes a well-defined, comparable, and interpretable property of evolving information states?*

Conventional approaches typically compare external representations such as files, records, or serialized data objects. Consequently, equality checks often require repeated traversal of underlying data, with computational effort that scales with document size even when no change exists. In addition, many approaches conflate two fundamentally different tasks: detecting structural differences and interpreting their practical significance [1,14].

The key premise of this study is that well-defined and meaningful change handling cannot be achieved through algorithmic optimization alone. It also requires an appropriate representation model. Documents must be structured in a way that ensures deterministic construction, structural stability, reproducibility, and explicit linkage between successive states. These conditions establish the basis for deterministic comparison and semantically defensible interpretation; computational efficiency remains implementation-dependent.

1.3. Research Question

Against this background, the paper is guided by the following research question:

> *Which representational, structural, and temporal conditions are required for documents to be compared as evolving information states in a way that supports deterministic change detection, localized structural analysis, and meaningful interpretation?*

This formulation makes clear that the focus is not a single comparison algorithm, but the conditions under which comparison becomes conceptually valid and operationally effective. The research perspective is therefore analytical, model-oriented, and systems-focused.

1.4. Conceptual Perspective

To address this problem, the paper adopts a state-centric perspective on documents. Instead of treating documents as static artifacts, each document instance is interpreted as a structured, versioned state embedded in an evolving information space. A state comprises hierarchical components, deterministic internal representations, contextual metadata, and explicit relations to prior states.

This perspective has several implications. First, comparison can operate on stable internal states rather than heterogeneous external formats. Second, comparability becomes explicit, because only states generated under a shared representation model can be meaningfully compared. Third, change can be defined formally as a transition between states. Fourth, the technical detection of differences can be separated from the contextual interpretation of relevance.

This distinction is especially important for practical information systems. Computational mechanisms may determine whether two states differ and where differences occur. However, the significance of these differences depends on domain context, organizational rules, regulatory requirements, and user roles [11,15]. Change-aware systems should therefore be understood as socio-technical systems in which deterministic detection and contextual interpretation interact, but are not identical.

1.5. Relation to Prior Applied Work

This paper is intentionally distinct from prior application-oriented work by the same authors on hierarchical hash-based change detection in manufacturing environments. That

earlier contribution focused on a domain-specific implementation for worker assistance and near-real-time instruction updates. It demonstrated how structured manufacturing instructions can be canonicalized, represented as versioned states, compared through hierarchical cryptographic digests, and used for role-specific dissemination of detected updates in an industrially motivated setting.

In contrast, the present study develops the domain-independent conceptual foundation underlying such approaches. Its contribution does not lie in proposing a new industrial application, a new cryptographic primitive, or a benchmark study of one specific implementation. Instead, it identifies the representational, temporal, and structural conditions under which robust change-detection mechanisms can operate effectively across domains.

The earlier manufacturing study therefore provides historical motivation for the present work rather than its primary empirical evidence. It illustrates that state-centric document comparison can be realized in an application-specific environment and informed the development of the generalized representation framework presented here.

The operational evidence for the present framework is instead provided by the reproducible case study introduced later in this paper. Unlike the previous domain-specific implementation, this case study is designed specifically to examine whether the proposed representation model can be instantiated as an executable workflow under transparent and independently reproducible conditions. The earlier manufacturing application and the present case study therefore serve different purposes: the former motivated the abstraction of the framework, whereas the latter provides reproducible evidence of its operational realizability. Empirical questions concerning implementation performance, scalability, robustness, and cross-domain applicability remain outside the scope of the present contribution.

1.6. Research Objective and Contributions

The objective of this paper is to develop a general framework for representing, linking, and comparing evolving document states in a way that supports deterministic comparability, structural localization of differences, and explicit temporal traceability. The framework is intentionally formulated at an abstract level so that it can inform the design of change-aware information systems across multiple domains.

Methodologically, the study follows a conceptual modeling and design-science perspective. More precisely, the contribution is a conceptual and formal representation framework with design-oriented implications. It is neither a new comparison algorithm nor a new canonicalization, hashing, version control, or differencing method. Instead, it synthesizes established mechanisms within an explicit representation model and formalizes the conditions under which evolving documents can be constructed as comparable states, linked through predecessor relations, structurally localized, and subsequently interpreted according to context.

The non-trivial methodological insight arising from this integration is that deterministic change analysis is not solely a property of the comparison procedure. It also depends on prior representational conditions: canonical state construction, explicit comparability, valid component correspondence, temporal linkage, and a clear separation between structural detection and contextual interpretation. The framework therefore contributes a formalized conceptual synthesis and a set of design conditions for change-aware document systems rather than a new computational primitive or an empirically optimized implementation.

The contributions of this work can be summarized as follows:

1. **State-Centric Document Model.** Documents are reformulated as structured states rather than static artifacts. The model emphasizes hierarchical decomposition, canonical representation, and deterministic comparability.

2. **Temporal Evolution Framework.** Explicit predecessor linkage is introduced to model document evolution within a versioned state space and to define meaningful comparison scope.
3. **Conceptual Change Model.** Change is defined as a transition between comparable states. A clear distinction is established between technical detection and contextual interpretation.
4. **Representation-Centered Conditions for Deterministic and Localized Comparison.** The framework identifies canonicalization, structural decomposition, valid component correspondence, and explicit comparability as representational conditions that can enable deterministic equality assessment and localized structural analysis. Computational efficiency remains dependent on the chosen implementation, document structure, preprocessing strategy, and workload.
5. **Computational Instantiation and Reproducible Operational Case Study.** The framework is instantiated through a hierarchical digest-based realization and evaluated in a controlled and reproducible operational case study. The study demonstrates executable realization under predefined conditions, including deterministic representation, controlled change detection, hierarchical localization, explicit component correspondence, and reproducible evaluation against a predefined ground truth.

Rather than treating comparison solely as an algorithmic problem, the proposed framework identifies deterministic representation as the prerequisite for well-defined comparability and structurally localized change analysis. The framework is examined through two complementary forms of evidence. An analytical assessment establishes the internal consistency of the representation model and its correspondence to the stated design objectives, while a reproducible operational case study demonstrates that these concepts can be instantiated in an executable workflow under controlled and independently reproducible conditions. Broader claims regarding computational efficiency, scalability, robustness, or implementation superiority remain outside the scope of the present contribution.

The remainder of the paper follows this two-layer validation strategy. Section 3 develops the representation framework and its formal constructs. Section 4 provides an analytical assessment of the conceptual model and its correspondence to the stated design objectives. Section 5 complements this assessment with a reproducible operational case study based on a controlled document-evolution scenario and an executable reference implementation. The Sections 6 and 7 then examine the implications, limitations, and future directions arising from these complementary forms of evidence.

1.7. Positioning and Scope

The proposed framework complements established approaches in version control [7,8], database change tracking [1,3], structured document processing [10,16], and hash-based integrity verification [17]. These approaches provide valuable mechanisms for storing, tracking, validating, and comparing evolving data, but they often assume an underlying representation model that remains implicit.

This paper addresses that gap by focusing on the representational foundations that enable such mechanisms to operate consistently. Rather than replacing existing techniques, the framework provides a unifying perspective that clarifies how representation, temporal linkage, and comparison interact.

The contribution of this work is conceptual and model-oriented rather than empirical. Consequently, the objective is not to establish quantitative superiority over existing approaches, but to formulate and analyze representational conditions for deterministic, traceable, and structurally localized comparison of evolving document states. The paper does not provide runtime measurements, memory measurements, scalability curves, statis-

tical analysis, or experimental comparison with baseline implementations. Accordingly, no claim of empirical performance superiority is made.

The hierarchical digest-based realization presented later provides one computational instantiation of the proposed representation framework. Its operational behavior is examined through a controlled and independently reproducible case study that evaluates deterministic representation, controlled change detection, structural localization, hierarchical refinement, and explicit component correspondence against a predefined ground truth. This case study establishes operational realizability rather than implementation superiority. General conclusions regarding runtime efficiency, scalability, robustness, and cross-domain effectiveness remain implementation-dependent and require dedicated empirical benchmark studies using representative workloads and alternative implementations.

Figure 1 summarizes the framework as a progression from representational foundations to context-dependent interpretation. The layered view also reflects the structure of the remainder of the paper.

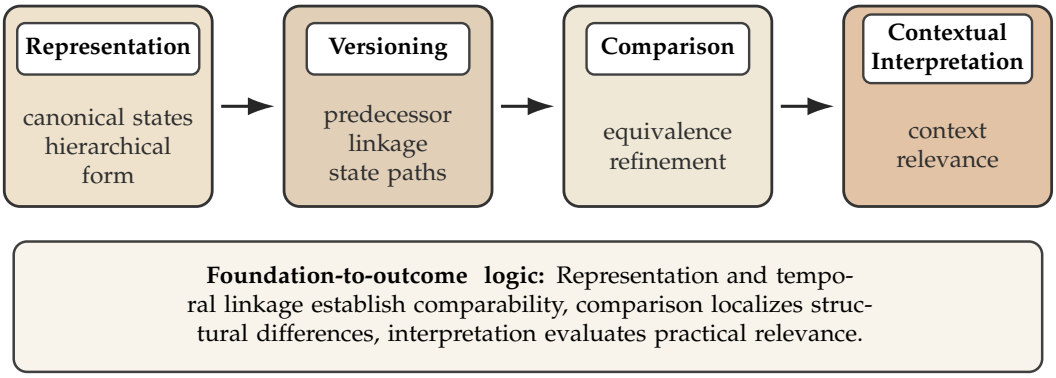


Figure 1. Conceptual architecture and workflow of the proposed framework. External document artifacts are first transformed into canonical hierarchical states. These states are then embedded in explicit predecessor relations, compared through deterministic equivalence and refinement operations, and finally interpreted according to application-specific relevance criteria. The figure therefore summarizes the complete progression from representation to contextual interpretation and serves as a roadmap for the methodological sections that follow.

Figure 1 summarizes the complete workflow of the proposed framework. An external document is first transformed into a canonical hierarchical state by the representation model. The resulting state is explicitly linked to its predecessor, enabling deterministic comparison against the immediately preceding version. Provided that comparable document states have been constructed according to the proposed representation model, structural differences can subsequently be localized hierarchically and interpreted according to application-specific context.

1.8. Paper Organization

The remainder of the paper is organized as follows.

Section 2 reviews related work in version control, database systems, structured document processing, and hash-based methods, and identifies the conceptual limitations motivating this study. Section 3.1 introduces the representational foundation by reformulating documents as structured states. Section 3.2 develops the temporal dimension through predecessor linkage and state evolution.

Section 3.3 defines change as a transition between comparable states and distinguishes deterministic structural detection from application-specific interpretation. Section 3.4 presents a hierarchical digest-based computational instantiation that demonstrates how the proposed representation model can be realized in an executable comparison workflow.

Section 4 provides the analytical assessment of the framework, including the evaluation of its conceptual consistency, design objectives, computational instantiation, and assessment scope. Section 5 complements this analysis with a controlled and reproducible operational case study that evaluates one reference implementation using predefined document states, ground truth, and boundary conditions. Section 6 discusses implications, limitations, and practical relevance for change-aware information systems. Finally, Section 7 summarizes the main contributions and outlines directions for future research.

2. Related Work and State of the Art

2.1. Overview

The management of evolving documents and the detection of relevant changes have been studied across several research domains. Particularly relevant streams include version control systems, database change management, hash-based integrity verification, and structured document processing. Each of these areas contributes mature mechanisms for storing, tracking, validating, or comparing changing data. However, they differ substantially in their underlying assumptions regarding what a document is, how change should be represented, and how comparison results should be interpreted.

Most existing approaches focus on selected subproblems such as differencing, event logging, synchronization, update propagation, or integrity assurance [1,7,10,11,17]. As a consequence, representation, temporal evolution, and comparison are often treated separately rather than as elements of one coherent model.

For the purposes of this paper, the key question is therefore not whether existing methods are useful, many clearly are, but whether they jointly provide a general foundation for modeling evolving documents as comparable information states. The following review evaluates the state of the art from this perspective.

2.2. Version Control Systems

Version control systems (VCS), such as Git [8] and earlier systems such as RCS [7], are among the most established approaches for managing evolving artifacts. They maintain explicit version histories and provide mechanisms for differencing, branching, merging, and reconstruction of prior states. Additional research on repository mining and collaborative software development further demonstrates the strategic importance of version histories in distributed workflows [9,18,19].

Beyond software repositories, revision-control principles have also been transferred to scientific modeling environments. For example, Zinner et al. [20] proposed a revision-control and automatic documentation strategy for numerical models, emphasizing traceability and reproducibility.

The major strength of VCS lies in reliable history management and collaboration support. Nevertheless, several limitations arise when these systems are viewed as a general framework for evolving documents.

First, VCS predominantly operate on line-based or token-based textual abstractions. Rich hierarchical semantics are typically managed outside the core version control mechanism itself and therefore depend on additional representation models or application-specific abstractions. Second, deterministic equivalence is not guaranteed, because formatting, ordering, or encoding differences may be treated as substantive changes unless external normalization is applied. Third, comparison remains an operation performed on stored artifacts rather than a property emerging from an explicitly defined representation model.

Accordingly, VCS provide powerful operational tooling, but only limited support for modeling heterogeneous documents as structured and inherently comparable states.

2.3. Database Change Detection and Data-Centric Approaches

Database systems commonly address change through triggers, transaction logs, timestamps, replication streams, and change data capture (CDC) mechanisms [1,3,14]. These methods are widely used to record, propagate, and synchronize updates in transactional and analytical environments. Related data-engineering perspectives likewise emphasize ordered event histories as a basis for downstream processing [2].

Such approaches offer strong guarantees regarding consistency, ordering, and system integration. They are particularly effective for fine-grained modifications at the level of records or attributes.

However, these approaches are fundamentally event-centric rather than state-centric. They describe what changed through operations or log entries, but do not necessarily provide a canonical representation of complete document states that can be directly compared across versions.

In addition, hierarchical document structures are often transformed into relational schemas, where structural relationships become distributed across multiple tables. Reconstructing document-level change therefore requires aggregation of multiple events and subsequent interpretation. Recent survey work on mapping semi-structured data to relational systems illustrates these structural compromises [21].

The present framework complements such approaches by shifting attention from event traces toward explicit state representation and state transitions.

2.4. Hash-Based Integrity and Hierarchical Verification

Hash functions and digest-based methods are widely used for data integrity checking and equality verification [17]. Hierarchical constructions such as Merkle trees extend this principle by aggregating hashes of subcomponents, thereby enabling scalable verification of large data structures. Subsequent work on authenticated data structures further generalized these concepts to dynamic and queryable settings [22–24].

These methods illustrate an important principle: compact derived representations can substantially reduce comparison effort. Hierarchical hashing additionally enables selective drill-down once a mismatch has been detected.

However, most hash-based approaches assume that an appropriate input representation already exists. Their main focus lies in efficient digest computation, integrity guarantees, and distributed consistency. Questions of canonicalization, semantic structure, and predecessor linkage are usually external to the hashing mechanism itself.

For this reason, hierarchical hashing should be understood less as a complete change model and more as a powerful implementation technique whose effectiveness depends on the quality of the underlying representation model.

2.5. Structured Document Models and XML-/JSON-Based Approaches

Structured document formats such as XML and JSON explicitly encode hierarchical relations and support schema-based validation. A substantial body of literature has therefore investigated tree differencing, structural matching, XML version management, and fine-grained document comparison [10–12,15,16,25]. Related work on multiversion querying further shows that document evolution can be treated as a first-class concern in structured environments [13].

Compared with purely text-based differencing, these approaches provide substantially better localization of changes within document hierarchies.

Nevertheless, most methods focus on pairwise comparison between already available document instances. They usually assume that documents are already represented in a suitable form and then optimize the differencing procedure itself.

Important prerequisites of robust comparability, such as deterministic canonicalization, normalization under representation-preserving transformations, and explicit predecessor semantics, are typically addressed separately rather than within a unified representational framework. For XML, canonicalization has been standardized by the W3C [26]. For JSON, related standards include canonicalization and patch/update formats [27–29]. While valuable, these standards do not by themselves yield an integrated framework of representation, evolution, and comparison.

2.6. Cross-Domain Synthesis of Limitations

Across the reviewed domains, five recurring limitations can be identified:

1. **Representation remains implicit.** Many approaches assume an existing representation rather than treating representation design as a primary variable [11,21].
2. **Canonicalization is incomplete or externalized.** Deterministic equivalence under formatting, ordering, or encoding transformations is not consistently guaranteed [26,27].
3. **Versioning: and comparison are weakly integrated.** Histories may exist, but predecessor semantics are not always formalized as the basis of comparison [7,12].
4. **Detection and interpretation are conflated.** Technical differences are often reported without a clear separation between structural detection and contextual relevance [11,15].
5. **Repeated comparison is underemphasized.** Many methods do not explicitly optimize scenarios involving frequent checks against evolving predecessors [3,24].

Taken together, the reviewed approaches address important individual aspects of evolving document management. However, they generally treat representation, temporal evolution, and comparison as separate concerns rather than as mutually dependent properties of a unified document-state model. This observation motivates the representation-centric perspective developed in the remainder of the paper.

A similar observation can be made in collaborative engineering decision-making. Existing work emphasizes that reliable coordination and decision processes depend on consistent, traceable, and reproducible information representations before higher-level reasoning or optimization can be applied. While these approaches investigate decision processes rather than document representation itself, they reinforce the need for well-defined information structures as the foundation for downstream information processing.

2.7. Positioning of the Proposed Framework

The framework proposed in this paper addresses these limitations by making the representational assumptions underlying change-aware information systems explicit. Its main distinguishing characteristics are:

- documents are modeled as structured states rather than static artifacts;
- canonical representation is treated as a prerequisite for deterministic comparability;
- temporal evolution is captured through explicit predecessor linkage;
- change is defined as a transition between comparable states;
- technical detection and contextual interpretation are separated conceptually.

Rather than replacing established approaches, the framework provides a unifying layer that makes their implicit assumptions explicit. It also clarifies how techniques such as hierarchical hashing, event logging, or structural differencing can be embedded into a more general representation model.

Although the computational instantiation and reproducible operational case study presented later employ established techniques such as hierarchical digest aggregation, canonical serialization, and predecessor-based version linkage, the scientific contribu-

tion of the present work is intentionally located at a different abstraction level. Existing approaches typically assume that an appropriate internal representation already exists and subsequently optimize verification, differencing, or version management. In contrast, the proposed framework specifies the representational conditions under which these mechanisms become interoperable within a coherent document-state model. Canonical state construction, explicit comparability, predecessor-constrained evolution, hierarchical refinement, and contextual interpretation are therefore treated as mutually dependent properties of the representation itself rather than as isolated implementation techniques. The accompanying case study demonstrates one executable realization of these principles under controlled and reproducible conditions but is not intended to establish algorithmic superiority or general implementation performance. Consequently, the framework should be understood as complementing rather than replacing existing digest, version control, or differencing methods.

2.8. Clarification of Novelty and Non-Contributions

The novelty of the proposed framework resides in its level of abstraction rather than in the invention of new comparison algorithms, cryptographic primitives, or canonicalization procedures. The paper does not claim to introduce a new version control system, a new tree-differencing algorithm, a new hash function, or a new authenticated data structure. These mechanisms are well established and provide important technical building blocks.

The contribution instead lies in integrating representation, temporal evolution, and comparison into a unified state-centric model. In this model, documents are first transformed into deterministic canonical states; these states are embedded in explicit predecessor relations, and change is defined as a transition between comparable states. Hashing, differencing, event logging, or database-native comparison can then be interpreted as possible operational mechanisms within this broader model. The reproducible operational case study presented later illustrates one such computational realization and demonstrates that the proposed representational conditions can be instantiated in an executable workflow without altering the conceptual scope of the contribution.

This distinction is important because many existing approaches optimize comparison after a representation has already been chosen. The proposed framework shifts attention one step earlier: it treats the representation model itself as the central prerequisite for deterministic comparability, structural localization, traceability, and repeated comparison efficiency.

The scientific contribution of the proposed framework should therefore be understood as representational rather than algorithmic. Its purpose is not to replace established comparison, verification, or version control mechanisms, but to provide a common representational foundation within which these complementary techniques can operate consistently across evolving document collections.

2.9. Summary

The literature considered here is selected according to direct conceptual relevance to canonical representation, structured comparison, temporal evolution, component correspondence, provenance, and authenticated hierarchical data structures rather than recency alone. Foundational references are retained where they define mechanisms or research traditions essential to the positioning of the proposed framework.

The state of the art offers a rich set of methods for storing, tracking, validating, and comparing evolving information. However, these methods were largely developed from different disciplinary perspectives and therefore emphasize different parts of the overall problem.

The contribution of this paper is to integrate these fragmented perspectives into a coherent conceptual foundation in which representation, temporal linkage, and comparison are treated jointly. The subsequent analytical assessment examines the internal consistency of this representation model, while the reproducible operational case study demonstrates one executable realization under controlled conditions. Together, these complementary forms of evidence support the proposed state-centric perspective without claiming general implementation superiority or benchmark performance.

Table 1 is intentionally descriptive. Its purpose is to distinguish the abstraction levels, assumptions, and scopes of adjacent research traditions rather than to assign support scores or establish superiority. The compared approaches address overlapping but non-identical problems, and their suitability depends on the application context, representation model, temporal semantics, and required form of change analysis.

Accordingly, the contribution should be understood neither as a new computational method nor as a taxonomy of existing systems. It is a conceptual and formal representation framework with design-oriented implications. Its methodological contribution lies in making explicit a set of interdependent conditions that are often treated separately: canonical state construction establishes representation stability; comparability restricts meaningful equality assessment to compatible states; component correspondence determines whether differences can be localized; predecessor linkage constrains temporal comparison scope; and contextual interpretation separates structural difference from application-specific relevance. The framework's contribution is the explicit formal integration of these conditions into one document-state model, not the invention of the individual mechanisms used to instantiate them.

Table 1. Descriptive positioning of the proposed framework relative to adjacent research paradigms. The comparison characterizes conceptual focus, representational assumptions, temporal organization, comparison scope, and principal limitations. It does not constitute an empirical ranking or performance evaluation.

Approach	Primary Focus	Representation and Temporal Organization	Comparison or Change Model	Scope Relative to the Present Study
Version control systems	Management of evolving artifacts and revision histories	Stored versions, content-addressed objects, commits, parent relations, and potentially branching histories	Artifact-, line-, token-, tree-, or object-based comparison depending on the system	Provide mature version-history and comparison mechanisms; do not generally prescribe a domain-independent representation framework for canonical document states and contextual interpretation.
Temporal and versioned databases	Storage and querying of historical data states	Time-indexed tuples, records, valid-time or transaction-time semantics, and reconstructed states	State reconstruction, temporal querying, event comparison, or database-specific change tracking	Provide explicit temporal semantics but remain coupled to database models and do not generally define hierarchical canonical document-state comparison as an integrated abstraction.
Event sourcing and change-data capture	Recording and propagation of state-changing events	Ordered event streams or database-level change events from which states may be reconstructed	Event-based reconstruction and change propagation	Provide strong temporal traceability but focus primarily on events rather than a general canonical representation model for hierarchical document states.
Structured XML/JSON differencing	Detection of structural differences in hierarchical documents	Trees or structured serializations, potentially with schemas and matching rules	Tree edit operations, structural matching, or format-specific differencing	Provide sophisticated structural comparison mechanisms; component correspondence and schema compatibility remain method-specific.

Table 1. *Cont.*

Approach	Primary Focus	Representation and Temporal Organization	Comparison or Change Model	Scope Relative to the Present Study
Merkle trees and authenticated data structures	Integrity verification, authenticated queries, and efficient identification of differing substructures	Hierarchical digest aggregation over an assumed underlying representation	Digest equality and recursive refinement	Provide established hierarchical comparison mechanisms but do not themselves define canonical document semantics, predecessor selection, component correspondence, or contextual relevance.
Data provenance and lineage systems	Tracing origins, transformations, dependencies, and derivations	Provenance graphs, lineage relations, transformation records, or dependency structures	Dependency analysis and derivation tracing	Provide rich traceability semantics but address a broader and different problem than deterministic comparison of canonical evolving document states.
Proposed state-centric framework	Formalization of representational conditions for comparable evolving document states	Canonical hierarchical states, explicit comparability assumptions, component correspondence, and predecessor-constrained temporal linkage	State-transition analysis with optional hierarchical localization and context-dependent interpretation	Provides a conceptual and formal synthesis of representation, temporal linkage, comparison, and interpretation. The framework is analytically assessed and instantiated through a reproducible operational case study but does not introduce a new differencing algorithm or claim empirical performance superiority.

3. Materials and Methods

This section develops the methodological foundation of the proposed framework. It introduces the formal representation model, temporal linkage mechanism, change formalization, and a computational instantiation that demonstrates how the framework can be realized without constraining subsequent implementation choices.

The objective is to establish a formal representation model for structured documents that permits deterministic reasoning about document identity, temporal evolution, and structural change independently of any particular implementation or storage technology. The formalization specifies the mathematical objects, mappings, and relations that constitute the conceptual basis for the analytical assessment in Section 4 and the reproducible operational case study in Section 5.

The development follows a layered construction. First, a representation model defines the admissible document universe together with a deterministic mapping into a canonical state space. On this foundation, document states are equipped with hierarchical structure, temporal relations, and comparison operators. Structural change is then characterized as a property of transitions between comparable document states. The final subsection presents one computational instantiation that demonstrates how these formal concepts can be translated into an executable comparison workflow. The correctness and scope of the framework are assessed analytically in Section 4, whereas its operational realizability is examined through a reproducible case study in Section 5.

The formal development distinguishes among assumptions, definitions, immediate consequences, and analytical properties. Definitions introduce the mathematical objects required by the framework. Some subsequent statements record direct consequences of those definitions in order to make dependencies explicit and to support later development; they are not presented as independent theoretical discoveries. Other propositions establish properties that require additional assumptions or connect several elements of the framework. The purpose of the formalization is therefore to provide a precise and internally consistent

representation model, not to claim that every formally stated consequence constitutes a non-trivial theorem.

3.1. From Documents to States

3.1.1. Limitations of File-Centric Representations

In many existing information systems, documents are treated as static artifacts, typically represented as files, database records, or loosely structured collections of attributes. Under this paradigm, change is defined implicitly through direct comparison of such artifacts. While this approach is straightforward, it exhibits fundamental limitations when applied to environments in which documents evolve continuously and must be compared repeatedly.

First, file-centric representations do not provide an explicit and stable representation of internal document structure. Although documents may exhibit rich hierarchical organization (e.g., sections, records, annotations), this structure is often embedded implicitly in storage formats and not represented in a form suitable for systematic and reproducible comparison. As a consequence, comparison operations frequently operate on flattened or partially structured data, obscuring the semantic organization of the document.

Second, direct comparison of document instances requires repeated traversal of the full representation. Even in the absence of change, the system must inspect the complete artifact to establish equality. In settings characterized by frequent polling, monitoring, or synchronization, this leads to computational effort that scales with both document size and query frequency.

Third, file-based representations lack a principled notion of comparability. Superficial variations in formatting, ordering, or encoding may lead to differences at the representation level despite semantic equivalence. Conversely, small localized changes may have disproportionate operational relevance. Without a deterministic and canonical representation, comparison results become sensitive to incidental representation choices rather than to meaningful structural differences.

Fourth, the relation between successive document versions is typically implicit or weakly defined. While multiple versions may be stored, the absence of explicit predecessor linkage allows comparison against arbitrary historical states. This weakens both computational efficiency and interpretability, as detected differences do not necessarily correspond to a coherent evolution step.

These limitations indicate that file-centric representations are insufficient for systems in which documents evolve continuously and must be compared in a consistent, efficient, and interpretable manner.

The proposed framework addresses these limitations by replacing artifact-level comparison with a staged state-based process. First, an external document artifact is transformed into a canonical document state under an explicit representation model. Second, the resulting state is decomposed into hierarchical components that provide stable units for comparison. Third, the state is linked to its valid predecessor so that change is evaluated along a coherent evolution path rather than between arbitrary files. Finally, comparison is performed on canonical state representations and refined only within structurally affected subregions. In this way, the framework converts the initially implicit problem of document change into an explicit sequence of representation, linkage, comparison, localization, and interpretation steps.

3.1.2. Formal Representation Model and Document States

To overcome the limitations of artifact-centered approaches, we introduce a state-centric perspective on documents. Rather than treating documents as static files or passive

storage objects, the proposed framework models them as structured states embedded in an evolving information space.

This perspective provides an implementation-independent basis for reasoning about document evolution, comparability, traceability, and change detection. The formalization draws on concepts from discrete mathematics, graph theory, relation theory, and deterministic state-transition modeling.

The formalization distinguishes consistently between external document representations and internal document states. External documents constitute observable artifacts that may differ in representation while preserving identical semantic content. Internal document states are canonical mathematical objects obtained by deterministic application of the representation mapping. All subsequent definitions operate exclusively on document states unless stated otherwise.

Definition 1 (Representation Model)**.** *A representation model $\mathcal{R}$ is a formally specified system of rules by which external document artifacts are interpreted and transformed into canonical document states. It determines how information is selected, organized, normalized, structured, linked, and made comparable within the system.*

A representation model typically comprises scope rules, structural rules, canonicalization rules, context rules, and comparison rules. Hence, $\mathcal{R}$ defines the semantic and structural boundary within which document states are constructed and change becomes formally evaluable.

Proposition 1 (Representation Consistency)**.** *Let $\Phi : \mathcal{D} \to \mathcal{S}$ be defined under a fixed representation model $\mathcal{R}$. If Φ is deterministic and representation-invariant, then equivalent admissible document instances yield identical document states and identical canonical serializations.*

Proof. By Definition 1, the representation mapping Φ is deterministic on the admissible document universe $\mathcal{D}$. Consequently, identical admissible documents are mapped to identical document states. Furthermore, the normalization rules prescribed by the fixed representation model $\mathcal{R}$ eliminate all representation-dependent variability. Hence, semantically equivalent admissible documents induce the same canonical state. Since canonical serialization is uniquely determined by the resulting state, both documents yield identical serialized representations. □

Proposition 1 holds within the validity domain of the fixed representation model $\mathcal{R}$. Accordingly, all representation-dependent ambiguities, including character normalization, locale-specific ordering, floating-point normalization, or comparable implementation-dependent effects, are assumed to be resolved by the canonicalization rules associated with $\mathcal{R}$. These assumptions define the admissible input domain of the formal model rather than additional properties of the comparison framework.

Proposition 2 (Determinacy of Canonical Serialization)**.** *Let* ser $: \mathcal{S} \to \Sigma^*$ *be the canonical serialization induced by the fixed representation model $\mathcal{R}$. Then every state $s \in \mathcal{S}$ has a unique canonical serialization* ser(s)*, and repeated evaluation of* ser *on the same state yields the same result.*

Proof. The serialization mapping is deterministic under the fixed representation model $\mathcal{R}$. Hence, for every admissible state $s \in \mathcal{S}$, the value ser(s) is uniquely determined. Repeated evaluation uses the same state and the same serialization rules and therefore returns the same canonical representation. □

This property guarantees stability under repeated processing and prevents normalization from generating artificial differences.

Definition 2 (Document Universe)**.** *Let $\mathcal{R}$ be a fixed representation model. Let $\mathcal{D}$ be the set containing all externally represented information artifacts satisfying the input assumptions of the chosen application domain. Let $\mathcal{S}$ be the set containing all canonical document states constructible from elements of $\mathcal{D}$ under $\mathcal{R}$. Then $\mathcal{D}$ is called the document universe, $\mathcal{S}$ the document state space.*

Typical elements of a document universe include files, database records, messages, structured payloads, or equivalent document-bearing objects.

Definition 3 (Representation Mapping)**.** *Let $\mathcal{R}$ be a given fixed representation model, $\mathcal{D}$ the document universe under $\mathcal{R}$, and $\mathcal{S}$ the document state space under $\mathcal{R}$. Let Φ be function $\Phi : \mathcal{D} \to \mathcal{S}$ that transforms an admissible external document artifact into a canonical internal state representation. The mapping Φ is assumed to be deterministic under fixed transformation rules. Then the function Φ is called representation mapping.*

The mapping Φ typically consists of parsing, normalization, structuring, and context enrichment. Parsing extracts relevant content, attributes, and structural elements from the source artifact. Normalization removes incidental representational variation such as formatting, ordering, encoding, or delimiter differences. Structuring transforms the result into a hierarchical component model with explicit logical relations. Context enrichment attaches metadata required for identity, provenance, scope, or predecessor linkage.

The mapping Φ establishes a strict separation between external representations, such as file formats or database schemas, and internal state representations, which serve as the basis for comparison, version linkage, and change analysis.

Definition 4 (Document State)**.** *Let $\Phi : \mathcal{D} \to \mathcal{S}$ be a representation mapping under a fixed representation model $\mathcal{R}$. A* document state *is an element $s \in \mathcal{S}$ obtained from some document instance $d \in \mathcal{D}$ by $s = \Phi(d)$.*

For $d_t \in \mathcal{D}$, the associated state is $s_t = \Phi(d_t)$. Thus, a document state is not merely a stored object, but the outcome of a formally defined transformation process. It is characterized by hierarchical decomposition into logical components, deterministic construction under fixed transformation rules, contextual metadata specifying identity and scope, and linkage to related states in an evolving sequence. Canonical representation ensures that inputs equivalent under $\mathcal{R}$ yield identical internal representations under $\mathcal{R}$, thereby establishing the basis for deterministic comparison, explicit temporal linkage, and traceable document evolution.

Proposition 3 (Properties of the Representation Mapping)**.** *Let $\Phi : \mathcal{D} \to \mathcal{S}$ be defined under a fixed representation model $\mathcal{R}$. Then, Φ satisfies the following properties:*

1. *Determinism: identical inputs produce identical states;*
2. *Representation Invariance: equivalent admissible inputs map to identical states;*
3. *Structural Preservation: provided that a valid component-correspondence relation exists, localized content differences are reflected in the corresponding state substructures;*
4. *Model-Based Comparability: resulting states can be compared deterministically under $\mathcal{R}$.*

Proof. Determinism follows because all transformation steps of Φ are rule-based and non-stochastic. Representation invariance follows from canonicalization: admissible differences in formatting, ordering, encoding, or delimiters are normalized before state construction.

Structural preservation follows, under a valid component-correspondence relation, because Φ maps logically corresponding document components to corresponding state substructures. Model-based comparability follows because all states are generated under the same representation model $\mathcal{R}$, including shared structural and canonicalization rules. □

The preceding definitions establish deterministic construction of canonical document states. Deterministic comparison, however, requires more than uniqueness of the global state representation. It additionally requires an internal structure that permits correspondence between individual document fragments, recursive refinement of affected regions, and localized reasoning about structural differences.

The next subsection therefore introduces hierarchical document states together with their recursive decomposition into structural components. This hierarchy constitutes the mathematical foundation for all subsequent notions of comparability, temporal evolution, and structural change.

3.1.3. Hierarchical Organization and Structural Semantics

A defining characteristic of document states is their explicit hierarchical organization. In contrast to flat attribute vectors or unstructured record sequences, the proposed framework models internal document composition directly. This is essential because many real-world documents consist of nested logical units such as sections, tables, records, annotations, procedures, and metadata containers. Ignoring such structure weakens both interpretability and comparison efficiency.

Let a document state at time index t be defined as

$$s_t = \left(c_t, u_t^{(1)}, u_t^{(2)}, \ldots, u_t^{(n_t)}\right), \tag{1}$$

where c_t denotes contextual information (e.g., identity, scope, provenance, timestamps, schema version), and each $u_t^{(i)}$ denotes a structured component. The number of components n_t may vary across states while remaining valid under the shared representation model.

Each component may itself contain subordinate components. Hence, the document state induces a rooted ordered tree.

Definition 5 (State Tree Representation). *For a document state s_t, the associated structural tree is the rooted ordered tree*

$$\mathcal{T}(s_t) = (V_t, E_t, r_t), \tag{2}$$

where V_t is the finite set of nodes representing structural units, $E_t \subseteq V_t \times V_t$ is the parent–child relation, $r_t \in V_t$ is the unique root node representing the complete document state.

The graph $\mathcal{T}(s_t)$ satisfies the standard tree properties: $|E_t| = |V_t| - 1$, and every node $v \in V_t \setminus \{r_t\}$ has exactly one parent. Thus, each component has a unique structural position within the document state.

Definition 6 (Subtree). *For a node $v \in V_t$, the subtree* $\mathrm{sub}(v)$ *is the rooted tree induced by v and all descendants reachable from v.*

The definition of subtrees enables comparison not only of complete states, but also of localized structural regions.

Figure 2 provides an intuitive visualization of the hierarchical document-state model introduced formally in the preceding definitions. Each node corresponds to a canonical structural component that can subsequently participate in deterministic comparison and hierarchical refinement. Thus, a document state can be modeled as a rooted hierarchical

structure in which components occupy explicit structural positions, thereby enabling multi-level comparison and localized change analysis.

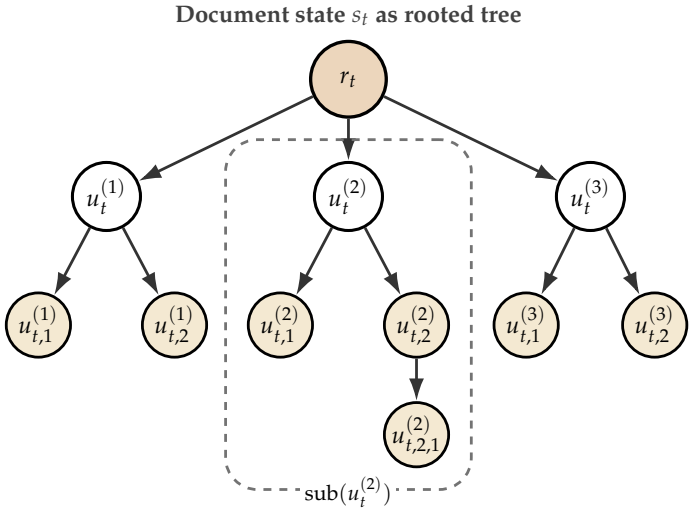


Figure 2. Hierarchical tree representation of a document state. The root node r_t represents the complete state s_t, internal and leaf nodes represent structural components, and directed edges encode parent–child relations. Highlighted subtrees define localized regions for comparison and refinement. Nodes represent document components, edges encode hierarchy. subtrees define local comparison regions.

The principal consequences of hierarchical state representation are summarized in Table 2.

Table 2. Essential properties of hierarchical document-state representations. The table summarizes the structural characteristics that enable scalable comparison, localized refinement, and semantically meaningful change analysis. The "Value" column indicates the relative significance of each property (■■■: High, ■■■: Moderate).

Property	Description	Value
Multi-level comparability	Comparison can be performed at different granularities, including whole-state comparison and subtree comparison for corresponding nodes.	■■■
Localization of change	If two comparable states differ, the change can be localized to affected nodes or subtrees.	■■■
Structural semantics	Structural organization is part of the representation itself. Consequently, changes in nesting, grouping, ordering (when semantically relevant), or component membership are treated as genuine structural changes rather than incidental formatting effects.	■■■

While hierarchy defines the internal organization of document states, meaningful relations between distinct states require an explicit notion of comparability and canonical equivalence, introduced next.

3.1.4. Comparability and Canonical Representation

The introduction of document states permits a precise notion of comparability. In the proposed framework, comparison is meaningful only when states are generated under a common representation model. Without this requirement, equality may depend on incompatible schemas, differing structural assumptions, or non-identical normalization rules.

The framework assumes that semantically equivalent document states are transformed into identical canonical representations by an application-appropriate canonicalization

procedure. The correctness of this procedure is therefore an external assumption rather than a contribution of the framework itself.

Canonical representation is independent of any specific storage or exchange format such as XML, JSON, relational tuples, or binary encodings. Its defining property is determinism: inputs equivalent under a fixed representational model must yield identical canonical representations.

Definition 7 (Comparability)**.** *Two document states $s_a, s_b \in \mathcal{S}$ are called comparable, written $s_a \sim s_b$, if they are constructed under the same representation model $\mathcal{R}$. In particular, they must share:*

1. *identical hierarchical decomposition rules,*
2. *identical canonicalization and normalization rules,*
3. *identical serialization rules, and*
4. *a common correspondence domain for matching components.*

Thus, comparability is a prerequisite for valid equality testing. It ensures that both states are expressed within the same formal representation system.

Definition 8 (State Equivalence)**.** *Let $s_a, s_b \in \mathcal{S}$ be two comparable states. The states are called* equivalent*, written $s_a \equiv s_b$, if and only if their canonical serializations coincide:*

$$\mathrm{ser}(s_a) = \mathrm{ser}(s_b),$$

where ser $: \mathcal{S} \to \Sigma^*$ *is a deterministic serialization mapping into finite symbol strings. Consequently, state equivalence is uniquely determined by canonical serialization under the fixed representation model $\mathcal{R}$.*

Thus, comparison outcomes are independent of incidental representation differences once both states are constructed under the same formal model.

Proposition 4 (Properties of State Equivalence)**.** *Let $s_a, s_b, s_c \in \mathcal{S}$ be pairwise comparable. Then, the relation $\equiv$ has the following properties:*

1. *Determinism: repeated evaluation under identical inputs yields identical outcomes;*
2. *Representation Invariance: admissible differences eliminated by canonicalization do not affect equivalence;*
3. *Symmetry:*
$$s_a \equiv s_b \Longrightarrow s_b \equiv s_a;$$
4. *Reflexivity:*
$$s_a \equiv s_a;$$
5. *Transitivity:*
$$s_a \equiv s_b \ \wedge \ s_b \equiv s_c \Longrightarrow s_a \equiv s_c.$$

Hence, on every fixed comparability class, $\equiv$ is an equivalence relation.

Proof. Determinism follows directly from determinism of ser. Representation invariance follows because all admissible representation-dependent variations are eliminated during canonicalization prior to serialization. Reflexivity, symmetry, and transitivity are inherited from equality in the codomain of ser. Hence, $\equiv$ is an equivalence relation on every comparability class induced by the fixed representation model $\mathcal{R}$. □

Definition 9 (Subtree Equivalence)**.** *Let $s_a \sim s_b$ be hierarchical states with structural trees $\mathcal{T}(s_a) = (V_a, E_a, r_a)$ and $\mathcal{T}(s_b) = (V_b, E_b, r_b)$. Let $v_a \in V_a$ and $v_b \in V_b$ be corresponding nodes. The rooted subtrees induced by these nodes are called equivalent if*

$$\mathrm{sub}(v_a) \equiv \mathrm{sub}(v_b),$$

that is, if their canonical serializations coincide under the shared representation model.

This permits comparison at multiple granularities, including whole-state, component-level, and subtree-level comparison.

Proposition 5 (Hierarchical Comparison Advantage)**.** *Let $s_a \sim s_b \in \mathcal{S}$ be comparable hierarchical states. If $\mathrm{sub}(v_a) \equiv \mathrm{sub}(v_b)$ for corresponding nodes v_a, v_b, then all descendants contained in these subtrees are equivalent under the shared representation model and need not be re-evaluated individually.*

Proof. Canonical serialization of a subtree uniquely determines the serialized representation of every descendant contained therein. Hence, equality of the complete subtree serialization excludes the existence of any differing descendant. Otherwise, the serialized subtree representations would differ, contradicting the premise. □

This property explains why hierarchy enables pruning of comparison effort and scalable refinement.

Definition 10 (Structural Difference Set)**.** *Let $s_a \sim s_b$ with node sets V_a and V_b. The structural difference set is*

$$\Delta(s_a, s_b) = \{(v_a, v_b) \in V_a \times V_b \mid \mathrm{ser}(\mathrm{sub}(v_a)) \neq \mathrm{ser}(\mathrm{sub}(v_b))\},$$

restricted to corresponding node pairs under the shared model.

Proposition 6 (Localization of Structural Change)**.** *Let $s_a, s_b \in \mathcal{S}$ be two comparable states with $s_a \not\equiv s_b$. Then at least one corresponding node pair differs in canonical representation:*

$$\Delta(s_a, s_b) \neq \varnothing.$$

Proof. Assume $\Delta(s_a, s_b) = \varnothing$. Then, every corresponding subtree possesses identical canonical serialization. In particular, the root subtrees coincide. Therefore $\mathrm{ser}(s_a) = \mathrm{ser}(s_b)$, implying $s_a \equiv s_b$, contradicting the premise. □

Remark 1. *The preceding constructions establish document states as canonical mathematical objects endowed with hierarchical structure and deterministic comparison semantics. This permits a further abstraction in which individual states become vertices of a state space connected by admissible temporal transitions.*

3.1.5. State Space Abstraction

The preceding notions of hierarchy, comparability, and canonical equivalence define document states as stable mathematical objects. The concept of document states permits a further level of abstraction: documents are no longer treated as isolated artifacts, but as elements of a formally defined state space. This shift is important because many practical tasks, including comparison, synchronization, traceability, auditing, and change propagation, concern not a single artifact, but relations among multiple evolving instances.

Definition 11 (State Space). *Let $\mathcal{R}$ be a fixed representation model. The document state space induced by $\mathcal{R}$ is the set*

$$\mathcal{S}_{\mathcal{R}} = \{s \mid s \text{ is a valid document state constructed under } \mathcal{R}\}.$$

The state space is naturally equipped with binary relations such as a comparability relation $\sim \subseteq \mathcal{S}_{\mathcal{R}} \times \mathcal{S}_{\mathcal{R}}$ and a successor relation $\to \subseteq \mathcal{S}_{\mathcal{R}} \times \mathcal{S}_{\mathcal{R}}$, describing admissible temporal transitions. Thus, a document is represented not merely by one state but by an ordered evolution path

$$s_{t_1}, s_{t_2}, \ldots, s_{t_k} \in \mathcal{S}_{\mathcal{R}}, \qquad s_{t_i} \to s_{t_{i+1}} \ \ (i = 1, \ldots, k-1).$$

Definition 12 (Evolution Path). *A finite or infinite sequence*

$$\pi = (s_{t_1}, s_{t_2}, \ldots)$$

is called an evolution path if every consecutive pair satisfies

$$s_{t_i} \to s_{t_{i+1}}.$$

The sequence above represents a temporally consistent trajectory of the same logical document under the shared representation model.

Proposition 7 (Graph-Theoretic Interpretation of Document Evolution). *Let*

$$G_{\mathcal{R}} = (\mathcal{S}_{\mathcal{R}}, \to)$$

be the directed graph induced by the successor relation. Then:

1. *every evolution path is a directed path in $G_{\mathcal{R}}$;*
2. *every finite document history corresponds to a finite directed path;*
3. *graph-theoretic notions such as branching, convergence, reachability, and path length are therefore directly applicable to document evolution.*

Proof. The vertices of $G_{\mathcal{R}}$ are precisely the admissible document states, while directed edges are exactly the successor relation. Every evolution path therefore satisfies the definition of a directed path. The remaining statements follow immediately from elementary graph theory. □

This abstraction enables reasoning about document evolution independently of storage formats, file encodings, database implementations, or application-specific interfaces. It also provides the formal basis for the temporal predecessor relations and change model developed in the following sections (Table 3).

Table 3. Summary of the principal elements of the proposed representation framework.

Context	Purpose
Representation model	Defines canonical state construction
Document state	Canonical representation of an information artifact
State tree	Hierarchical decomposition for localized comparison
Comparability	Preconditions for deterministic comparison
Successor relation	Represents document evolution
Interpretation	Assigns application-specific relevance

3.1.6. Conceptual Implications

The preceding formal development admits several immediate conceptual consequences that are relevant for the design of change-aware information systems.

1. Comparison is reformulated as a relation between canonical document states rather than between raw files, storage objects, or incidental encodings. Equality testing therefore becomes deterministic, reproducible, and explicitly governed by the chosen representation model.
2. Temporal succession is embedded directly in the successor relation $\rightarrow$. Versions are no longer external labels attached to artifacts, but ordered states along an evolution path within $\mathcal{S}_{\mathcal{R}}$. This provides an intrinsic representation of document history.
3. The framework separates structural detection from contextual interpretation. Whether two states differ is a computational question determined by the representation model, whereas the significance of that difference depends on external criteria such as domain rules, risk assessments, organizational priorities, or user objectives.
4. Stable, canonical, and hierarchical states enable efficient comparison strategies, including subtree refinement, digest-based equality checks, incremental evaluation, and selective traversal. Hence scalability arises primarily from representational design rather than from algorithmic optimization alone.
5. Because document evolution is represented as a path or subgraph of $G_{\mathcal{R}} = (\mathcal{S}_{\mathcal{R}}, \rightarrow)$, one may analyze branching, rollback, drift, convergence, predecessor consistency, and propagation effects using graph-theoretic methods.

The state-space perspective replaces artifact-centered reasoning with a structured and evolution-aware model. It thereby provides a coherent theoretical basis for temporal linkage, formal change definitions, localized comparison, and scalable information system architectures.

3.2. Temporal Evolution and Version Linkage

The preceding state-space abstraction establishes document states as comparable objects connected by admissible relations. We now refine this view by introducing an explicit temporal structure that models how states succeed one another over time and how version linkage constrains valid comparison.

3.2.1. Temporal Interpretation of Successor Relations

Within the state-space framework introduced above, document states are not isolated entities but temporally related elements of an evolution process. The successor relation $\rightarrow \subseteq \mathcal{S}_{\mathcal{R}} \times \mathcal{S}_{\mathcal{R}}$ therefore admits a temporal interpretation: if $s_{t_i} \rightarrow s_{t_{i+1}}$, then $s_{t_{i+1}}$ is a valid later state of the same logical document under the representation model $\mathcal{R}$. Accordingly, document evolution is represented by chains of successive states

$$s_{t_1} \rightarrow s_{t_2} \rightarrow \cdots \rightarrow s_{t_k}.$$

The index t denotes logical order rather than necessarily physical clock time. It captures the succession of version events, approvals, synchronizations, or other admissible update operations.

Each state provides a complete canonical representation at one stage of evolution, while the successor relation links these states into a coherent version history. Consequently, change is evaluated between temporally related states rather than between arbitrary stored artifacts.

Depending on the application, one transition may represent a single edit, a validated release, a synchronization step, or an aggregated batch update.

The notions of document state, state space, successor relation, and evolution path have already been established in Section 3.1. Building upon these concepts, the temporal model introduced below specifies how predecessor relations determine admissible comparison scope within an evolution path. Rather than comparing arbitrary historical states, comparison is anchored to the contextually valid predecessor of the current state.

As shown in Figure 3, predecessor linkage does not merely impose ordering; it defines the valid local scope of comparison by anchoring each state to a coherent evolution step within its path.

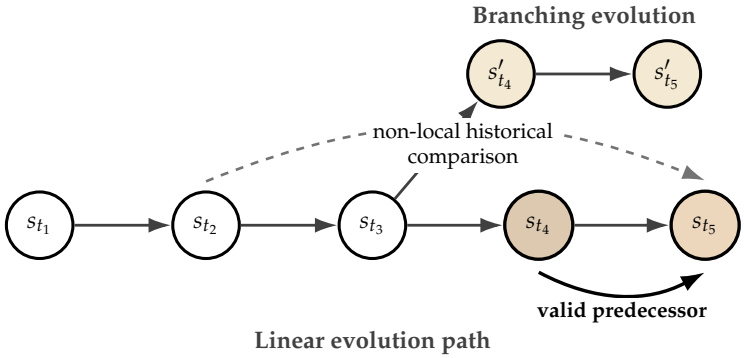


Figure 3. Temporal evolution in the proposed framework. Document states form linear or branching paths in the state graph $G_{\mathcal{R}}$. For the current state s_{t_5}, comparison is anchored to the valid predecessor s_{t_4} rather than to a non-local historical state such as s_{t_2}.

3.2.2. Predecessor Relations and Comparison Scope

To determine the valid reference state for change detection, temporal linkage is modeled explicitly through a predecessor mapping. This mapping transforms version linkage from external metadata into a formal component of the state model.

Definition 13 (Predecessor Mapping)**.** *Let $\mathcal{S}_{\mathcal{R}}$ denote the document state space induced by the fixed representation model $\mathcal{R}$. A predecessor mapping is a partial function* pred : $\mathcal{S}_{\mathcal{R}} \to \mathcal{S}_{\mathcal{R}}$ *that assigns to every non-initial state its unique contextually admissible predecessor. States for which* pred *is undefined are called initial states.*

Definition 14 (Contextually Valid Predecessor)**.** *Let* $\text{pred}(s) = s^-$*. Then, s^- is called the contextually valid predecessor of s if $s^- \to s$ and both states belong to the same comparison context induced by the fixed representation model.*

The comparison context may include branch identity, document identity, schema version, product variant, approval state, synchronization scope, or organizational domain. Hence, predecessorhood is not determined by chronological order alone.

Proposition 8 (Contextuality of Temporal Reference)**.** *Temporal order alone is insufficient to define a valid predecessor. Two states may satisfy $t_a < t_b$ without forming a valid predecessor–successor pair.*

Proof. Chronological order alone establishes only temporal precedence. It does not guarantee identical document identity, branch membership, schema compatibility, or comparison context. Consequently, temporal precedence alone does not imply a valid predecessor relation. □

Proposition 9 (Unique Local Comparison Scope)**.** *If predecessor selection is deterministic within a fixed comparison context, every non-initial state possesses at most one valid predecessor. Consequently, the admissible comparison partner of a state is uniquely determined by* $\text{pred}(s)$.

Proof. Since pred is a function, it assigns at most one predecessor to every state. Therefore, the comparison reference state is uniquely determined by $\mathrm{pred}(s)$ whenever the predecessor exists. □

Accordingly, comparison is restricted to the predecessor pair s vs. $\mathrm{pred}(s)$, so that detected differences correspond to one coherent evolution step rather than an arbitrary historical distance.

Thus, the temporal model does not merely impose ordering. It determines admissible comparison scope, preserves path consistency, and provides the semantic basis for reproducible change detection.

3.2.3. Branching, Granularity, and Historical Reconstruction

The framework does not require globally linear version histories. A state may have multiple admissible successors, for example $s_{t_i} \to s^{(1)}_{t_{i+1}}$ and $s_{t_i} \to s^{(2)}_{t_{i+1}}$. Such branching may represent collaborative editing, product variants, alternative configurations, distributed synchronization, or parallel development paths. Predecessor selection remains well-defined as long as the comparison context is fixed.

The present predecessor mapping nevertheless represents a predecessor-selected evolution path rather than a complete version-graph or merge model. Although one state may have several successors, each non-initial state is assigned at most one contextually valid predecessor within the selected comparison context. The formalization therefore supports branching histories viewed along individually selected paths, but it does not define multiple-parent states, merge commits, conflict-resolution rules, branch reconciliation, or the aggregation of changes from several predecessors.

Representing such cases would require an extension from the partial predecessor function

$$\mathrm{pred} : \mathcal{S}_{\mathcal{R}} \to \mathcal{S}_{\mathcal{R}}$$

to a relation or set-valued mapping capable of associating one state with several parent states. It would also require application-specific semantics for conflict detection, precedence, correspondence across branches, and merge-result interpretation. These extensions lie outside the scope of the present framework.

The logical time index t denotes an ordered versioning event whose granularity is model-dependent. In fine-grained versioning, each modification may yield a new state. In coarse-grained versioning, several modifications may be aggregated into one transition. The selected granularity therefore affects traceability, storage effort, and the temporal precision of later change interpretation.

Because states are explicitly linked, prior histories can be reconstructed by repeated application of the predecessor mapping $s, \mathrm{pred}(s), \mathrm{pred}^2(s), \ldots$ until an initial state is reached.

Proposition 10 (Traceability through Explicit Linkage)**.** *Assume that every non-initial state has a unique predecessor within a fixed comparison context and that predecessor chains terminate at an initial state. Then each state belongs to a uniquely reconstructable evolution path.*

Proof. Starting from a state s, repeated application of pred yields a unique backward sequence because pred is functional. By assumption, this sequence terminates at an initial state. Reversing the sequence reconstructs the evolution path of s within the fixed comparison context. □

Figure 4 illustrates this reconstruction principle. Even in the presence of branching, the predecessor mapping identifies the contextually valid path used for comparison and traceability.

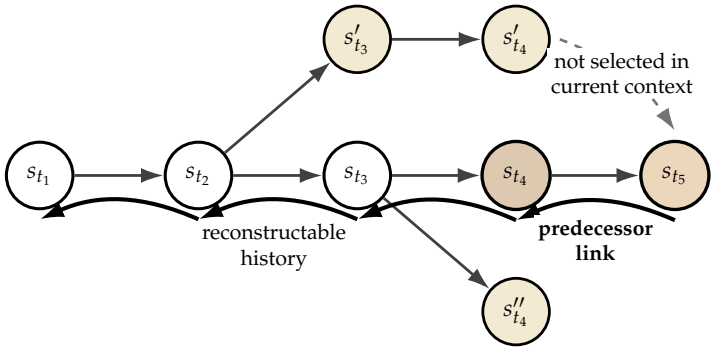


Figure 4. Predecessor-based historical reconstruction in a branching state graph. Although several branches may exist, the predecessor mapping identifies the contextually valid path for the current state s_{t_5}. Repeated application of pred$(\cdot)$ reconstructs the corresponding evolution history until an initial state is reached.

This property supports auditability, rollback, explanation of changes, detection of missing transitions, and lifecycle analysis.

3.2.4. Implications for Change Modeling

The temporal constructions developed above establish the admissible comparison relation required by the subsequent change model. By defining change relative to pred(s), comparison is applied only to well-formed predecessor–successor pairs. As a result, detected differences correspond to coherent update steps, and interpretation is anchored in the actual document lifecycle.

Without explicit predecessor linkage, comparison would rely on arbitrary historical references or accumulate unrelated differences across distant versions. Temporal linkage is therefore a necessary condition for meaningful and reproducible change detection.

3.2.5. Summary

The temporal model complements the representation model by introducing admissible evolution relations between canonical document states. Rather than treating version histories as external metadata, predecessor relations become intrinsic elements of the formal framework. Consequently, comparison is restricted to contextually valid predecessor–successor pairs, ensuring reproducible, traceable, and semantically coherent change detection. Together with the hierarchical representation developed previously, this temporal abstraction forms the foundation of the formal change model introduced in the following section.

3.3. Change Model

The preceding section established documents as temporally linked states within a structured evolution graph. Once valid predecessor–successor relations are available, the next question is how change between such states can be defined rigorously. The present section addresses this question by introducing a representation-based model of change grounded in canonical state comparison.

3.3.1. From Difference to Structured Change

Within the proposed state-centric framework, change is not interpreted as an arbitrary discrepancy between external artifacts, but as a formally specified relation between comparable document states in a shared representation space. This shifts the notion of change away from implementation-dependent encodings such as files, records, or raw serialized objects toward explicitly modeled and structurally interpretable states.

Conventional approaches often compare artifacts directly by means of field-wise checks, textual differencing, checksums, or binary mismatch detection. While such procedures may reveal deviations, they do not by themselves determine whether a difference is structurally meaningful, caused only by incidental formatting, or relevant within the given application context. They therefore tend to conflate representation-level variation, actual state modification, and contextual significance.

The proposed framework resolves this ambiguity by defining change exclusively with respect to canonical document states generated under a common representation model. Under this view, change becomes a reproducible property of the state representation itself, whereas its practical significance remains external and depends on domain rules, user objectives, or operational context.

Hence, the framework explicitly separates what changed from why it matters.

3.3.2. Formal Definition of Change

Definition 15 (State Change). *Let $s_t \in \mathcal{S}_{\mathcal{R}}$ be a non-initial document state with $\mathrm{pred}(s_t) = s_{t^-}$, where s_{t^-} denotes the contextually valid predecessor of s_t. Let $\mathrm{ser}(\cdot)$ denote the deterministic canonical serialization induced by the fixed representation model $\mathcal{R}$. A state change at s_t exists if and only if the canonical representations of s_t and its predecessor differ:*

$$\Delta(s_{t^-}, s_t) \neq \varnothing \quad \Longleftrightarrow \quad \mathrm{ser}(s_{t^-}) \neq \mathrm{ser}(s_t).$$

Equivalently,

$$\Delta(s_{t^-}, s_t) = \varnothing \quad \Longleftrightarrow \quad s_{t^-} \equiv s_t.$$

This definition establishes change as a binary whole-state property relative to the contextually valid predecessor. A state is either representationally identical to its predecessor, or it is not. The formulation is intentionally independent of any specific implementation technique such as hashing, tree differencing, delta encoding, or database logs. Because canonical serialization is deterministic under the fixed representation model $\mathcal{R}$, the existence of a state change is uniquely determined.

If hierarchical decomposition is available, the binary change predicate may be refined into localized structural differences.

Definition 16 (Localized Change Set). *Let $\mathcal{T}(s_{t^-}) = (V_{t^-}, E_{t^-}, r_{t^-})$, $\mathcal{T}(s_t) = (V_t, E_t, r_t)$ be the structural tree representations of two comparable states. The localized change set is defined as*

$$\Delta(s_{t^-}, s_t) \subseteq V_{t^-} \times V_t$$

containing corresponding nodes whose canonical representations differ.

3.3.3. Component Correspondence and Scope of Deterministic Localization

Deterministic localization requires a reproducible correspondence relation between the structural units of predecessor and successor states. Let

$$\mathcal{C}_{t^-,t} \subseteq V_{t^-} \times V_t$$

denote the correspondence relation between the node sets of the structural trees

$$\mathcal{T}(s_{t^-}) = (V_{t^-}, E_{t^-}, r_{t^-}) \qquad \text{and} \qquad \mathcal{T}(s_t) = (V_t, E_t, r_t).$$

A pair

$$(v^-, v) \in \mathcal{C}_{t^-,t}$$

indicates that the nodes v^- and v are treated as corresponding structural units for localized comparison.

The framework does not prescribe a universal procedure for constructing $\mathcal{C}_{t^-,t}$. Depending on the application domain and representation model, correspondence may be established through persistent identifiers, schema-defined keys, stable structural paths, canonical component names, deterministic positional rules, or application-specific matching procedures. For example, structured engineering records may rely on persistent component identifiers, whereas schema-constrained document trees may use unique keys or stable paths.

The validity of the correspondence relation is an explicit prerequisite for deterministic localization rather than a property guaranteed by the framework itself. Four cases must therefore be distinguished:

1. Complete and unambiguous correspondence: every relevant component has one uniquely determined counterpart. Component-level comparison and localization are deterministic.
2. Partial correspondence: Valid counterparts exist only for a subset of components. Localization is restricted to that subset, while unmatched components may be classified as insertions or deletions only if the representation model provides corresponding identity and presence rules.
3. Ambiguous correspondence: One component has several admissible counterparts. Unique localization is not possible unless an external disambiguation rule or matching procedure resolves the ambiguity.
4. Unavailable correspondence: No valid component relation can be established. Whole-state inequality may still be detected through canonical state comparison, but component-level localization is not guaranteed.

Accordingly, all claims concerning deterministic component-level localization in the following sections are conditional on the availability of a valid and sufficiently unambiguous correspondence relation. Establishing semantic correspondence between heterogeneous, weakly structured, or substantially reorganized documents is a separate research problem addressed by methods such as tree matching, graph matching, entity resolution, and schema alignment. The present framework remains independent of a particular matching algorithm and specifies the conditions under which localization becomes deterministic once correspondence has been established.

The following definitions therefore assume that the correspondence relation required by the selected representation model has already been established for the relevant components of the predecessor and successor states.

3.3.4. Hierarchical Decomposition of Change

Because document states are hierarchically structured, global state change can be refined into localized structural differences. This is a principal advantage of the state-centric framework: whole-state inequality need not imply uniform re-evaluation of all internal components.

Definition 17 (Component Change). *Let* $s_t = \left(c_t, u_t^{(1)}, u_t^{(2)}, \ldots, u_t^{(n_t)}\right)$, *where* c_t *denotes contextual information and* $u_t^{(i)}$ *denotes the ith structured component. Let* s_t *be a non-initial state with predecessor* $\mathrm{pred}(s_t) = s_{t^-}$. *Assume that corresponding components between* s_{t^-} *and* s_t *are*

established under the shared representation model. For index i, the component pair $\left(u_{t^-}^{(i)}, u_t^{(i)}\right)$ is said to exhibit a component change if

$$\mathrm{ser}\left(u_{t^-}^{(i)}\right) \neq \mathrm{ser}\left(u_t^{(i)}\right).$$

The corresponding set of changed component indices between predecessor and current state is defined by:

$$\Delta(s_{t^-}, s_t) = \left\{ i \mid \mathrm{ser}\left(u_{t^-}^{(i)}\right) \neq \mathrm{ser}\left(u_t^{(i)}\right) \right\}.$$

Thus, whole-state change is decomposed into a finite set of localized structural differences.

Proposition 11 (Localization Property)**.** *Let*

$$J = \Delta(s_{t^-}, s_t).$$

Then

$$i \in J \iff \mathrm{ser}(u_{t^-}^{(i)}) \neq \mathrm{ser}(u_t^{(i)}).$$

Consequently,

$$i \notin J \iff \mathrm{ser}(u_{t^-}^{(i)}) = \mathrm{ser}(u_t^{(i)}).$$

Proof. Both implications follow directly from the definition of the component change set. □

This locality property enables hierarchical refinement and selective traversal of affected substructures rather than uniform comparison of the complete representation.

Proposition 12 (Hierarchical Refinement Principle)**.** *Let*

$$J = \Delta(s_{t^-}, s_t).$$

Then

1. *$s_{t^-} \not\equiv s_t$ implies $J \neq \varnothing$;*
2. *only components indexed by J require recursive refinement;*
3. *all $i \notin J$ may be treated as unchanged under $\mathcal{R}$.*

Proof. Assume first that $s_{t^-} \not\equiv s_t$ and suppose, for contradiction, that $J = \varnothing$. Then every corresponding component has identical canonical serialization. Together with identical contextual data under the fixed representation model, this implies $\mathrm{ser}(s_{t^-}) = \mathrm{ser}(s_t)$ and hence $s_{t^-} \equiv s_t$, contradicting the premise. Therefore $J \neq \varnothing$. For every $i \notin J$, the component serializations are equal by definition of J. Such components are unchanged under $\mathcal{R}$ and therefore do not require recursive refinement. Only indices in J can contain localized structural differences and therefore require further analysis. □

This locality principle is central to scalable comparison because it permits selective drill-down instead of exhaustive re-analysis of unchanged regions.

Figure 5 illustrates how state-level inequality can be separated from component-level localization and refined recursively within the affected subtree.

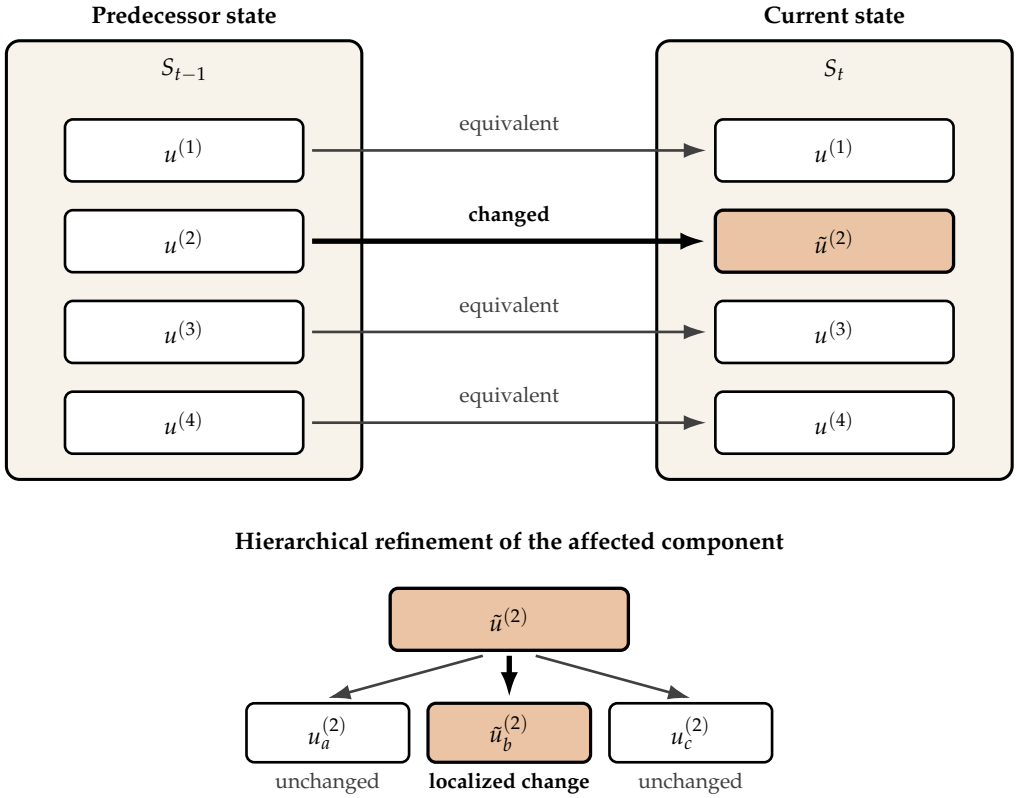


Figure 5. State equivalence, component-level change, and hierarchical refinement in the proposed change model. Equality is first evaluated at the document-state level. If inequality is detected, refinement proceeds within the hierarchy to localize the affected component and, if necessary, the changed subcomponent. In the example, the second component differs between S_{t-1} and S_t, and refinement localizes the change further to $\tilde{u}_b^{(2)}$.

3.3.5. Types of Structural Change

Within the hierarchical model, localized change can be classified by the transformation affecting corresponding components.

Definition 18 (Structural Change Types)**.** *Let* $\mathrm{pred}(s_t) = s_{t^-}$. *A localized difference between* s_{t^-} *and* s_t *belongs to one or more of the following categories:*

1. *Insertion: a component exists in* s_t *without corresponding predecessor component;*
2. *Deletion: a component of* s_{t^-} *has no corresponding successor component;*
3. *Modification: corresponding components exist in both states, but their canonical representations differ;*
4. *Reordering: corresponding components persist, but their order changes within an order-sensitive parent structure.*

These categories are structural rather than semantic. Their detectability depends on the representation model, in particular on correspondence rules and ordering semantics.

Reordering constitutes change only if order is declared semantically relevant under $\mathcal{R}$. Otherwise, order variation may be normalized during canonicalization.

The preceding categories identify possible classes of structural difference but do not constitute a complete, representation-independent edit algebra. Their reliable detection depends on additional assumptions supplied by the selected representation model.

A modification can be localized directly when the predecessor and successor components are connected by a valid correspondence relation. An insertion or deletion can be identified only when the representation model permits the presence or absence of a

component to be determined unambiguously. Movement requires persistent component identity across different structural positions. Reordering additionally requires explicit order semantics, because a change in position is relevant only where sibling order forms part of the canonical representation.

Cross-schema comparison introduces a further requirement. If two states were generated under different schema versions or structurally incompatible representation rules, they are not directly comparable unless an explicit compatibility relation, migration function, or schema-alignment mapping has first transformed them into a shared comparison space. The framework does not itself provide such migration or alignment procedures.

Consequently, the present model identifies the representational conditions under which these structural differences can be expressed and localized, but it does not provide a universal algorithm for resolving arbitrary insertions, deletions, movements, reorderings, or schema transformations.

The notion of change is complementary to the state-equivalence relation introduced in Section 3.1. On this basis, representation-relative invariance characterizes transformations that preserve canonical state equivalence.

Proposition 13 (Representation-Relative Invariance)**.** *Let $\tau : \mathcal{S}_{\mathcal{R}} \to \mathcal{S}_{\mathcal{R}}$ be a state transformation, i.e., a mapping that modifies a document state while preserving membership in the state space. Then τ is irrelevant under $\mathcal{R}$ if and only if it preserves canonical serialization for every admissible state $s \in \mathcal{S}_{\mathcal{R}}$:*

$$\mathrm{ser}(\tau(s)) = \mathrm{ser}(s).$$

Equivalently,

$$\tau(s) \equiv s \qquad \textit{for all } s \in \mathcal{S}_{\mathcal{R}}.$$

Proof. If τ is irrelevant under $\mathcal{R}$, then by definition it does not introduce any difference recognized by the representation model. Hence, the canonical representation remains unchanged, so $\mathrm{ser}(\tau(s)) = \mathrm{ser}(s)$ for all s.

Conversely, if $\mathrm{ser}(\tau(s)) = \mathrm{ser}(s)$ for all s, then $\tau(s)$ and s belong to the same equivalence class induced by canonical serialization. Therefore, the transformation cannot be detected as change under $\mathcal{R}$ and is thus irrelevant. □

Typical examples of such transformations include formatting adjustments, whitespace normalization, permutation of order-insensitive components, or metadata changes explicitly excluded by the representation model. Canonicalization therefore determines the mathematical boundary between relevant and irrelevant variation. It is part of the representation model itself rather than a mere preprocessing step.

Change is evaluated relative to the contextually valid predecessor rather than arbitrary historical states. Thus, change is modeled as a local relation on the evolution graph, not as an unrestricted comparison across the full history.

3.3.6. Detection Versus Interpretation

A central conceptual distinction of the proposed framework is the separation between computational detection and contextual interpretation. Structural change is determined within the representation model and can therefore be identified deterministically by comparison of canonical states. By contrast, the significance of a detected change is not encoded in the representation itself, but depends on external criteria such as safety relevance, business priority, regulatory constraints, user role, or operational context.

Definition 19 (Interpretation Function). *Let $\Delta(s_a, s_b)$ denote the detected structural change between two comparable states. An interpretation function is a mapping $\mathcal{I} : \mathcal{C}_\Delta \times \mathcal{C} \to \mathcal{Q}$, where $\mathcal{C}$ denotes a set of external contexts (e.g., organizational, regulatory, operational, or user-specific conditions) and $\mathcal{Q}$ a set of relevance assessments, priorities, or actions.*

Proposition 14 (Detection–Interpretation Separation). *Let $\mathcal{R}$ be fixed. Then structural change detection is determined solely by the representation model, whereas change interpretation additionally depends on external context.*

Proof. Let $s_a, s_b \in \mathcal{S}_\mathcal{R}$ be comparable states. Since canonicalization and serialization are deterministic under $\mathcal{R}$, the truth value of $\mathrm{ser}(s_a) \neq \mathrm{ser}(s_b)$ is uniquely determined by $\mathcal{R}$, s_a, and s_b. Hence the detected change set $\Delta(s_a, s_b)$ is a function of the representation model alone.

Now consider interpretation. By definition, interpretation is given by $\mathcal{I}\big(\Delta(s_a, s_b), c\big)$ with $c \in \mathcal{C}$. For two distinct contexts $c_1, c_2 \in \mathcal{C}$, the same structural change may yield different outcomes:

$$\mathcal{I}\big(\Delta(s_a, s_b), c_1\big) \neq \mathcal{I}\big(\Delta(s_a, s_b), c_2\big).$$

Therefore interpretation cannot, in general, be derived from representation alone and requires additional contextual input. □

Hence, a technical system may reliably determine whether change exists and where it is localized, while the practical meaning of that change requires a separate interpretive layer.

Although the interpretation function is formally separated from deterministic change detection, its operational realization is straightforward within practical information systems. After the structural difference set has been determined, application-specific rules may classify each detected modification according to document semantics, organizational policies, or user roles. For example, a modified safety instruction may trigger mandatory operator notification, a changed process parameter may require engineering approval, whereas formatting-only modifications eliminated during canonicalization generate no operational action. The interpretation stage therefore operates on the deterministic output of the comparison process without influencing the underlying representation or equality relations. This separation enables identical document states to be interpreted differently across application domains while preserving reproducible state comparison.

3.3.7. Implications for Efficient Comparison

The formalization of change as a relation between structured states has direct computational implications.

Proposition 15 (Representation-Induced Comparison Efficiency). *Let $s_a, s_b \in \mathcal{S}_\mathcal{R}$ be comparable states. If canonical representations or fixed-size derived summaries are already available, the whole-state equality decision can be performed without renewed traversal of the underlying raw artifacts. If hierarchical structure is available and inequality is detected, further comparison may be restricted to affected substructures.*

Proof. Whole-state equality is determined by the canonical representations or derived summaries associated with s_a and s_b. Hence the decision stage operates on already constructed comparison objects rather than on the original raw artifacts. If the states are hierarchically decomposed, equality can first be tested at coarse granularity. Only differing components or subtrees require recursive refinement, whereas unchanged substructures need not be re-evaluated. □

Thus, computational effort is not eliminated, but reorganized. It is shifted from repeated exhaustive comparison toward one-time state construction, canonicalization, reusable identifiers or digests, and selective refinement of changed regions. Efficiency therefore arises primarily from representational design rather than from algorithmic optimization alone.

3.3.8. Summary

The change model completes the formal framework by defining change as a deterministic relation between temporally adjacent canonical document states. Whole-state inequality is refined hierarchically into localized structural differences, whereas contextual interpretation is treated as a separate layer operating on deterministic comparison results. Because change is defined relative to the contextually valid predecessor, detected differences correspond to coherent evolution steps rather than arbitrary historical discrepancies. Together with the representation and temporal models developed previously, the resulting framework provides a mathematically consistent basis for deterministic, scalable, and interpretable document comparison.

3.4. Computational Instantiation: Hierarchical Digest-Based Comparison

The framework proposed in this paper is intentionally independent of any specific comparison algorithm, digest function, or software architecture. Its purpose is to establish the representational, structural, and temporal conditions under which document change can be detected consistently and interpreted meaningfully.

This subsection presents one computational instantiation of the proposed framework based on hierarchical digest comparison. The instantiation is selected because it directly realizes the central representational principles developed in the preceding sections: deterministic canonical state construction, predecessor-constrained comparison, and hierarchical refinement of structural differences.

The objective is not to introduce a new hashing method or comparison algorithm. Rather, the purpose is to demonstrate that the abstract state-centric representation model admits a concrete and executable realization. The operational behavior of this realization is examined separately in the reproducible case study presented in Section 5, whereas the present subsection focuses exclusively on the computational structure of the instantiation itself.

A related hierarchical digest architecture has previously been applied in a manufacturing-specific environment [30]. That work demonstrates one domain-specific realization. In contrast, the present paper develops the general representation framework from which such implementations can be derived and specifies the computational principles independently of any particular application domain.

3.4.1. Canonical Representation and Digest Construction

The digest-based realization begins with the canonical document state introduced earlier. Rather than comparing raw external artifacts directly, each document instance is first transformed into a deterministic internal representation and only thereafter summarized by digest values. This separation is essential: digest comparison is reliable only when semantically equivalent inputs are mapped to identical canonical states.

1. **Construction of Canonical Document State.** Let $d_t \in \mathcal{D}$ denote a document instance observed at logical time t. Its internal state representation is obtained through the representation mapping $s_t = \Phi(d_t)$. This step is essential because raw external artifacts may differ in syntax while expressing the same content. For example, differences in formatting, attribute order, whitespace, encoding, or null-value notation should not be interpreted as genuine change. Under the fixed representation model $\mathcal{R}$, the mapping

Φ is therefore assumed deterministic, representation-invariant, structure-preserving. Hence, digest construction is performed on stable internal states rather than unstable raw artifacts.

2. **Canonical Serialization of the State.** To obtain a machine-processable representation, the canonical state is serialized by a deterministic mapping $\mathrm{ser}(\cdot)$. Thus, $\mathrm{ser}(s_t)$ denotes a unique reproducible encoding of s_t. This step is necessary because digest functions operate on byte strings or symbol sequences rather than directly on abstract state objects.
3. **Construction of a State-Level Digest.** Let $\mathrm{h}(\cdot)$ denote a collision-resistant digest function. A digest of the complete document state is then defined by

$$h_t^{\mathrm{doc}} = \mathrm{h}\big(\mathrm{ser}(s_t)\big).$$

The purpose of h_t^{doc} is to provide a fixed-size summary of the full state. Instead of repeatedly comparing entire structured objects, one may first compare their digests. If two state digests differ, the states necessarily differ structurally. If the digests are equal, the states are treated as equal with high practical confidence, subject to the collision properties of the chosen hash function.

4. **Construction of Component-Level Digests.** If the state is hierarchically decomposed as

$$s_t = \big(u_t^{(1)}, u_t^{(2)}, \ldots, u_t^{(n_t)}\big),$$

then additional digests may be assigned to individual components:

$$h_t^{(i)} = \mathrm{h}\big(\mathrm{ser}(u_t^{(i)})\big), \qquad i = 1, \ldots, n_t.$$

This serves a different purpose than the global digest. Whereas h_t^{doc} supports rapid whole-state decisions, the component digests support localization. Once a global mismatch has been detected, refinement can be restricted to components whose digests differ.

5. **Preservation of Hierarchy through Aggregation.** Component digests may be combined recursively into parent-level digests, for example,

$$h_t^{\mathrm{doc}} = \mathrm{h}\Big(h_t^{(1)} \parallel h_t^{(2)} \parallel \cdots \parallel h_t^{(n_t)}\Big).$$

This mirrors the hierarchical tree structure of the document state. As a consequence, comparison can proceed top-down: first at document level, then only within affected branches, and finally within changed descendants if necessary.

The preceding construction clarifies why digest-based comparison can be efficient: Expensive parsing and canonicalization are performed once per state, unchanged states can be tested through compact summaries, mismatch triggers refinement only where needed, hierarchical locality avoids exhaustive traversal of unchanged regions. Thus, efficiency does not arise from hashing alone. It arises from combining hashing with canonical state construction and explicit hierarchical structure.

The precise digest function, aggregation scheme, storage strategy, and system architecture are implementation choices. They may vary across domains and applications.

Accordingly, hierarchical digests are not the theoretical contribution of this paper. They constitute one computational instantiation of the proposed representation framework. Their role is to demonstrate that the formal concepts developed in the preceding sections can be realized within an executable comparison workflow. The operational characteristics of this realization are evaluated separately through the reproducible case study in Section 5.

3.4.2. Comparison Workflow as an Instance of the Change Model

Let s_t denote the current document state and let $\mathrm{pred}(s_t) = s_{t^-}$ be its contextually valid predecessor. In accordance with the formal change model introduced in Section 3.3, structural change is evaluated relative to this predecessor pair. In the present computational instantiation, the same principle is realized through hierarchical digest comparison.

A digest h_t^{doc} is assigned to the canonical representation of the complete state, while subordinate digests $h_t^{(i)}$ summarize individual components or substructures. Comparison then proceeds in a staged top-down manner. First, equality is tested at the whole-state level by comparing the document digests. Only if inequality is detected is the decision refined to component level and, if required, recursively to deeper structural levels.

Hence, the digest-based workflow operationalizes the refinement logic of the proposed change model: global state equality terminates the procedure immediately, whereas global inequality triggers localized inspection of affected structural regions.

Algorithm 1 summarizes the operational workflow. Equality is first evaluated at the coarsest available structural level. Additional computational effort is incurred only when mismatch necessitates refinement within affected branches of the hierarchy.

Algorithm 1 Hierarchical digest comparison as an instance of the change model.

Require: Current state s_t
Require: Contextually valid predecessor $s_{t^-} = \mathrm{pred}(s_t)$
Require: Document-level digests h_t^{doc}, $h_{t^-}^{\mathrm{doc}}$
Require: Component-level digests $h_t^{(i)}$, $h_{t^-}^{(i)}$
Ensure: Change decision and localized change set $\mathcal{C}_\Delta$

Initialization
$\mathcal{C}_\Delta \leftarrow \varnothing$

Whole-state equality check
if $h_t^{\mathrm{doc}} = h_{t^-}^{\mathrm{doc}}$ **then**
  return UNCHANGED, $\mathcal{C}_\Delta$
end if

Hierarchical refinement
for each corresponding component i of s_t and s_{t^-} **do**
  if $h_t^{(i)} \neq h_{t^-}^{(i)}$ **then**
    $\mathcal{C}_\Delta \leftarrow \mathcal{C}_\Delta \cup \{i\}$
    Recursively refine comparison within component i
  end if
end for

Change result
return CHANGED, $\mathcal{C}_\Delta$

The presented workflow specifies one computational instantiation of the proposed representation framework. Alternative realizations may employ different digest functions, serialization procedures, storage architectures, or correspondence mechanisms while preserving the same representational assumptions. The purpose of this workflow is therefore to establish computational realizability rather than to identify an optimal implementation strategy.

Since the proposed workflow compares digest values instead of repeatedly comparing full canonical state representations, a correctness argument is required. Proposition 16 establishes that this substitution is justified under standard assumptions on deterministic serialization and collision-resistant hashing.

Proposition 16 (Digest Soundness)**.** *Assume that* $\mathrm{ser}(\cdot)$ *is deterministic and that* $\mathrm{h}(\cdot)$ *is collision-resistant. Then, equality of digest values implies equality of the corresponding canonical representations, except with negligible collision probability.*

Proof. If two canonical representations are identical, deterministic hashing yields identical digest values. Conversely, equal digest values for distinct representations imply a collision of $\mathrm{h}(\cdot)$. Under collision resistance, the probability of such an event is negligible for practical purposes. □

The principal computational benefit of this workflow lies in the reorganization of effort. Expensive operations such as parsing, canonicalization, and digest generation are performed once during state construction. Subsequent repeated comparisons can then be executed through compact fixed-size summaries rather than through renewed traversal of full raw document artifacts. Only when inequality is detected must additional work be invested in localized structural refinement.

Accordingly, the computational characteristics of the workflow emerge from the interaction between canonical representation, hierarchical decomposition, and predecessor-based comparison rather than from hashing alone. The reproducible operational case study presented later demonstrates these principles within a controlled execution scenario, whereas broader questions concerning runtime efficiency, scalability, robustness, and implementation trade-offs remain application-dependent. A domain-specific realization and performance evaluation are reported separately in [30].

3.4.3. Design Space, Scope, and Limitations

Hierarchical digest comparison represents only one realization within a broader design space compatible with the framework. Other realizations may include structural fingerprints, incremental update propagation, tree differencing procedures, or semantic comparison mechanisms enriched by domain knowledge.

The framework does not depend on hashing as such. It specifies the structural conditions under which many comparison mechanisms can operate consistently and efficiently.

The present computational instantiation focuses on structural equivalence under a fixed representation model. It does not by itself resolve semantic equivalence beyond the chosen canonicalization, nor does it determine the practical relevance of detected changes. Moreover, digest equality is probabilistic evidence of representation equality rather than a formal proof, since it depends on the collision properties of the chosen hash function.

In summary, the hierarchical digest-based computational instantiation demonstrates that the proposed state-centric representation framework can be translated into an executable comparison workflow while preserving its formal principles. Deterministic representation, hierarchical organization, and predecessor-based comparison jointly enable localized change analysis within the stated representational assumptions. The reproducible operational case study in the following section complements this computational specification by evaluating the behavior of one reference implementation under controlled and independently reproducible conditions. Together, these complementary sections distinguish the conceptual framework, its computational instantiation, and its operational realization.

3.5. Complexity and Scalability Considerations

The preceding sections focused primarily on the representational and structural foundations required for deterministic comparison of evolving document states. Beyond conceptual correctness, however, practical applicability also depends on scalability under repeated comparison workloads. This is particularly relevant for large document repositories, continuous synchronization scenarios, and change-aware information systems in

which documents are evaluated repeatedly over time. The following discussion is analytical in nature and characterizes the expected computational behavior implied by the representation model. It does not constitute an empirical benchmark, which would require systematic evaluation across representative datasets, implementations, and execution environments.

A central distinction of the proposed framework is that comparison efficiency emerges primarily from representational structure rather than from isolated differencing procedures alone. In conventional flat comparison approaches, equality verification typically requires traversal of the complete representation regardless of whether changes are present. By contrast, hierarchical decomposition combined with canonical subtree representations permits selective refinement and recursive pruning of unchanged regions.

Table 4 summarizes the qualitative complexity characteristics of conventional full traversal and hierarchical digest-based comparison.

Table 4. Qualitative comparison of computational characteristics for conventional flat traversal and hierarchical digest-based comparison. The assessment column summarizes the relative suitability of each strategy for repeated comparison scenarios. n denotes the total document size, m the size of the affected subtree, and k the number of repeated comparison operations. Suitability indicates qualitative fit for repeated comparison scenarios. Suitability indicates qualitative fit for repeated comparison scenarios (■■■: High, ■□□: Neutral)

Comparison Strategy	Equality Check	Localization Effort	Repeated Comparison Cost	Suitability
Flat full traversal	$O(n)$	$O(n)$	$O(kn)$ repeated full traversal	■□□
Hierarchical digest comparison	$O(1)$ root digest check + selective refinement	$O(\log n)$ to $O(m)$	Reduced traversal overhead through localized refinement	■■■

The complexity statements in Table 4 describe individual operational stages under explicit assumptions; they do not represent end-to-end complexity for ingesting and processing a previously unseen document. In particular, initial parsing, canonicalization, state construction, component-correspondence establishment, digest computation, repeated equality checking, hierarchical localization, and contextual interpretation must be distinguished.

A constant-time root-digest equality check applies only after two comparable states have already been constructed, their required correspondence assumptions have been satisfied, and their root digests have been computed and stored. It therefore excludes parsing, canonicalization, state construction, correspondence establishment, and digest computation. Similarly, localization complexity depends on hierarchy depth, branching structure, update locality, the number and distribution of changed components, correspondence quality, and the type of structural modification. The reported bounds should therefore be interpreted as conceptual operational characteristics under the stated assumptions rather than universal complexity guarantees.

Thus, the complexity characteristics summarized in Table 4 describe the expected behavior under the assumptions of deterministic canonical representations and hierarchical document structures. The exact computational cost depends on implementation-specific design decisions, including serialization strategy, storage organization, digest maintenance, and structural balance of the document hierarchy. In repeated comparison scenarios, the one-time preprocessing required for canonical state construction can be amortized across multiple subsequent comparison operations, whereas highly unbalanced hierarchies may reduce the effectiveness of hierarchical pruning. Consequently, the reported complexity ranges should be interpreted as conceptual characteristics of the framework rather than implementation-independent asymptotic guarantees.

The principal scalability advantage of the proposed framework does not arise solely from digest computation itself, but from the interaction of canonical representation, hierarchical decomposition, and predecessor-constrained comparison scope. In repeated comparison scenarios, unchanged subtrees can be excluded recursively from further evaluation once subtree equivalence has been established. Consequently, comparison effort scales primarily with the structurally affected regions rather than with the complete document size.

The complexity values shown in Table 4 are intentionally qualitative and implementation-independent. Actual runtime behavior depends on factors such as tree depth, branching structure, serialization overhead, caching strategy, and digest maintenance policy. Nevertheless, the analysis illustrates an important principle of the proposed framework: scalability is achieved primarily through structured representation and localized refinement rather than through repeated exhaustive traversal of complete document artifacts.

3.6. Summary

Table 5 summarizes the central elements of the proposed state-centric framework and their respective roles within change-aware information systems. The overview consolidates the methodological architecture developed throughout this section by showing how deterministic representation, hierarchical organization, temporal linkage, comparison logic, and contextual interpretation interact as complementary layers of the overall model. Together, these elements establish the representational and structural foundation on which the subsequent validation and operational analysis are based.

Table 5 summarizes the analytical validation of the proposed framework with respect to the design objectives introduced in Section 1. The assessment evaluates whether the framework satisfies its intended conceptual requirements. It should therefore be interpreted as a structured design validation rather than as an empirical comparison against competing implementations.

Table 5. Core elements of the proposed state-centric framework and their respective functions within change-aware information systems. The table summarizes how representation, hierarchical organization, temporal linkage, comparison logic, and contextual interpretation jointly contribute to scalable and interpretable change handling.

Framework Element	Primary Function	Role in Change Handling
Canonical representation	Transforms heterogeneous external artifacts into stable internal document states	Ensures deterministic comparability independent of incidental formatting, ordering, or encoding differences
Hierarchical decomposition	Organizes document states into nested structural components	Supports localized refinement and structural localization of detected changes
Predecessor linkage	Defines explicit temporal relations between successive document states	Constrains comparison with coherent evolution paths and improves traceability
State comparison	Evaluates equivalence and divergence between comparable states	Provides the formal basis for deterministic change detection and transition analysis
Interpretation layer	Associates detected differences with contextual relevance	Separates technical detection from domain-specific meaning and operational significance

The analytical validation presented in Table 5 evaluates the internal consistency of the proposed framework and its ability to satisfy the intended representational objectives.

It does not replace empirical benchmarking, which addresses a complementary research question concerning implementation efficiency under practical operating conditions.

Figure 6 illustrates the operational flow of the proposed framework from external document ingestion to context-dependent interpretation. The figure emphasizes that comparison is embedded within a broader representation-centric process in which canonicalization, hierarchical state construction, temporal linkage, and localized refinement jointly enable scalable and interpretable change handling.

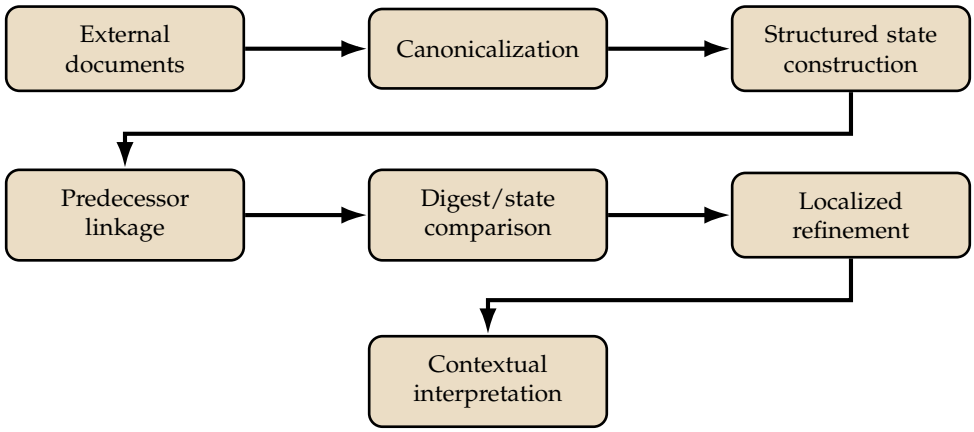


Figure 6. Operational flow of the proposed state-centric framework. External document artifacts are transformed into canonical structured states, linked to predecessor versions, compared through digest- or state-based mechanisms, refined locally in the case of detected differences, and finally interpreted with respect to contextual relevance.

4. Analytical Assessment

4.1. Validation Objective and Scope

The assessment strategy follows directly from the nature of the contribution. The proposed work develops a conceptual and formal representation framework for modeling evolving documents as comparable information states. Consequently, its primary objective is not the empirical evaluation of a specific implementation or comparison algorithm, but the analytical examination of whether the proposed representational principles establish a coherent and sufficiently precise foundation for change-aware information systems.

Accordingly, the validation strategy adopted in this paper consists of two complementary but clearly distinguished layers.

The first layer, presented in this section, is an analytical assessment. It examines whether the framework is internally consistent, whether its formal definitions and assumptions form a coherent representation model, whether the resulting state-transition model supports deterministic comparison under the stated assumptions, and whether the framework provides a logically complete basis for localized structural change analysis. This assessment therefore establishes conceptual validity rather than implementation performance.

The second layer is provided separately in Section 5 through a fully reproducible applied case study. Unlike the present analytical assessment, the case study demonstrates how the proposed representation model can be instantiated in an executable information system setting using an openly specified document collection, deterministic processing pipeline, reference implementation, predefined ground truth, and reproducible evaluation procedure. Its purpose is to demonstrate operational realizability and reproducibility rather than empirical superiority over existing approaches.

This separation is intentional. Analytical consistency and operational reproducibility address different research questions. The analytical assessment establishes whether the framework constitutes a coherent conceptual model. The reproducible case study

subsequently demonstrates that this model can be realized in practice under controlled conditions. Neither component is intended to replace a comprehensive benchmark study, statistical evaluation, or large-scale performance analysis.

Consequently, the present section does not provide runtime measurements, memory consumption analyses, scalability curves, statistical hypothesis testing, or comparative benchmarking against existing implementations such as Git-based differencing, XML- or JSON-specific comparison methods, Merkle-tree implementations, database-native change tracking, or event-sourcing frameworks. Likewise, no claim of computational superiority, implementation efficiency, or general operational robustness is made. Such properties remain implementation-dependent and require separate empirical investigation under representative workloads.

Within this scope, the proposed framework is considered analytically supported if it establishes:

1. A deterministic representation model for constructing canonical document states from heterogeneous external artifacts;
2. A formally defined representation of document evolution through explicit predecessor–successor relations;
3. A logically consistent mechanism for deterministic comparison and hierarchical localization of structural differences between comparable states; and
4. A coherent separation between representation, comparison, and application-specific interpretation of detected changes.

The remainder of this section therefore evaluates the proposed framework exclusively from an analytical perspective. The assessment focuses on the internal consistency of the representation model, the correspondence between its formal constructs and the stated design objectives, and the computational realizability of one implementation-independent instantiation. The subsequent section complements this analytical assessment with a reproducible operational case study that evaluates one executable realization under controlled conditions.

4.2. Validation Against Design Objectives

The preceding sections developed the proposed framework incrementally by introducing its representation model, temporal state concept, formal comparison model, and computational realization. The analytical question addressed here is whether these individual components collectively satisfy the design objectives formulated in Section 1. Rather than introducing additional concepts, this subsection synthesizes the framework and evaluates its internal coherence with respect to these objectives.

The assessment is conducted through explicit traceability between the intended system capabilities and the corresponding formal constructs established throughout the paper. This traceability serves two purposes. First, it demonstrates that each design objective is supported by clearly defined representation-theoretic concepts rather than informal assumptions. Second, it shows that the individual framework components form a logically consistent whole, in which each construct contributes to a well-defined aspect of change-aware document management.

Table 6 summarizes this analytical assessment. For every design objective, it identifies the formal basis within the framework together with the corresponding representational capability that follows from the proposed definitions. Consequently, the table should not be interpreted as an empirical evaluation but as an analytical synthesis demonstrating that the framework provides a coherent conceptual foundation for deterministic representation, temporal document evolution, hierarchical change localization, and application-independent state comparison.

The operational realization of these concepts is intentionally treated separately in Section 5, where the complete processing pipeline is instantiated and evaluated under fully reproducible conditions. The present subsection therefore focuses exclusively on analytical consistency between the research objectives and the proposed representation model.

Table 6. Analytical traceability between the research objectives and the formal constructs of the proposed framework. For each objective, the table identifies the corresponding representation-theoretic foundation and the resulting conceptual capability. The mapping demonstrates that all stated objectives are supported by explicit formal constructs, thereby establishing the analytical completeness and internal consistency of the proposed representation model.

Design Objective	Formal Basis	Capability
Deterministic representation	Representation model, mapping Φ, serialization $\mathrm{ser}(\cdot)$	Equivalent admissible inputs are transformed into stable, reproducible document states
Meaningful comparison	Comparability relation $\sim$ and state equivalence $\equiv$	Only states generated under a shared model are compared under well-defined equality rules
Localized change detection	Hierarchical state structure and change set $\Delta(\cdot,\cdot)$	Detected differences can be attributed to specific components or substructures
Temporal consistency	Predecessor mapping $\mathrm{pred}(\cdot)$	Comparison is restricted to coherent evolution steps rather than arbitrary historical pairs
Operational scalability	Digest-based realization and staged refinement	Repeated equality checks can be executed efficiently, with deeper analysis only where required

The traceability analysis demonstrates that each research objective is addressed through explicit formal definitions rather than implementation-specific assumptions. Deterministic representation is achieved through canonical state construction, meaningful comparison is restricted to formally comparable states, hierarchical localization follows from the recursive document representation, and temporal consistency is established through explicit predecessor–successor relationships. Collectively, these constructs form a logically coherent representation framework in which the individual components reinforce one another without introducing conflicting assumptions or hidden dependencies.

This analytical synthesis establishes that the proposed framework satisfies the conceptual requirements formulated for change-aware document management. It does not, however, constitute evidence of implementation performance or operational effectiveness. Those aspects are examined separately through the reproducible applied case study presented in Section 5, which demonstrates that the analytical concepts developed here can be instantiated in a transparent, executable, and independently reproducible workflow.

4.3. Illustrative Walkthrough of the Representation Model

The preceding analytical assessment established that the proposed framework provides a coherent representation model for deterministic document-state construction, temporal evolution, and hierarchical comparison. To complement this conceptual assessment, the following walkthrough illustrates how these formal concepts interact during a representative state transition. The objective is not to evaluate implementation performance, but to demonstrate the logical behavior of the representation model under a simple yet representative evolution scenario.

The example deliberately considers only the conceptual processing steps defined by the framework. It illustrates how canonical document states are compared, how structural

differences are localized, and how the hierarchical representation supports progressively refined analysis. Consequently, the walkthrough serves as an explanatory example that bridges the formal definitions introduced in the previous sections with the executable realization presented later in the reproducible applied case study (Section 5).

Consider two successive document states generated under an identical representation model:

$$s_{t-1} = \left(c, u^{(1)}, u^{(2)}, u^{(3)}\right), \qquad s_t = \left(c, u^{(1)}, \tilde{u}^{(2)}, u^{(3)}\right),$$

where c denotes contextual metadata common to both document states, while $\tilde{u}^{(2)} \not\equiv u^{(2)}$ indicates that exactly one structural component has changed between the two successive versions.

This situation represents one of the most common evolution scenarios encountered in practice: the overall document identity is preserved while only a localized part of the document is modified. The example therefore illustrates the essential reasoning process performed by the proposed framework without introducing implementation-specific assumptions. The comparison proceeds hierarchically. The framework first evaluates whether the two canonical document states remain globally equivalent. Equality is determined exclusively from their canonical representations and is therefore independent of the original source format or document serialization.

$$\mathrm{ser}(s_{t-1}) \neq \mathrm{ser}(s_t).$$

The differing canonical representations immediately establish that the two document states are no longer equivalent. At this stage, however, only the existence of a structural difference has been established. Neither its location nor its semantic relevance has yet been determined. The hierarchical representation therefore enables the comparison process to continue selectively toward progressively finer structural levels.

$$\mathrm{ser}(u_{t-1}^{(1)}) = \mathrm{ser}(u_t^{(1)}), \qquad \mathrm{ser}(u_{t-1}^{(2)}) \neq \mathrm{ser}(u_t^{(2)}), \qquad \mathrm{ser}(u_{t-1}^{(3)}) = \mathrm{ser}(u_t^{(3)}).$$

Therefore,

$$\Delta(s_{t-1}, s_t) = \{2\}.$$

The two states differ globally, but the detected modification is localized exclusively to the second component. If $u^{(2)}$ itself contains nested substructures, the same comparison logic can be applied recursively at deeper levels.

Figure 5 visualizes the transition from whole-state detection to component-level localization and subsequent hierarchical drill-down.

This example highlights a central advantage of the framework: global inequality does not imply that every part of the document must be re-evaluated. Once unchanged regions are verified as equivalent, further analysis can be concentrated on the affected branch only.

A practical interpretation of the example can be given by considering a structured manufacturing instruction. Let $u^{(1)}$ represent general document metadata, $u^{(2)}$ a process instruction block, and $u^{(3)}$ an inspection or approval block. A modification of a torque value, safety instruction, or parameter table inside $u^{(2)}$ changes the corresponding canonical subtree while leaving the remaining components unchanged. A flat file comparison would identify the two document artifacts as different, but would not by itself provide a stable representation-level explanation of where the change is located. In contrast, the proposed framework first detects inequality at the state level and then localizes the difference to the affected process-instruction component and, if necessary, to a subordinate parameter or safety node.

The comparison with existing approaches is therefore conceptual rather than benchmark-based. Conventional file differencing emphasizes textual or serialized differ-

ences, version control systems emphasize historical revision management, and Merkle-type structures emphasize efficient digest verification. The proposed framework integrates these capabilities at the representation level by requiring canonical state construction, predecessor-constrained comparison, hierarchical localization, and separate contextual interpretation. Consequently, the contribution of the present work lies in making these representational dependencies explicit within a unified document-state framework rather than in replacing established differencing or version control mechanisms.

The walkthrough is intentionally compact and serves to expose the logical sequence from whole-state comparison to component-level localization and recursive refinement under complete and unambiguous component correspondence. It is not an empirical robustness study and does not establish behavior under large-scale workloads, partial or ambiguous correspondence, insertion or deletion of unmatched components, component movement, order-sensitive restructuring, schema evolution, or multiple-parent merge scenarios. These cases require additional representation-specific correspondence, migration, ordering, or version-graph semantics and should be examined through dedicated implementation and empirical studies.

The walkthrough highlights an important conceptual property of the proposed framework. Global comparison, hierarchical localization, and temporal consistency are not independent processing steps but direct consequences of the underlying representation model. Once canonical document states have been constructed, each subsequent comparison operation follows deterministically from the formal definitions introduced in Sections 3 and 3.4. Consequently, the behavior illustrated here is not specific to the chosen example but reflects the general analytical properties of the framework.

The walkthrough intentionally remains at the representation level. Its purpose is to explain how the framework operates conceptually rather than to evaluate implementation characteristics. The practical realization of this analytical workflow, including executable processing, reference implementation, predefined ground truth, and reproducible evaluation protocol, is presented separately in the reproducible applied case study in Section 5.

4.4. Computational Instantiation of the Representation Model

The representation framework developed in this paper is intentionally independent of any particular implementation strategy. Its formal definitions specify how canonical document states are constructed, related, and compared, without prescribing concrete data structures or algorithms. Nevertheless, a conceptual framework intended for practical information systems should admit computational realizations that preserve these formal properties. This subsection therefore examines one representative instantiation introduced in Section 3.4.

The digest-based realization is not presented as an alternative model, nor as an independent contribution. Rather, it serves as a constructive instantiation demonstrating that the abstract representation model can be mapped consistently onto an executable computational mechanism. Consequently, the purpose of this subsection is analytical: it establishes that the formal concepts introduced throughout the framework are computationally realizable while preserving their representation-theoretic semantics.

More specifically, the correspondence between the formal representation model and the digest-based realization can be summarized as follows:

1. Canonical document states are transformed into deterministic digest representations, allowing state equivalence to be evaluated without repeated inspection of the complete document structure.

2. Whenever two corresponding digest values differ, the comparison process is refined hierarchically to the associated structural components, thereby preserving the recursive localization mechanism defined by the representation model.
3. Recursive traversal is restricted to structurally affected branches, while unchanged substructures remain analytically equivalent and therefore require no further examination.
4. Explicit predecessor–successor relationships determine the valid temporal reference state for every comparison, thereby preserving the formal evolution model introduced in Section 3.

Importantly, these observations do not constitute empirical performance claims. The digest-based realization merely demonstrates that the formal representation model admits a computational instantiation whose individual processing steps correspond directly to the underlying analytical definitions. The specific implementation technology, hashing algorithm, storage architecture, execution environment, or optimization strategy may vary without changing the conceptual behavior of the framework, provided that the defining representation properties remain preserved.

Consequently, the digest instance should be understood as an existence proof for computational realizability rather than as evidence of implementation superiority. It illustrates that the concepts of canonical representation, deterministic comparison, temporal state evolution, and hierarchical localization can be translated into a coherent executable workflow without modifying the underlying analytical model.

The practical realization of this computational workflow is examined separately in the reproducible operational case study presented in Section 5. There, the complete processing pipeline is instantiated under transparent and independently reproducible conditions using an openly specified document collection, reference implementation, predefined ground truth, and reproducible evaluation procedure. Accordingly, the present subsection establishes the analytical correspondence between the formal model and one computational realization, whereas the subsequent section evaluates that realization under controlled operational conditions.

4.5. Scope of the Analytical Assessment

The analytical assessment presented in this section is intentionally bounded by the conceptual nature of the proposed contribution. It examines whether the definitions, assumptions, and representational constructs of the framework form a logically coherent model for representing and comparing evolving documents. Accordingly, the assessment addresses canonical state construction, explicit comparability, predecessor-constrained evolution, hierarchical localization, and the separation of structural detection from application-specific interpretation.

The assessment does not establish implementation-independent performance characteristics. In particular, it does not provide evidence of general runtime superiority, memory efficiency, scalability, robustness, or cross-domain effectiveness. Such properties depend on concrete implementation decisions, document structures, correspondence mechanisms, storage architectures, execution environments, and workload characteristics. They therefore require dedicated empirical studies of specific implementations rather than analytical examination of the representation framework alone.

The digest-based instantiation discussed in the preceding subsection establishes computational plausibility by showing that the formal constructs can be mapped onto an executable comparison mechanism. It should not, however, be interpreted as an empirical evaluation of practical performance. Analytical coherence and operational behavior constitute distinct evidential questions and are therefore examined separately in this paper.

Section 5 addresses the second of these questions through a controlled and reproducible operational case study. It evaluates one concrete implementation using successive structured-document states, an explicitly defined ground truth, a reference implementation, a textual-diff baseline, and a boundary case involving ambiguous component correspondence. The case study therefore complements the present analytical assessment by examining whether the formal concepts can be instantiated and observed in an executable workflow under transparent and reproducible conditions.

The subsequent operational case study therefore should not be interpreted as an extension of the analytical assessment but as a complementary form of evidence. Whereas the present section establishes the conceptual coherence of the proposed representation model, the following section demonstrates that these concepts can be instantiated within an executable and independently reproducible document-processing workflow.

5. Reproducible Operational Case Study

The analytical assessment established that the proposed framework constitutes a coherent representation model for deterministic document comparison under explicitly stated assumptions. The purpose of the present section is different. It examines whether these formal concepts can be instantiated within an executable document-processing workflow and whether the resulting implementation exhibits the expected operational properties under controlled and independently reproducible conditions.

The case study uses a structured technical maintenance document represented through five successive JSON states. Its purpose is to determine whether the formal concepts introduced in Section 3 can be translated into an executable workflow and whether the resulting implementation exhibits the expected properties of deterministic representation, controlled change detection, structural localization, hierarchical refinement, and explicit component correspondence.

The evaluation is deliberately bounded. It examines the behavior of one reference implementation against a predefined ground truth and includes an explicit schema-correspondence map, a conventional unified textual-diff baseline, and a boundary case involving ambiguous component identity. It is not intended to establish general computational superiority, application-independent scalability, production-level robustness, or implementation-independent performance.

All input documents, source code, expected change events, automated tests, and generated result files as Supplementary Materials and are openly available through Zenodo [31]. The implementation uses only modules from the Python standard library and can be executed without external dependencies under Linux-compatible environments using Python 3.10 or later.

5.1. Case-Study Objective and Evaluation Questions

The applied evaluation addresses a narrower question than a comparative performance benchmark. It examines whether the formal concepts introduced in the framework can be translated into a reproducible document-processing workflow whose outputs can be checked against an explicitly defined ground truth. The case study therefore evaluates observable properties of one concrete implementation rather than the general performance of all possible implementations derived from the framework.

Four evaluation questions were defined.

EQ1: Deterministic representation. Do externally different JSON serializations of the same admissible document content produce identical canonical state digests?

EQ2: Detection completeness. Are all deliberately introduced component-level change events represented in the ground-truth manifest detected by the prototype?

EQ3: Localization correctness. For changes for which valid component correspondence is available, does the prototype associate each detected event with the expected component identifier?

EQ4: Hierarchical refinement scope. When two root states differ, what proportion of the hierarchical state representation is inspected during digest-guided recursive refinement?

The evaluation further examines two boundary conditions. First, a controlled schema transformation is used to determine whether explicit migration information can preserve component correspondence across representational change. Second, an ambiguous example without persistent component identifiers is used to demonstrate the limit of deterministic localization when no valid correspondence rule is available.

The evaluation does not include statistical hypothesis testing because the document transitions and expected outcomes are deterministically constructed rather than sampled from a stochastic population. The reported ratios are therefore descriptive measurements of the controlled case-study instance.

Accordingly, the case study should be interpreted as a reproducible operational demonstration rather than a statistical performance evaluation. Its objective is to verify that the computational instantiation behaves consistently with the formal representation model developed in the preceding sections and to provide transparent reference artifacts that enable independent reproduction of the reported results.

5.2. Dataset and Document Structure

The dataset represents a maintenance instruction entitled "Replacement and Calibration of a Spindle-Temperature Sensor". The document was selected because maintenance instructions constitute a realistic class of evolving technical documents and contain both hierarchical structure and operationally meaningful components. The example is synthetic to avoid confidentiality restrictions and to permit complete control over the introduced changes, component identities, and expected outputs.

The document is encoded as JSON and contains a root document object with applicability metadata, revision information, and four principal sections:

1. safety instructions and warnings,
2. required tools,
3. procedural steps,
4. acceptance criteria.

The initial state contains addressable components for individual sections, warnings, tools, procedural steps, and acceptance criteria. Each addressable component is assigned a persistent identifier through an `id` property. Examples include `sec-procedure`, `warning-3`, `step-4`, and `criterion-1`. These identifiers define the component-correspondence relation used by the prototype. Objects and arrays without an `id` property remain structural nodes but are not treated as independently addressable components.

The five document states form the linear predecessor sequence

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4. \tag{3}$$

Each state records its immediate predecessor. Comparisons are performed only between adjacent states in Equation (3). This design corresponds to predecessor-constrained local comparison and does not model multiple-parent merge semantics. Table 7 summarizes the controlled document evolution.

An additional serialization-equivalence document contains the same information as S_0 but uses a different JSON object-key order and different external formatting. This document is used exclusively for the deterministic-representation test.

Table 7. Controlled document-state sequence used in the applied case study.

Transition	Primary Change Category	Controlled Modification
$S_0 \rightarrow S_1$	Value modification	The tightening torque is changed from 4 to 5 Nm. The calibration tolerance and corresponding acceptance threshold are changed from 2.0 to 1.5 °C.
$S_1 \rightarrow S_2$	Insertion and deletion	A certificate-verification warning and a maintenance-log step are inserted. An obsolete connector-seal inspection step is deleted.
$S_2 \rightarrow S_3$	Movement and reordering	The certificate warning is moved from the safety section to the procedure preconditions. Two existing procedural steps exchange their sequence positions while retaining their identifiers.
$S_3 \rightarrow S_4$	Schema evolution	The scalar fields `torque` and `unit` are replaced by the nested object `tightening{value, unit}` under an explicit schema-correspondence rule.

5.3. Prototype Implementation

The reference prototype was implemented in Python and follows the processing pipeline shown in Figure 7. Its principal stages are document loading, canonical serialization, hierarchical state construction, digest computation, predecessor validation, component indexing, state comparison, and structured change reporting.

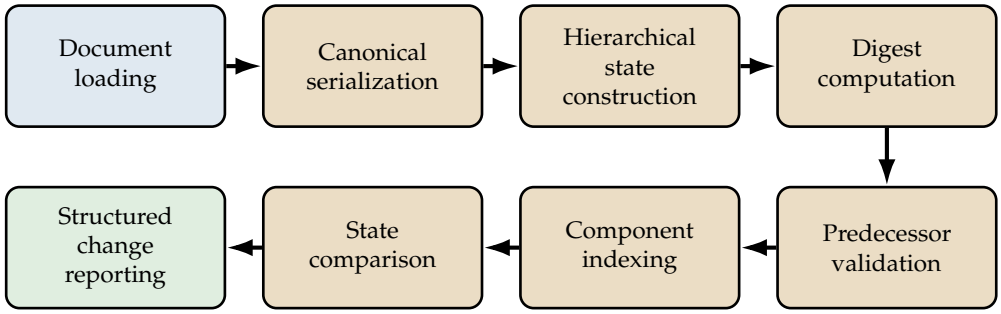


Figure 7. Processing pipeline of the Python reference prototype used in the reproducible operational case study. Each input document is loaded and transformed into a canonical serialization from which a hierarchical document state and its associated digests are constructed. After validation of the predecessor relation, components are indexed to establish the correspondence required for comparison. The resulting current and predecessor states are then compared recursively, and detected differences are returned as a structured change report.

The pipeline separates representation construction from comparison and reporting. Document loading, canonical serialization, hierarchical state construction, and digest computation establish the deterministic internal representation of each document state. Predecessor validation and component indexing then determine whether two states are admissible for comparison and whether their internal components can be related unambiguously. State comparison operates on the resulting hierarchical representations and terminates refinement for unchanged subtrees, whereas detected differences are emitted as structured change records. This separation allows representation errors, correspondence limitations, and comparison outcomes to be examined independently within the case study.

Canonical serialization is performed by recursively ordering object keys, removing insignificant whitespace, preserving Unicode content, and rejecting non-standard numerical values such as `NaN` and infinity. Let x denote an admissible JSON value and let $\mathrm{cjson}(x)$ denote its deterministic serialization. The canonical document digest is calculated as

$$h_{\mathrm{doc}}(x) = \mathrm{SHA256}\big(\mathrm{UTF8}(\mathrm{cjson}(x))\big). \tag{4}$$

The hierarchical representation distinguishes scalar, object, and array nodes. A scalar-node digest includes its type and value. An object-node digest is constructed from its node

type, optional component identifier, and the ordered pairs of child labels and child digests. An array-node digest is constructed from its node type, optional component identifier, and the ordered sequence of child digests. Consequently, a modification of a descendant node changes the digest of each ancestor on the path to the root, while an unchanged subtree retains its previous digest.

Component correspondence is established through persistent `id` values. For every identified component u, the implementation records

$$I(u) = \big(\mathrm{id}(u), \mathrm{path}(u), \mathrm{parent}(u), \mathrm{index}(u), h(u)\big), \tag{5}$$

where $\mathrm{path}(u)$ is its JSON pointer, $\mathrm{parent}(u)$ is the nearest identified ancestor, $\mathrm{index}(u)$ denotes its position within an ordered parent collection where applicable, and $h(u)$ is its hierarchical digest.

For two successive states, changes are classified according to the following implementation rules:

- An identifier occurring only in the successor state is classified as an insertion;
- An identifier occurring only in the predecessor state is classified as a deletion;
- An identifier whose nearest identified parent changes is classified as a movement;
- An identifier whose position changes within the same ordered parent is classified as a reordering;
- An identifier whose canonical component content changes is classified as a modification.

These classifications are not mutually exclusive. For example, a reordered procedural step may also be classified as modified when its explicit `order` attribute changes. Likewise, the digests of enclosing sections and the root document change when one of their descendants changes. The resulting ancestor-level modifications express digest propagation and are distinguished from the primary component-level events specified in the ground-truth manifest.

The implementation performs digest-guided refinement. If two compared nodes have identical digests, their descendants are not inspected further. If their digests differ, comparison proceeds recursively into the corresponding child nodes. The implementation records the number of nodes visited during this refinement procedure.

The software package contains the reproducibility artifacts given in Table 8.

Table 8. Reproducibility artifacts included in the software package.

No.	Artifact	Purpose
1	Five JSON document states	Represent the successive document versions S_0, S_1, S_2, S_3, and S_4 used in the controlled multi-version case study.
2	Serialization-equivalent form of S_0	Provides an externally different JSON serialization of the initial document state for evaluating deterministic canonicalization under irrelevant formatting and object-key-order variation.
3	Schema-correspondence map for $S_3 \rightarrow S_4$	Defines the explicit mapping between the original scalar fields and the restructured nested fields used in the controlled schema-evolution transition.
4	Ground-truth change manifest	Specifies the expected change type and component identifier for each controlled transition and provides the reference for calculating detection completeness and localization correctness.
5	Ambiguous correspondence example	Demonstrates the boundary condition in which structural inequality can be detected but unique component-level localization is not possible because persistent identity information is unavailable.

Table 8. *Cont.*

No.	Artifact	Purpose
6	Executable Python source code	Implements canonical serialization, hierarchical state construction, digest computation, predecessor validation, component indexing, state comparison, change classification, and result generation.
7	Automated unit tests	Verify canonical equivalence, modification detection, insertion and deletion detection, movement and reordering detection, schema-alias processing, and the numerical bounds of the refinement ratio.
8	Machine-readable JSON results	Provide the complete structured outputs of the applied evaluation, including detected changes, localization information, refinement measurements, baseline differences, and ground-truth evaluation metrics.
9	Human-readable result report	Summarizes the execution environment, canonical equivalence test, transition results, evaluation metrics, ambiguous correspondence case, and methodological scope in a directly inspectable format.

5.4. Evaluation Procedure and Metrics

The evaluation was executed under Linux using Python 3.13.5. No third-party runtime dependency was required. Before evaluating the document transitions, six automated unit tests were executed to verify canonical equivalence, modification detection, insertion and deletion detection, movement and reordering detection, schema-alias processing, and the numerical bounds of the refinement ratio. All six tests completed successfully.

For the deterministic-representation test, S_0 was compared with an externally different serialization containing equivalent document content. The two source artifacts differed in object-key order and formatting but represented the same JSON value. The test was considered successful if their canonical document digests, as defined by Equation (4), were identical.

Detection completeness was calculated from the predefined ground-truth manifest as

$$C_{\text{det}} = \frac{N_{\text{detected}}}{N_{\text{expected}}}, \tag{6}$$

where N_{expected} denotes the number of expected change events and N_{detected} denotes the number of expected event–component pairs present in the prototype output. An expected event is represented as the ordered pair

$$e = \big(\text{type}(e), \text{id}(e)\big). \tag{7}$$

Localization correctness was evaluated only for events for which valid component correspondence had been defined. It was calculated as

$$C_{\text{loc}} = \frac{N_{\text{correctly localized}}}{N_{\text{localizable}}}. \tag{8}$$

This qualification is essential because the framework does not infer component identity where correspondence is absent or ambiguous.

The hierarchical refinement ratio for a transition from S_i to S_{i+1} was defined as

$$R_{\text{ref}}(S_i, S_{i+1}) = \frac{N_{\text{visited}}(S_i, S_{i+1})}{\max(N_{\text{nodes}}(S_i), N_{\text{nodes}}(S_{i+1}))}, \tag{9}$$

where N_{visited} is the number of nodes inspected by digest-guided recursive refinement and N_{nodes} is the complete number of nodes in the respective hierarchical representation. The ratio describes the refinement scope of the implemented comparison for the present dataset. It is not a general complexity or scalability measure.

For descriptive comparison, each pair of successive states was additionally processed using a conventional line-oriented unified diff applied to pretty-printed JSON. The baseline

was selected because it is transparent, reproducible, and representative of artifact-level textual comparison. The baseline comparison is qualitative: it examines the form and structural granularity of the reported differences rather than claiming superiority over a specific version control product or structured-differencing library.

5.5. Results

5.5.1. Deterministic Representation

The canonical digest of the initial document state was

$$h_{\text{doc}}(S_0) = \texttt{40b169077d2b050c6cc85f5b6a13ec91}\\ \texttt{1fa7a1543b4e6ed9df9de3345228cfa6}.$$

The externally different but semantically equivalent serialization produced the same digest. The test therefore confirms that, for the transformations included in the case study, object-key order and insignificant JSON formatting do not affect the constructed canonical state. This result addresses EQ1. It does not imply equivalence across arbitrary source formats unless their respective canonicalization mappings produce the same admissible internal state.

5.5.2. Controlled Change Detection and Localization

The ground-truth manifest contained 13 expected event–component pairs across the four transitions. All 13 expected pairs were present in the prototype output, yielding

$$C_{\text{det}} = \frac{13}{13} = 1.000. \tag{10}$$

All expected events were defined over components for which valid correspondence information was available. Each of these events was associated with the expected persistent component identifier, yielding

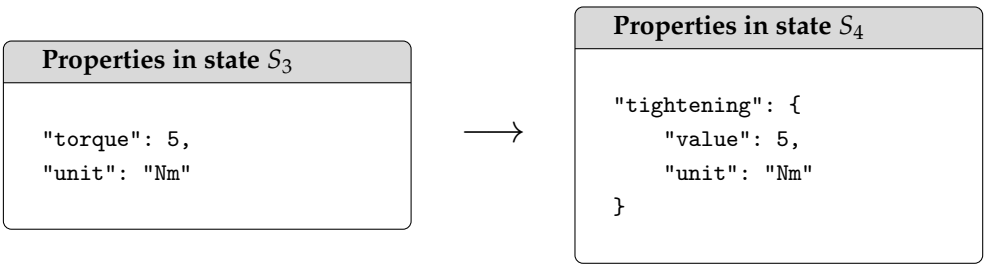
$$C_{\text{loc}} = \frac{13}{13} = 1.000. \tag{11}$$

Table 9 reports the primary controlled events. Modifications additionally propagated to enclosing identified components because their hierarchical digests and canonical contents include the modified descendants. These propagated ancestor modifications are retained in the machine-readable result file but are not counted as separate primary ground-truth events unless explicitly listed in the manifest.

Table 9. Primary controlled change events and prototype results.

Transition	Expected Type	Component	Detected Type	Localized
$S_0 \to S_1$	Modification	`step-4`	Modification	Yes
$S_0 \to S_1$	Modification	`step-5`	Modification	Yes
$S_0 \to S_1$	Modification	`criterion-1`	Modification	Yes
$S_1 \to S_2$	Insertion	`warning-3`	Insertion	Yes
$S_1 \to S_2$	Insertion	`step-8`	Insertion	Yes
$S_1 \to S_2$	Deletion	`step-6`	Deletion	Yes
$S_2 \to S_3$	Movement	`warning-3`	Movement	Yes
$S_2 \to S_3$	Reordering	`step-2`	Reordering	Yes
$S_2 \to S_3$	Reordering	`step-3`	Reordering	Yes
$S_3 \to S_4$	Schema-mediated preservation	`step-4`	No content modification after alias normalization	Yes

The transition $S_3 \rightarrow S_4$ requires separate interpretation. In `step-4`, two scalar properties in S_3 are reorganized into a nested object in S_4.



Although the values are preserved, their structural paths differ. The case study therefore defines the explicit correspondence map given in Table 10.

Table 10. Path correspondence between states S_3 and S_4.

Source Path		Canonical Path
`/torque`	$\longmapsto$	`tightening.value`
`/unit`	$\longmapsto$	`tightening.unit`
`/tightening/value`	$\longmapsto$	`tightening.value`
`/tightening/unit`	$\longmapsto$	`tightening.unit`

After normalization under this map, the operational content of `step-4` remains equal and the prototype does not classify the component itself as modified. The enclosing procedure and document remain structurally modified because the schema representation and document-level metadata changed. This result demonstrates that schema evolution can be handled deterministically only when migration or correspondence semantics are supplied externally. The prototype does not infer them.

5.5.3. Hierarchical Refinement Scope

Table 11 reports the number of nodes visited during hierarchical refinement. In each transition, the root digests differed. Recursive inspection was subsequently terminated at equal subtrees.

Table 11. Observed hierarchical refinement scope for the controlled transitions.

Transition	Visited Nodes	Maximum State Size	Refinement Ratio
$S_0 \rightarrow S_1$	40	84	0.476
$S_1 \rightarrow S_2$	33	88	0.375
$S_2 \rightarrow S_3$	41	89	0.461
$S_3 \rightarrow S_4$	31	91	0.341

For the four transitions, between 34.1% and 47.6% of the nodes in the larger state representation were visited by the implemented refinement procedure. These measurements show that unchanged subtrees were excluded from deeper inspection in the present example. They do not establish an asymptotic advantage or a general reduction factor. The observed scope depends on the location and distribution of modifications, the hierarchical decomposition, array-order semantics, component granularity, and the implementation of child correspondence.

The relatively larger refinement scope for $S_2 \rightarrow S_3$ is consistent with the nature of the transition. Reordering procedural steps and moving a warning between sections affect

multiple structural paths and therefore invalidate more ancestor and array digests than an isolated leaf-value modification.

5.5.4. Comparison with a Textual-Diff Baseline

The conventional unified-diff baseline detected differences for every non-identical transition. Its output, however, remained tied to serialized lines and did not independently express persistent component identity, predecessor relations, or the distinction between movement and deletion followed by insertion.

For $S_2 \rightarrow S_3$, the warning identified as `warning-3` was removed from the serialized safety-section location and inserted under the procedure preconditions. The textual baseline represented these observations as deleted and added line blocks. In contrast, the state-centric prototype used the persistent component identifier and changed parent relation to classify the event as a movement.

The reordering of `step-2` and `step-3` produced a corresponding displacement of serialized line blocks in the baseline. The prototype associated the new positions with the same persistent identifiers and therefore classified both events as reorderings. This distinction does not imply that line-oriented differencing is incorrect. The two methods answer different questions: the textual diff identifies changes in a serialization, whereas the prototype interprets differences relative to the declared state representation and component-correspondence model.

The baseline comparison therefore supports a limited applied conclusion. Representa tion-aware comparison can produce component-level classifications that are not contained in a plain textual diff unless additional identity and structural interpretation mechanisms are applied. No conclusion is drawn regarding runtime superiority, implementation maturity, merge support, or suitability for all document types.

5.6. Boundary Case: Ambiguous Component Correspondence

A separate negative case was constructed to examine the behavior of the implementation when persistent identity is unavailable. The predecessor document contains two identical procedural entries, both represented only by the text "Inspect connector", followed by the entry "Record result". In the successor document, one occurrence of "Inspect connector" appears after "Record result". None of the three entries has a persistent component identifier.

The predecessor and successor states have different root digests, and structural inequality is therefore detected. The implementation cannot, however, determine which of the two indistinguishable inspection entries retained its identity and changed position. Consequently, it cannot establish a unique identity-preserving movement event.

This result is not an implementation failure relative to the formal assumptions. It demonstrates the dependence of component-level localization on a valid correspondence relation. Given two indistinguishable components u_1 and u_2 in the predecessor state and an indistinguishable component $\tilde{u}$ in the successor state, both assignments $u_1 \leftrightarrow \tilde{u}$ and $u_2 \leftrightarrow \tilde{u}$ are compatible with the available structural information. A unique movement classification would therefore require an additional rule, such as a persistent identifier, a schema-defined key, an application-specific identity constraint, or a semantic disambiguation procedure.

The boundary case establishes an important distinction between change detection and change localization. State inequality may be deterministic even where component identity is underdetermined. The framework can therefore report that a structural transition occurred without claiming unique localization where its correspondence assumptions are not satisfied.

5.7. Reproducibility, Interpretation, and Limitations

The case study provides a complete executable chain from source documents to reported change events. Reproduction requires a Linux-compatible environment with Python 3.10 or later. The automated tests and the applied evaluation can be executed using

```
python3 -m unittest discover -s tests -v
python3 run_case_study.py
```

The second command generates both a machine-readable JSON result file and a Markdown report. An optional execution mode performs repeated timing measurements:

```
python3 run_case_study.py --benchmark --iterations 500
```

Timing measurements are not reported as primary evidence in the present study because the dataset is intentionally small and because absolute execution times depend on hardware, operating-system activity, interpreter version, and implementation details. The timing mode is included to permit independent inspection of the reference implementation and to support subsequent benchmark extensions.

The applied results support four bounded conclusions. First, canonicalization removed the deliberately introduced JSON key-order and formatting variation. Second, the prototype detected and localized all expected controlled events for which valid correspondence was supplied. Third, digest-guided refinement avoided recursive inspection of equal subtrees in the evaluated transitions. Fourth, the ambiguous case confirmed that unique component-level localization cannot be obtained from structural inequality alone when identity information is insufficient. Several limitations constrain the interpretation of these results.

1. The dataset contains one synthetic maintenance document and five linear states. It does not represent the distribution, heterogeneity, or scale of industrial document repositories.
2. The ground truth was constructed together with the dataset. The reported completeness and localization ratios therefore establish conformance of the prototype to the controlled specification, not predictive accuracy on independently annotated documents.
3. Component correspondence is primarily based on persistent identifiers. The implementation does not provide a universal matching algorithm for renamed, duplicated, semantically similar, or weakly structured components.
4. Schema evolution is handled through an explicit correspondence map. The prototype demonstrates the use of supplied migration semantics but does not infer such semantics automatically.
5. The evaluated history is linear. Branching histories, multiple-parent merges, conflict resolution, and branch reconciliation remain outside the implemented case-study scope.
6. The textual-diff baseline is included to illustrate differences in output granularity. It is not a comprehensive comparison with Git, XML-differencing tools, JSON Patch implementations, temporal databases, or authenticated data structures.
7. The observed refinement ratios characterize only the selected document structures and modifications. They do not establish general scalability, storage efficiency, latency, throughput, or computational superiority.

Within these limits, the case study provides direct applied evidence that the proposed framework can be instantiated as an executable change-analysis workflow. Broader conclusions concerning efficiency, robustness, scalability, and cross-domain utility require evaluation using independent document collections, alternative correspondence mechanisms, larger version histories, and implementation-specific benchmarks. The following subsection integrates these applied findings with the analytical assessment presented in Section 4.

5.8. Synthesis of the Analytical and Applied Assessment

The evaluation combines two complementary forms of evidence. The analytical assessment (Section 4) established the conceptual consistency of the representation framework by examining the correspondence between its formal constructs and the research objectives. The operational case study complements this analysis by demonstrating that these constructs can be instantiated in an executable and independently reproducible workflow whose behavior can be evaluated against predefined expectations.

The analytical assessment showed that the framework provides a coherent foundation for canonical document-state construction, deterministic comparison, predecessor-constrained evolution, hierarchical localization, and the separation of structural detection from contextual interpretation. It further demonstrated that the representation model admits a computational realization based on hierarchical digest comparison. These results establish conceptual validity and computational feasibility but do not, by themselves, verify the behavior of a concrete implementation.

The operational case study provides this empirical evidence. Equivalent JSON serializations produced identical canonical document digests throughout the evaluation. All 13 expected event–component pairs were detected correctly, and every event for which valid correspondence information was available was assigned to the expected persistent component identifier. Hierarchical refinement consistently terminated at equal subtrees, while the explicit correspondence map preserved component identity across the controlled schema transformation. Conversely, the negative correspondence scenario demonstrated that structural differences alone are insufficient for unique component-level localization when persistent correspondence cannot be established.

Together, these findings support the following conclusion. Under the assumptions defined by the framework, canonical state construction, hierarchical decomposition, explicit predecessor linkage, and valid component correspondence can be combined to provide deterministic document comparison and component-level localization in structured document histories. Equally important, the negative correspondence scenario identifies a fundamental boundary of the approach: deterministic detection of structural differences and deterministic attribution of those differences to persistent components are distinct capabilities that rely on different information.

The scope of these conclusions is intentionally limited. The evaluation does not establish implementation-independent performance, scalability, memory efficiency, robustness, or cross-domain applicability. The reported results are derived from a controlled document collection, a single reference implementation, a predefined ground truth, and explicitly specified correspondence rules. Broader claims regarding practical effectiveness therefore require independent implementations, more heterogeneous datasets, and systematic comparative benchmark studies.

The analytical assessment and the operational case study thus provide complementary evidence at different levels of abstraction. The former establishes the theoretical soundness of the proposed framework, whereas the latter demonstrates the behavior of one computational realization under controlled and reproducible conditions. Together, they provide the evidential basis for the discussion of implications and limitations in the following section.

6. Discussion

6.1. Architectural and Design Implications

The preceding sections established the proposed framework through two complementary forms of evidence. The analytical assessment examined the internal consistency of the representation model and its correspondence to the stated design objectives, whereas the reproducible operational case study demonstrated that these concepts can be instantiated

within an executable document-processing workflow under controlled and independently reproducible conditions. The present discussion considers the broader architectural implications of these findings while distinguishing conclusions supported by the current work from questions requiring further empirical investigation.

A central implication of the proposed framework is a shift from algorithm-centric toward representation-centric design of change-aware information systems. In many conventional approaches, data representations are inherited implicitly from storage formats, database schemas, or external file structures, while design effort concentrates mainly on comparison procedures. The present work suggests the reverse priority: the reproducibility, scalability, and interpretability of change handling depend first on how evolving information is represented.

The proposed framework should therefore be understood as providing representational support for higher-level engineering workflows rather than replacing domain-specific decision models. Decision-support frameworks may subsequently exploit the canonical document-state representations, explicit predecessor relations, and localized change information established by the proposed representation model to perform application-specific reasoning, engineering assessment, or downstream decision support.

Within the proposed model, stable document states form the enabling layer for all subsequent comparison. If no canonical representation exists, comparison remains tied to raw artifacts and therefore becomes sensitive to incidental variation such as ordering, formatting, encoding, or storage layout. Once information is transformed into deterministic and hierarchically structured states, comparison can instead operate on stable internal objects.

This change in perspective has direct design consequences. Decisions regarding canonicalization rules, structural granularity, hierarchical decomposition, and predecessor linkage determine not only how information is stored, but also:

1. What counts as equality;
2. How efficiently change can be detected;
3. Where differences can be localized;
4. How document evolution can be reconstructed.

Representation is therefore not a secondary implementation detail, but part of the formal system specification.

A second architectural implication is the explicit separation between structural detection and contextual interpretation. The comparison layer determines whether change exists and where it is located. The interpretation layer determines whether that change is important, risky, urgent, or actionable. This separation supports modular system architectures in which a stable comparison core is complemented by higher layers for prioritization, explanation, notification, or workflow control.

6.2. Potential Computational Implications, Traceability, and Human Interaction

From a computational perspective, the framework identifies representational conditions that may reduce repeated comparison effort in suitable implementations. In artifact-centric settings, raw documents may need to be parsed, traversed, and reconciled repeatedly. Under the proposed model, part of this effort can instead be shifted to state construction, canonicalization, correspondence establishment, and optional digest computation, after which repeated comparison may operate on reusable state representations or derived summaries.

This observation should not be interpreted as an empirical scalability result. The actual computational behavior depends on document size and structure, preprocessing cost, correspondence establishment, hierarchy depth, change distribution, storage architecture, caching strategy, implementation architecture, and workload. The framework therefore

provides a structural rationale for investigating efficient repeated comparison, but does not establish universal performance or scalability guarantees.

The reproducible operational case study supports the computational plausibility of these design principles within a controlled execution scenario. In particular, it demonstrates deterministic canonicalization, hierarchical localization, predecessor-constrained comparison, and selective recursive refinement for one executable realization. However, it is intentionally not designed as a comparative benchmark and therefore does not establish implementation-independent runtime behavior, scalability, or robustness across heterogeneous document collections.

This reallocation of effort yields several practical consequences:

1. Repeated comparison becomes feasible under high query frequency;
2. Decision effort becomes less dependent on raw artifact size;
3. Unchanged regions can be skipped systematically;
4. Concurrent access to evolving collections can be supported more efficiently.

The framework does not claim that comparison becomes cost-free. Rather, it changes where effort is incurred: from repeated full-artifact comparison toward state construction, reusable summaries, and selective refinement.

Explicit predecessor linkage also strengthens traceability and auditability. Because each state refers to a valid predecessor, document evolution can be reconstructed as an explicit transition path rather than inferred retrospectively from unrelated snapshots. This is particularly relevant in regulated or safety-critical environments requiring accountability, quality assurance, or technical verification.

The framework further has implications for human–system interaction. Users are rarely interested in structural differences as such; instead, they seek information relevant to their responsibilities. Hierarchical localization therefore supports role-adapted information delivery. Decision-makers may require high-level summaries, whereas technical experts or auditors may require fine-grained structural evidence. Because interpretation is separated from detection, user-facing relevance logic can evolve without changing the formal comparison layer.

6.3. Cross-Domain Applicability and Research Positioning

Although motivated by structured document scenarios, the framework is not restricted to a single domain. Its core principles, canonical state representation, hierarchical decomposition, predecessor linkage, and context-dependent interpretation, apply wherever evolving structured information must be compared repeatedly and transparently. Representative application areas include:

- Engineering and technical documentation;
- Software and configuration management;
- Enterprise master-data and lifecycle records;
- Compliance and regulatory documentation;
- Maintenance procedures and operational instructions;
- Knowledge repositories and digital manuals.

Across these domains, information evolves over time, requires traceable comparison, and benefits from relevance-oriented interpretation.

The framework is also relevant to research data management. Scientific datasets frequently evolve while remaining reproducible, machine-readable, and versionable. The FAIR principles emphasize findability, accessibility, interoperability, and reusability of data [4]. The present model complements such infrastructures by representing datasets

as linked states, thereby supporting deterministic comparison, reconstruction of prior versions, and formal reasoning about dataset evolution.

At the same time, the framework does not replace existing standards, metadata schemas, or domain infrastructures. Rather, it provides a structural layer that can be integrated with them.

Relative to established paradigms, the framework intersects with version control systems, database change tracking, and digest-based integrity models. Its distinctive contribution is not a competing implementation but a unifying representation model that makes their often implicit assumptions explicit and analytically tractable. The accompanying reproducible operational case study demonstrates that these representational principles can be realized within an executable workflow while remaining independent of any specific application domain or implementation architecture.

6.4. Methodological Limitations

The representation-centric perspective also has clear limitations.

The proposed framework is conceptual and representation-oriented. It does not provide a comprehensive empirical benchmark of specific comparison algorithms, nor does it claim that one implementation architecture is universally optimal. Instead, its purpose is to define the representational, structural, and temporal conditions under which efficient and meaningful comparison becomes possible.

A first limitation concerns the assumption of structured and canonicalizable document content. The framework is most directly applicable to documents that can be transformed into stable internal representations, such as structured engineering documents, compliance records, technical documentation, XML/JSON-based artifacts, or database-derived document states. Highly unstructured, multimedia-rich, or semantically ambiguous documents may require additional preprocessing, extraction, or domain-specific modeling before the framework can be applied.

A second limitation concerns the interpretation of detected changes. The framework can support deterministic detection and localization of structural differences, but it cannot determine the practical relevance of every change independently of context. Relevance depends on domain rules, organizational responsibilities, user roles, regulatory constraints, and operational consequences. Therefore, the interpretation layer remains necessarily context-dependent and may require expert validation.

A third limitation concerns implementation choices. The hierarchical digest-based comparison discussed in this paper is one possible realization of the framework, but not the only one. Alternative implementations may use tree differencing, database-native mechanisms, graph-based representations, event streams, or hybrid approaches. Their suitability depends on document structure, update frequency, latency requirements, storage constraints, and integration environments.

Fourth, the framework currently assumes deterministic correspondence establishment between structurally related components. Highly dynamic or weakly structured document environments may therefore require additional semantic matching mechanisms beyond the present scope. Deterministic localization requires a valid correspondence relation between structural components of the compared states. The general framework does not prescribe a universal matching algorithm. Correspondence may be established through persistent identifiers, schema-defined keys, stable structural paths, canonical names, or application-specific procedures. Where correspondence is partial, localization is correspondingly partial; where it is ambiguous, unique localization requires an external disambiguation rule; and where no valid correspondence exists, component-level localization is not guaranteed.

Fifth, the framework does not define a complete edit algebra for arbitrary structural evolution. Insertions, deletions, movement, reordering, and schema evolution require representation-specific identity, ordering, correspondence, or migration semantics. Likewise, the predecessor model addresses linear or predecessor-selected evolution paths and does not provide general multiple-parent merge semantics, conflict resolution, or branch reconciliation.

Sixth, the analytical assessment and the reproducible operational case study provide complementary conceptual and operational evidence but do not constitute comprehensive empirical validation. The analytical assessment establishes the internal consistency of the representation model, whereas the case study demonstrates one executable realization under controlled conditions. The paper does not provide benchmark datasets, comparative performance studies, runtime or memory measurements, statistical analyses, or evaluations across multiple independent implementations. Consequently, statements concerning computational efficiency, scalability, robustness, or general operational effectiveness remain conditional and implementation-dependent. Future work should therefore investigate multiple independent computational instantiations derived from the proposed representation framework. Comparative benchmark studies across heterogeneous document collections, correspondence strategies, canonicalization procedures, and implementation architectures would help quantify the practical implications of the representational principles established in this paper.

Finally, the framework focuses on the modeling and comparison of successive document states. It does not replace governance processes for version approval, semantic validation, access control, or organizational decision-making. Rather, it provides a representational foundation on which such processes can be implemented, evaluated, and compared in a transparent and reproducible manner.

6.5. Possible Application Scenarios

Figure 8 illustrates how heterogeneous document-driven domains can be mapped to a common state-centric change engine. Despite differences in source systems and business context, the same structural logic can be reused: artifacts are transformed into canonical states, linked through predecessor relations, compared efficiently, and interpreted according to domain-specific relevance criteria.

The operational case study presented in this paper represents one concrete instantiation of this general architecture using structured JSON documents. The same representational principles are intended to remain applicable across heterogeneous document domains, although the required canonicalization rules, correspondence mechanisms, and interpretation models necessarily remain application-specific.

The resulting benefit is reusable change handling across information systems, combining traceable comparison, localized refinement, and context-sensitive decision support.

6.6. Summary

The discussion shows that the proposed framework contributes not only a formal model, but also a broader systems perspective on evolving information.

By treating representation as a first-class design concern, separating detection from interpretation, and linking states through explicit temporal relations, the framework provides a basis for scalable, transparent, and adaptable change-aware information systems. Its practical effectiveness depends on appropriate representation design, carefully chosen canonicalization rules, and integration with contextual decision processes.

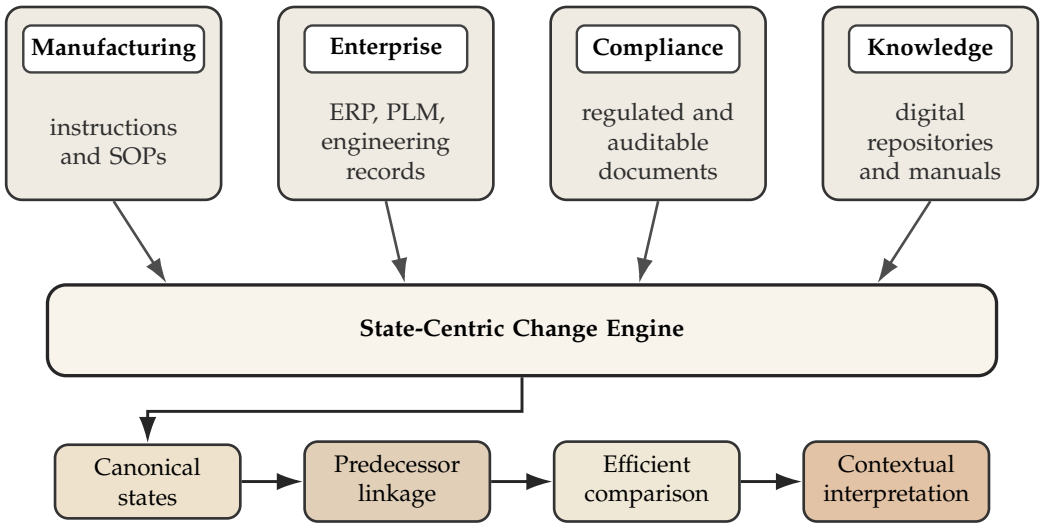


Figure 8. Application-oriented view of the proposed state-centric framework. The upper row uses a neutral color to indicate heterogeneous application domains that can provide evolving document artifacts. The lower row uses the same functional color sequence as the conceptual architecture: representation through canonical states, temporal organization through predecessor linkage, comparison through efficient state evaluation, and interpretation through contextual assessment. The figure illustrates how heterogeneous document-driven domains can be mapped to a common change engine while preserving domain-specific interpretation.

7. Conclusions

7.1. Principal Findings

This paper developed a conceptual and formal representation framework for representing, linking, and comparing evolving documents in information systems. The central argument is that well-defined and interpretable change analysis cannot be reduced to comparison algorithms alone. It also requires explicit representational conditions under which external artifacts are transformed into canonical, hierarchically structured, and temporally linked document states. Under the stated assumptions, including valid comparability and component correspondence where localization is required, change can be defined as a transition between states, structurally localized, and subsequently interpreted according to domain-specific relevance criteria.

Conventional artifact-centered approaches typically compare files, records, or serialized objects directly. While operationally useful, such approaches often inherit incidental variation caused by formatting, ordering, encoding, or storage conventions. They may therefore increase computational overhead, weaken reproducibility, and blur the distinction between meaningful and superficial differences.

To address this issue, the proposed framework reformulates documents as canonical and hierarchically organized states embedded in an evolving information space. Within this perspective, change is defined as a formally describable relation between comparable states rather than as an accidental mismatch between external artifacts.

The framework was evaluated through two complementary forms of evidence. The analytical assessment examined the internal consistency of the representation model, its correspondence to the stated design objectives, and the computational realizability of one digest-based instantiation. The accompanying reproducible operational case study demonstrated that these concepts can be instantiated within an executable comparison workflow using predefined document states, explicit correspondence information, and independently reproducible evaluation artifacts. Together, these complementary assessments establish the conceptual coherence and operational realizability of the proposed framework under the stated representational assumptions. They do not constitute comparative empirical

validation of runtime performance, scalability, robustness, or implementation superiority. The contribution of the present work therefore lies in establishing and analyzing the representational conditions for change-aware document systems rather than benchmarking a particular software realization.

Three findings emerged as foundational for effective change-aware systems:

1. **Hierarchical decomposition** enables localized comparison and progressive refinement, allowing systems to focus on affected structural regions rather than re-evaluating complete artifacts.
2. **Canonicalization** establishes deterministic comparability by reducing sensitivity to superficial representational variation and by clarifying which distinctions are relevant under the chosen model.
3. **Predecessor linkage** constrains comparison with coherent evolution steps, thereby strengthening temporal consistency, version traceability, rollback analysis, and historical reconstruction.

Taken together, these findings show that representation, temporal evolution, and comparison are not independent design concerns but mutually dependent components of one integrated document-state model. The analytical assessment establishes the conceptual consistency of this model, while the reproducible operational case study demonstrates that it can be instantiated within an executable comparison workflow under controlled conditions.

The hierarchical digest-based computational instantiation demonstrates one executable realization of the proposed representation framework. The accompanying operational case study shows how deterministic canonicalization, predecessor-constrained comparison, and hierarchical refinement interact within a reproducible execution scenario. These observations support the conclusion that efficient localized comparison depends not only on comparison algorithms but also on the quality of the underlying representation model. General conclusions concerning computational efficiency remain implementation-dependent and require systematic benchmark evaluation across alternative realizations and representative document collections.

7.2. Implications for Information Systems Design and Application

The framework contributes to a broader systems perspective in which change is not merely detected after the fact, but becomes explicitly modeled through relations among successive states. This perspective is particularly relevant for environments characterized by frequent updates, repeated comparison requests, distributed data ownership, and strong governance requirements.

Representative application domains include:

- Engineering documentation and technical manuals;
- Enterprise knowledge repositories;
- Product-lifecycle-management systems;
- Digital twins and cyber-physical information models;
- Maintenance and service documentation;
- Regulatory and compliance environments;
- Research-data infrastructures.

In these domains, organizations increasingly require systems that can reliably determine whether something changed, precisely localize where it changed, preserve auditable version histories, and support timely downstream decisions.

The framework also clarifies an important architectural distinction. Structural comparison alone cannot determine practical significance. Whether a change is critical, risky, urgent, or actionable depends on external criteria such as domain expertise, organizational policies,

safety constraints, legal requirements, and user responsibilities. Effective change-aware systems therefore require two complementary layers:

- A deterministic detection layer, responsible for establishing whether states differ and where differences occur;
- A context-sensitive interpretation layer, responsible for evaluating relevance, urgency, consequences, and response options.

This distinction is especially important for socio-technical systems in which automated processing and human judgment must interact transparently and accountably.

7.3. Limitations

The present study is conceptual and model-oriented. Its contribution lies in defining a general framework rather than empirically proving the superiority of one specific implementation technique.

Several limitations should therefore be noted. First, the framework is most directly applicable where document content can be transformed into sufficiently stable internal representations. Highly unstructured, ambiguous, or multimedia-dominated data landscapes may require additional preprocessing or domain-specific modeling. Second, the framework focuses primarily on structural comparability and temporal linkage. It does not by itself determine semantic relevance, business priority, or organizational criticality. Third, implementation trade-offs remain context-dependent. Runtime requirements, storage constraints, governance rules, security needs, and legacy-system integration may justify different technical realizations. Fourth, the reproducible operational case study evaluates one computational instantiation of the proposed framework under controlled conditions. Although it demonstrates executable realization and operational consistency, it does not exhaust the range of possible comparison mechanisms, canonicalization strategies, correspondence models, or implementation architectures that are compatible with the proposed representation framework.

The conclusions of this study are therefore deliberately limited to the conceptual and formal level. The framework identifies representational conditions for deterministic comparability, predecessor-constrained evolution, and localized structural analysis under explicit assumptions. It does not establish empirical superiority over existing differencing, version control, database, provenance, or authenticated-data-structure approaches. Nor does it provide a universal solution to component matching, arbitrary structural editing, schema evolution, or multi-parent merge semantics.

7.4. Future Research

Several research directions emerge naturally from the proposed framework. The present work establishes a conceptual representation model, analytically examines its internal consistency, and demonstrates one executable realization through a reproducible operational case study (Table 12). Future research should therefore investigate multiple independent computational instantiations across different document domains, storage architectures, correspondence mechanisms, and workload conditions. Comparative benchmark studies should evaluate runtime behavior, memory consumption, comparison effort, localization precision, preprocessing overhead, and scalability using standardized and heterogeneous document collections. Such investigations would complement the present contribution by identifying the implementation conditions under which the proposed representational principles support efficient and robust change-aware information systems.

Table 12. Structured future research agenda for extending the proposed framework. The table summarizes future work according to temporal horizon, research direction, scientific focus, priority level, and expected impact. The "Horizon" column uses symbols to indicate the planned time frame (▬▭: Near-term, ▬▬: Mid-term, ▬▬: Long-term), while the "Priority" column indicates the relative research priority (•••: High, ••◦: Medium).

Horizon	Research Direction	Scientific Focus	Priority	Expected Impact
Mid-term ▬▬	Knowledge representation and semantic enrichment	Integration of ontologies and richer semantic models to improve interpretation of document structures, process dependencies, and change categories; enhanced querying and prioritization of detected changes according to operational relevance, role context, or safety criticality; stronger support for locally deployed AI assistants and more natural user interaction.	High •••	Extension from primarily syntactic change detection toward semantically informed prioritization, contextual interpretation, and intelligent decision support.
Near-term ▬▭	Automated quality assurance and implementation verification	Integration of sensor data, machine states, tool parameters, and process measurements to compare prescribed and observed execution; detection of discrepancies between documentation and operational execution; support for closed-loop quality assurance and feedback to engineering, supervision, and training processes.	High •••	Direct industrial applicability through verification of implementation quality and improved consistency between documented and executed processes.
Near-term ▬▭	Application domains and deployment contexts	Extension to maintenance and repair, quality inspection, pilot production, and training environments; evaluation of alternative interfaces such as augmented-reality systems, wearable devices, low-interruption notification channels, and pre-shift briefing concepts; comparison of individualized and group-based deployment strategies.	Medium ••◦	Validation of transferability, broader industrial applicability, and improved understanding of deployment-specific requirements.
Long-term ▬▬	Advanced technical extensions	Adaptive instructional depth based on user expertise; AI-supported explanation and learning-content generation; event-driven architectures and advanced caching strategies; integration with digital-twin concepts; predictive assistance mechanisms based on change complexity, historical patterns, and contextual information.	Medium ••◦	Improved scalability, adaptability, explainability, and integration with future Industry 5.0 and digital-twin ecosystems.

Overall, future work should progress in two complementary directions: (i) richer semantic intelligence layered on top of deterministic state comparison, and (ii) broader empirical and implementation-oriented evaluation across operational environments.

7.5. Final Remark

In conclusion, this paper establishes a representation-centric foundation for change-aware information systems by integrating canonical document-state representation, explicit predecessor relations, hierarchical comparison, and contextual interpretation within a unified conceptual framework. Rather than proposing a new comparison algorithm or version control mechanism, the framework makes explicit the representational conditions under which deterministic, traceable, and meaningful comparison of evolving document states becomes possible. In this way, the proposed framework complements existing implementation-oriented approaches by providing a coherent conceptual basis for the design and analysis of change-aware information systems.

Supplementary Materials: The supplementary materials include the complete software implementation, case-study dataset, ground-truth annotations, and reproducibility artifacts. The same materials are also permanently archived and publicly available at https://doi.org/10.5281/zenodo.21490010.

Author Contributions: Conceptualization, K.F., M.Z. and H.W.; Data Curation, K.F.; Formal Analysis, K.F. and M.Z.; Funding Acquisition, H.W. and S.I.; Investigation, K.F. and M.Z.; Methodology, K.F. and M.Z.; Project Administration, K.F., M.Z. and H.W.; Resources, K.F. and M.Z.; Software, K.F. and M.Z.; Supervision, H.W. and S.I.; Validation, K.F. and M.Z.; Visualization, K.F. and M.Z.; Writing—Original Draft, K.F. and M.Z.; Writing—Review & Editing, K.F., M.Z., H.W. and S.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the German Federal Ministry for Economic Affairs and Climate Protection (BMWK) under the funding guideline "Digitization of the vehicle manufacturers and supplier industry" within the framework "Future investments in vehicle manufacturers and the supplier industry", financed by the European Union, and supervised by the project sponsor VDI Technologiezentrum GmbH, as part of the joint research project "Werk 4.0" (grant number 13IK022K). The article processing charge (APC) was funded by the Saxon State and University Library Dresden (SLUB Dresden).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The complete dataset, executable reference implementation, source code, ground-truth manifest, schema-correspondence map, automated tests, generated results, and supporting documentation for the reproducible operational case study are openly available through Zenodo Version 1.0.0 [31]. The manuscript provides the formal framework, methodological descriptions, and analytical assessment required to reproduce and interpret the reported findings.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gray, J.; Reuter, A. *Transaction Processing: Concepts and Techniques*; Morgan Kaufmann: Burlington, MA, USA, 1992.
2. Kleppmann, M. *Designing Data-Intensive Applications*; O'Reilly Media: Sebastopol, CA, USA, 2017.
3. Das, S.; Botev, C.; Surlaker, K.; Ghosh, B.; Varadarajan, B.; Nagaraj, S.; Zhang, D.; Gao, L.; Westerman, J.; Ganti, P.; et al. All Aboard the Databus!: LinkedIn's Scalable Consistent Change Data Capture Platform. In Proceedings of the 3rd ACM Symposium on Cloud Computing, San Jose, CA, USA, 14–17 October 2012. [CrossRef]
4. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef] [PubMed]
5. Sarfarazi, S.; Fulgione, M.; Fabbrocino, F. Physics-Informed Decision Framework for Reuse of Reclaimed Steel Members Under Uncertainty. *Metals* **2026**, *16*, 171. [CrossRef]
6. Wang, K.; Cheng, L.; Yin, M.; Zhang, K.; Wang, R.; Zhang, M.; Sun, R. Evolutionary Game Theory in Energy Storage Systems: A Systematic Review of Collaborative Decision-Making, Operational Strategies, and Coordination Mechanisms for Renewable Energy Integration. *Sustainability* **2025**, *17*, 7400. [CrossRef]
7. Tichy, W.F. RCS—A System for Version Control. *Softw. Pract. Exp.* **1985**, *15*, 637–654. [CrossRef]
8. Chacon, S.; Straub, B. *Pro Git*; Apress: New York, NY, USA, 2014.
9. Mens, T. A State-of-the-Art Survey on Software Merging. *IEEE Trans. Softw. Eng.* **2002**, *28*, 449–462. [CrossRef]
10. Chawathe, S.S.; Rajaraman, A.; Garcia-Molina, H.; Widom, J. Change Detection in Hierarchically Structured Information. In Proceedings of the SIGMOD, Montreal, QC, Canada, 4–6 June 1996; pp. 493–504.
11. Chawathe, S.S.; Garcia-Molina, H. Meaningful Change Detection in Structured Data. In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, AZ, USA, 13–15 May 1997; pp. 26–37. [CrossRef]
12. Chien, S.Y.; Tsotras, V.J.; Zaniolo, C. XML Document Versioning. *ACM SIGMOD Rec.* **2001**, *30*, 46–53. [CrossRef]
13. Chien, S.Y.; Tsotras, V.J.; Zaniolo, C. Supporting Complex Queries on Multiversion XML Documents. *ACM Trans. Database Syst.* **2006**, *31*, 407–447. [CrossRef]
14. Sukarsa, I.M.; Wisswani, N.W.; Putra, I.K.G.D.; Linawati. Change Data Capture on OLTP Staging Area for Nearly Real Time Data Warehouse Base on Database Trigger. *Int. J. Comput. Appl.* **2012**, *52*, 32–37. [CrossRef]
15. Dohrn, H.; Riehle, D. Fine-Grained Change Detection in Structured Text Documents. In Proceedings of the 2014 ACM Symposium on Document Engineering, Collins, CO, USA, 16–19 September 2014; pp. 87–96. [CrossRef]
16. Cobena, G.; Abiteboul, S.; Marian, A. Detecting Changes in XML Documents. In Proceedings of the ICDE, San Jose, CA, USA, 26 February–1 March 2002.

17. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In *Proceedings of the Advances in Cryptology—CRYPTO '87*; Springer: Berlin/Heidelberg, Germany, 1988; pp. 369–378.
18. Bird, C.; Rigby, P.C.; Barr, E.T.; Hamilton, D.J.; German, D.M.; Devanbu, P. The Promises and Perils of Mining Git. In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, Canada, 16–17 May 2009; pp. 1–10.
19. Kalliamvakou, E.; Gousios, G.; Blincoe, K.; Singer, L.; German, D.M.; Damian, D. An In-Depth Study of the Promises and Perils of Mining GitHub. *Empir. Softw. Eng.* **2016**, *21*, 2035–2071. [CrossRef]
20. Zinner, M.; Rink, K.; Jäkel, R.; Feldhoff, K.; Grunzke, R.; Fischer, T.; Song, R.; Walther, M.; Jejkal, T.; Kolditz, O.; et al. Revision Control and Automatic Documentation for the Development Numerical Models for Scientific Applications. *Int. J. Adv. Softw.* **2018**, *11*, 214–226.
21. Yuan, G.; Lu, J.; Yan, Z.; Wu, S. A Survey on Mapping Semi-Structured Data and Graph Data to Relational Data. *ACM Comput. Surv.* **2023**, *55*, 1–38. [CrossRef]
22. Tamassia, R. Authenticated Data Structures. In *Proceedings of the Algorithms—ESA 2003*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2832, pp. 2–5. [CrossRef]
23. Di Battista, G.; Palazzi, B. Authenticated Relational Tables and Authenticated Skip Lists. In *Proceedings of the Data and Applications Security XXI*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4602, pp. 31–46. [CrossRef]
24. Papamanthou, C.; Shi, E.; Tamassia, R.; Yi, K. Streaming Authenticated Data Structures. In *Proceedings of the Advances in Cryptology—EUROCRYPT 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 353–370.
25. Falleri, J.R.; Martinez, M. Fine-Grained, Accurate and Scalable Source Differencing. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Lisbon, Portugal, 14–20 April 2024. [CrossRef]
26. W3C. *Canonical XML*, version 1.1; W3C Recommendation: Wakefield, MA, USA, 2008.
27. Rundgren, A.; Jordan, S.; Erdtman, G. *RFC 8785: JSON Canonicalization Scheme (JCS)*; RFC Editor: Wilmington, DE, USA, 2020.
28. Bryan, P.; Nottingham, M. *RFC 6902: JavaScript Object Notation (JSON) Patch*; RFC Editor: Wilmington, DE, USA, 2013.
29. Hoffman, P.; Snell, J. *RFC 7386: JSON Merge Patch*; RFC Editor: Wilmington, DE, USA, 2014.
30. Zinner, M.; Feldhoff, K.; Wiemer, H.; Ihlenfeldt, S. Hierarchical Hash-Based Change Detection for Near-Real-Time Instruction Updates in Manufacturing. *Appl. Sci.* **2026**, *16*, 5980. [CrossRef]
31. Feldhoff, K.; Zinner, M.; Wiemer, H.; Ihlenfeldt, S. State-Centric Structured-Document Case Study: Software, Dataset, Ground Truth, and Reproducibility Artifacts. Software and Reproducibility Package. Available online: https://zenodo.org/records/21490010 (accessed on 27 July 2026). [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.