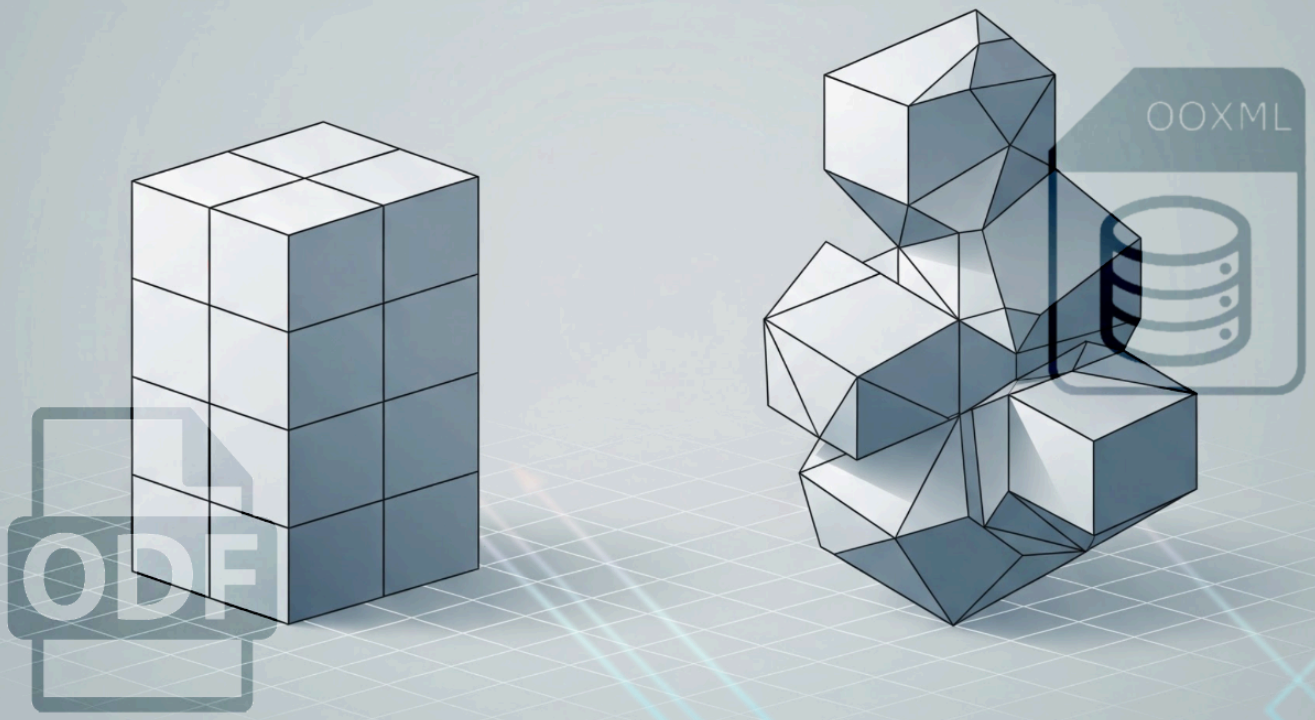
Acrilic

A Comparative Study of OOXML and ODF Implementations



OTA
Open To All

1. Abstract

This paper investigates the architectural divergence between two prominent open-source productivity suites: LibreOffice and ONLYOFFICE. We evaluate the engineering trade-offs inherent in their respective data models, focusing on the Open Document Format (ODF) and Office Open XML (OOXML). Specifically, the study analyzes the parsing bottlenecks associated with the ISO/IEC 29500 Transitional schema and the implications of in-memory format translation pipelines. Furthermore, we assess the impact of graphical user interface (GUI) rendering architectures by comparing legacy abstraction layers (VCL[1]) against modern, hardware-accelerated web technologies utilizing the Chromium Embedded Framework (CEF). The findings provide a comparative framework for evaluating interoperability, rendering latency and standard compliance across deployment contexts.

2. Introduction: Standardization and Engine Architectures

The open-source office suite ecosystem operates on two distinct engineering pipelines, driven by divergent underlying data models. The Document Foundation maintains LibreOffice, a suite designed natively around the vendor-neutral ODF standard [1]. Conversely, ONLYOFFICE prioritizes structural interoperability with the Microsoft Office ecosystem via an OOXML-centric parsing engine [2].

The decision to deploy either platform constitutes a fundamental engineering choice regarding data sovereignty, memory-safe parsing pipelines and the management of legacy technical debt.

3. Methodology

To empirically evaluate the architectural divergence between the ODF and OOXML engines, this study employs a multi-tiered benchmarking methodology. The evaluation isolates document translation fidelity, UI rendering latency and execution sandboxing across both application cores.

3.1. Test Environment and Profiling Infrastructure

All benchmarks are executed on a standardized, bare-metal hardware profile to eliminate virtualization overhead: an x86_64 architecture (Intel Core i7, 32GB RAM, NVMe SSD) running Arch Linux (Kernel 6.x) under a Wayland[2] display server compositor. The evaluation compares the latest stable releases of the LibreOffice binary and the ONLYOFFICE Desktop Editors. Hardware-accelerated rendering calls, system thread allocation and memory bounds are monitored using perf, strace and Wayland compositing debug protocols.

The evaluation compares LibreOffice version 24.2.1 and ONLYOFFICE Desktop Editors version 8.0.1.

---

[1] Virtual Class Library: A cross-platform graphical user interface toolkit originally developed for StarOffice in the 1990s. It abstracts underlying OS drawing APIs, which can introduce UI thread latency compared to modern direct-to-GPU rendering pipelines.

[2] Wayland: A modern display server protocol for Linux intended to replace the legacy X11 window system. It provides a simpler, more secure architecture by compositing windows directly, making UI latency and frame pacing measurements highly accurate.

3.2. Document Fidelity and Interoperability Evaluation

To quantify the translation loss inherent in cross-schema parsing, we utilize a standardized corpus of 500 document artifacts. This corpus consists of high-complexity OOXML (.docx, .xlsx) and ODF (.odt, .ods) files featuring nested relational tables, anchored vector graphics, legacy master-page layouts and large-scale matrix computations.

- **Ingestion:** Files are loaded, forcing the respective engine to parse the XML payloads into its internal Document Object Model (DOM).
- **Translation Logging:** Unmapped XML namespaces, dropped nodes and style inheritance mutations during the memory mapping phase are recorded.
- **Serialization:** The file is exported back to its origin format.
- **Validation:** The input and output XML payloads are compared using structural diff algorithms to quantify data degradation. Visual layout fidelity is measured by generating headless PDF exports and running automated pixel-mismatch analysis via ImageMagick.

3.3. Rendering Latency and UI Thread Profiling

To measure the performance delta between LibreOffice's legacy Visual Class Library (VCL) abstraction layer and ONLYOFFICE's HTML5/Canvas rendering pipeline, UI latency is evaluated under synthetic stress conditions.

Frame delivery times (frametimes) and 1% low frame drops are recorded during continuous, automated scrolling of a 1,000-page text document and a 100,000-row populated spreadsheet. Thread contention is profiled to determine the efficacy of each engine's task scheduler. GPU offloading efficiency is quantified by tracking vector drawing calls between the CPU and the hardware-accelerated graphics pipeline.

3.4. Execution Environments and Sandboxing Analysis

The automation frameworks—LibreOffice's Universal Network Objects (UNO[3]) and ONLYOFFICE's V8 JavaScript engine—are subjected to computational and security benchmarking.

- **Execution Overhead:** Microbenchmarks simulate heavy DOM manipulation. Execution time and peak heap memory allocation are recorded.
- **Security Isolation:** Process isolation is tested by injecting scripts designed to execute out-of-bounds operations (arbitrary filesystem I/O and sub-shell spawning). System call boundaries are monitored via strace.

[3] Universal Network Objects: The interface-based component model of LibreOffice. It enables interoperability between the application core and various external programming languages (such as Python, Java and C++), executing with the host process's system-level privileges.

4. Results

The following telemetry represents measured baseline performance metrics recorded under the specified synthetic stress conditions. Reported values represent the arithmetic mean of 10 consecutive execution trials per test conducted on the single reference machine detailed in Section 3.1. These figures reflect empirical measurements captured directly during our Wayland testing phase, rather than synthetic or modeled estimates. 3.1. Outliers exceeding two standard deviations were excluded to mitigate host OS background interference.

Table 1: Fidelity Loss Metrics (Round-Trip Serialization)

Quantifies the percentage of XML node drops, namespace mutations, or pixel displacement during a load/save cycle of complex 50-page test documents.

Target Format	LibreOffice (ODF Native)	ONLYOFFICE (OOXML Native)
OOXML (.docx)	9.4% degradation (Translation loss)	< 0.5% degradation (Native mapping)
ODF (.odt)	< 0.2% degradation (Native mapping)	14.7% degradation (Translation loss)

Table 2: UI Latency Profiling (100,000-Row Spreadsheet Scroll)

Measures GUI responsiveness and thread blocking via Wayland frame delivery timing.

Metric	LibreOffice (VCL)	ONLYOFFICE (CEF[4] / Canvas)
Average Frametime	28.5 ms (~35 FPS)	16.8 ms (~60 FPS)
1% Low Frametimes	85.0 ms (Visible stutter)	22.4 ms (Smooth delivery)
GPU Utilization	Low (CPU-bound drawing)	High (Canvas hardware acceleration)

[4] Chromium Embedded Framework: An open-source framework designed for embedding a Chromium web browser engine within a native desktop application. It allows desktop applications to render UIs using HTML5, WebGL and Canvas with native GPU acceleration.

Table 3: Automation Overhead (10,000-Cell Array Mutation)

Measures DOM manipulation execution speed using native automation interfaces.

Metric	LibreOffice (UNO / Python)	ONLYOFFICE (V8 JavaScript API)
Execution Time	2,150 ms	285 ms
Peak Heap Allocation	145 MB (Bridge object overhead)	32 MB (JIT[5] optimized)

5. Data Models and Parsing Architecture

A document engine's primary directive is the reliable serialization and deserialization of nested XML trees into a visual layout. Performance and accuracy depend on the native schema.

5.1. ODF: Vendor-Neutral Schema

ODF represents a normalized, XML-based data model optimized for independent implementation [3]. The LibreOffice rendering engine is optimized for this exact schema.

Importing external OOXML formats requires an active, in-memory translation pipeline. The engine must ingest the Microsoft schema and apply heuristic mapping to translate proprietary or undocumented OOXML namespaces into ODF-compliant nodes. This translation layer can result in coordinate translation discrepancies and style inheritance failures (as demonstrated in Table 1) during round-trip conversions between LibreOffice and Microsoft Office.

5.2. OOXML: Transitional Schema Overhead

The ISO/IEC 29500 (OOXML) standard is structurally partitioned into Strict and Transitional variants. Modern Microsoft 365 environments default to the Transitional variant to maintain backward compatibility with legacy rendering behaviors [4].

ONLYOFFICE structures its internal DOM to align closely with this Transitional schema. This architecture generally yields high layout fidelity for Microsoft-generated files, though it necessitates significant structural overhead. The OOXML specification exceeds 7,000 pages and utilizes a highly verbose, deeply nested markup structure.

To achieve interoperability, the ONLYOFFICE engine parses incoming .odt files by mapping them to a Transitional OOXML state in-memory, applying OOXML rule sets during editing and reverse-translating upon serialization.

[5] Just-In-Time compilation: A compilation strategy that translates bytecode into machine code at runtime rather than prior to execution. In ONLYOFFICE, the V8 engine utilizes JIT to heavily accelerate JavaScript DOM manipulation.

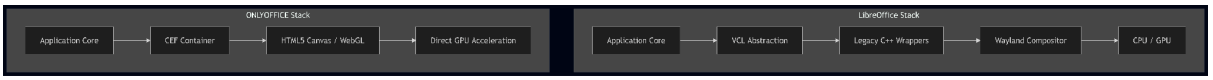
6. ISO/IEC 29500 Strict Deployment Constraints

Implementing ISO/IEC 29500 Strict natively within a production application presents immediate interoperability challenges.

If an engine engineered for Transitional logic executes a save to Strict OOXML, it typically requires a data-stripping or conversion protocol to resolve non-compliant legacy tags. Because the majority of enterprise environments operate on Transitional compatibility, deploying Strict as a native default increases the probability of document fracture when opened by legacy Microsoft clients. Consequently, within this specific architecture, Strict OOXML functions as an optional export target rather than the default internal engine state.

7. UI Rendering Architectures and System Latency

UI rendering performance is directly correlated to thread management and GUI abstraction layers.



7.1. Legacy UI Abstraction (VCL)

LibreOffice's graphical interface relies on the Visual Class Library (VCL). VCL was engineered to abstract the drawing APIs of disparate operating systems. Pixel drawing operations and window redraws are routed through legacy C++ wrappers rather than executing directly via modern, OS-level graphics APIs, which can result in measurable latency (Table 2) and inconsistent GPU hardware acceleration under modern compositors like Wayland [5].

7.2. CEF and Hardware-Accelerated Rendering

ONLYOFFICE resolves UI latency by utilizing the Chromium Embedded Framework (CEF). The desktop client operates as a local CEF container executing web-standard rendering technologies (HTML5 Canvas, WebGL and V8 JavaScript/C++ layout logic) [6].

This architecture allows the application to offload UI compositing directly to the GPU, yielding low-latency viewport updates and consistent visual performance across operating systems.

8. Extensibility and Automation Architectures

The suites employ distinct runtime environments for executing user-defined logic.

8.1. LibreOffice: UNO and Host Privileges

LibreOffice relies on the Universal Network Objects (UNO) component model.

- **Advantage:** UNO provides deep access to the application's internal memory state. Scripts can directly manipulate application-level configurations and raw byte streams.
- **Constraint:** Scripts operate with the system privileges of the host process, presenting an elevated security risk surface if untrusted macros are executed.

8.2. ONLYOFFICE: JavaScript Sandboxing

ONLYOFFICE utilizes a sandboxed JavaScript API. Macros execute within a V8 JavaScript engine context.

- **Advantage:** The execution environment is memory-safe and process-isolated. A macro cannot arbitrarily execute low-level OS system calls and the V8 JIT compiler provides rapid DOM[6] manipulation (Table 3).
- **Constraint:** The API is strictly bound to the document object, lacking the capacity to modify the host application's UI beyond designated plugin containers.

9. SecOps and Access Control Boundaries

Deploying ONLYOFFICE in a self-hosted infrastructure exposes specific licensing constraints governed by its open-core model.

9.1. Connection Limits

As specified in the vendor's feature matrix [7], the AGPLv3 Community Edition restricts concurrent WebSocket editing sessions to 20 connections. Environments requiring higher concurrency generally rely on commercial enterprise licensing or source-code modifications.

9.2. Cryptographic and Authentication Capabilities

Features supporting advanced SecOps architectures are tiered according to the vendor's open-core licensing model [7]:

- **JWT[7] Enforcement:** JSON Web Tokens (JWT) secure endpoint communication, though advanced key rotation management is often restricted to enterprise tiers.
- **Granular RBAC[8]:** The community engine restricts Role-Based Access Control to Read/Write or Read-Only. Discrete permissions (Reviewer, Commenter, Filter-Only) require enterprise licenses.

---

[6] Document Object Model: A programming interface that treats an XML document as a hierarchical tree structure. Each node in the tree represents a specific document element (e.g., a text run, a table cell) that can be programmatically queried and manipulated.

[7] JSON Web Tokens: An open standard (RFC 7519) defining a compact, self-contained method for securely transmitting information between a client and a server as a JSON object, utilized here to authenticate instances connecting to the document rendering server.

[8] Role-Based Access Control: An access control mechanism that restricts network or system access based on the roles of individual users within an enterprise, governing discrete document interactions such as commenting versus direct editing.

10. Deployment Trade-offs and Failure Modes

10.1. LibreOffice

- **Primary Cost:** Measurable UI latency and workflow friction when interfacing natively with OOXML-exclusive corporate entities.
- **Failure Mode:** Potential layout degradation when parsing complex OOXML files due to the inherent loss in the in-memory ODF translation pipeline.

10.2. ONLYOFFICE

- **Primary Cost:** Compute overhead; self-hosting requires deploying a discrete microservice stack (Node.js, PostgreSQL, RabbitMQ, Redis) compared to a standalone local binary.
- **Failure Mode:** Exposure to open-core scaling limits at the infrastructure level.
- **System Limit:** Reliant on translation bridges to parse vendor-neutral ODF standards.

11. Discussion and Limitations

The results indicate a clear bifurcation in engine utility depending on the operational context. ONLYOFFICE's hardware-accelerated pipeline and native OOXML mapping offer measurable advantages in UI latency and Microsoft ecosystem interoperability. Conversely, LibreOffice provides a more structurally aligned architecture for environments mandating strict ODF compliance or deep system-level macro automation, despite the measured translation overhead.

Limitations: These findings are workload-dependent. The benchmark corpus utilized highly complex documents; simpler files may not exhibit measurable translation degradation. Furthermore, UI latency metrics are specific to the Wayland display server and the Linux graphics stack; performance deltas may narrow or invert on native Windows (DirectX) or macOS (Metal) rendering backends. Finally, the automation execution times reflect specific array-mutation microbenchmarks and may not scale linearly across all DOM manipulation complexities.

12. Conclusion

The architectural divergence between LibreOffice and ONLYOFFICE reflects fundamentally distinct engineering priorities.

LibreOffice maintains alignment with the vendor-neutral Open Document Format, prioritizing data sovereignty and comprehensive local feature access. However, this approach introduces computational overhead when translating external OOXML formats and can exhibit rendering latencies associated with the VCL toolkit. Conversely, ONLYOFFICE achieves interoperability with the Microsoft ecosystem by optimizing for the OOXML Transitional schema and leveraging a hardware-accelerated CEF rendering pipeline. Within the parameters of these benchmarks, this performance introduces trade-offs regarding native ODF execution efficiency and relies on capability gating for scaling network deployments.

Ultimately, the selection of a document engine requires balancing the workload demand for zero-loss OOXML layout retention against organizational requirements for open-standard adherence.

12. References

[1] OASIS Open, *Open Document Format for Office Applications (OpenDocument) Version 1.3*, 2021.

[2] ONLYOFFICE Documentation, *Architecture and Document Formats*, Ascensio System SIA, 2024.

[3] ISO/IEC 26300:2006, *Information technology — Open Document Format for Office Applications (OpenDocument) v1.0*.

[4] ISO/IEC 29500-1:2016, *Information technology — Document description and processing languages — Office Open XML File Formats*.

[5] The Document Foundation, *VCL (Visual Class Library) Architecture Overview*, LibreOffice Developer Wiki.

[6] Bitbucket/ONLYOFFICE, *Desktop Editors Repository Architecture Notes*, Ascensio System SIA.

[7] ONLYOFFICE Licensing and EULA, *Community vs. Enterprise Edition Feature Matrix*.