Metaversalize

www.meta.reapress.com

Metaverse. Vol. 3, No. 1 (2026) 22–30.

Paper Type: Original Article

Architecting the Spatial Web: A Review of the Sneeze Metaverse Browser Engine and Multi-Origin Scene Composition

Zeynab Parandavar*

Department of Computer Engineering, Ayandegan University, Tonekabon, Iran; z.parand.1997@gmail.com.

Citation:

Received: 06 August 2025 *Revised: 23 October 2025* *Accepted: 21 December 2025*	Parandavar, Z. (2026). Architecting the spatial web: A review of the sneeze metaverse browser engine and multi-origin scene composition. *Metaversalize, 3*(1), 22-30.

Abstract

This technical report examines the architectural case for a dedicated Metaverse Browser Engine (MBE), as put forward by the Open Metaverse Browser Initiative (OMBI), an open-source, vendor-neutral testbed hosted under the Metaverse Standards Forum (MSF). The report is not original empirical research; it is a structured synthesis and critical discussion of publicly disclosed architectural material presented by MSF/OMBI and its RP1 prototyping effort at a W3C Immersive Web Community Group discussion held in New York in May 2026, released under a Creative Commons Attribution 4.0 International license. The purpose of this report is to make that material accessible to a broader research audience and to situate it within the wider immersive-web and spatial-computing standardization landscape. The report first identifies why current web and WebXR browser stacks lack a browser-level model for multi-origin spatial scene composition, automatic geolocated service discovery, persistent shared geospatial anchoring, and a standard live 3D service abstraction. It then describes the proposed architecture, centered on Sneeze, an open-source MBE intended to serve the metaverse stack in a role comparable to that of WebKit or Blink for the two-dimensional web. Sneeze is built around a Scene Object Model (SOM) as its central scene graph, a Remote Model Access Protocol (RMAP) for synchronizing service state with clients, WebAssembly-sandboxed per-service execution, and a certificate-based, organization-scoped trust model for multi-origin content. Four architectural patterns for composing spatial (Sneeze) and conventional web (Blink) rendering are reviewed, together with open standardization gaps in geospatial pose and anchoring interoperability. The report concludes that a dedicated, complementary spatial browser engine is a technically motivated and plausible path toward an open, decentralized spatial web, while noting that the architecture remains at an early, testbed stage of maturity.

Keywords: Metaverse browser engine, WebXR, Spatial computing, Multi-origin scene composition, WebAssembly, Open metaverse browser initiative.

1 | Introduction

The convergence of computer vision, sensor fusion, and wearable computing is producing a new category of "spatial web" client: AR glasses, phones, and headsets that anchor digital content to physical locations,

Corresponding Author: z.parand.1997@gmail.com

https://doi.org/10.22105/metaverse.v3i1.98

alongside autonomous systems such as vehicles, drones, and robots that consume spatial services to navigate and act in the physical world [1]. Both classes of client share a common requirement: the ability to draw on multiple independent, geolocated services from many different origins and compose them into a single, coherent three-dimensional scene.

The two-dimensional web solved an analogous problem for documents decades ago through URLs, hyperlinks, DNS, and a standard document object model. The spatial web has, so far, no equivalent. WebXR provides a mechanism for a web page to render immersive content [2], [3]. Still, it does not provide a browsing model: it offers a surface on which pixels can be displayed, without addressing discovery, composition, or multi-origin trust for spatial services. This gap is the starting point for the Open Metaverse Browser Initiative (OMBI) [4], a collaborative, vendor-neutral, open-source testbed hosted at the Metaverse Standards Forum (MSF) [5], whose stated purpose is to build consensus on the need for spatial service composition, to explore deployment of spatial services using current web technologies, and to identify where existing web standards fall short.

This report synthesizes and discusses a technical presentation released by MSF/OMBI, prepared in connection with RP1's earlier prototyping work on a global, WebXR-based metaverse browser (a project supporting on the order of 100,000 concurrent users with full six-degrees-of-freedom tracking and spatial audio, spanning an endless, earth-scale map). That prototyping effort is presented as the empirical motivation for a subsequent architectural proposal: a purpose-built Metaverse Browser Engine (MBE), exemplified by an open-source implementation named Sneeze, which was contributed to OMBI in March 2026 under the Apache 2.0 license.

The remainder of this report is organized as follows. Section 2 describes the source material and the method used to structure this report. Section 3 presents the architecture under review, including the requirements it addresses, the Sneeze engine, the Scene Object Model (SOM), the Remote Model Access Protocol (RMAP), WebAssembly sandboxing, certificate-based multi-origin security, geopositioning, and patterns for combining spatial and conventional web rendering. Section 4 discusses the implications and open standardization questions. Section 5 concludes.

2 | Methodology

This report is a desk-based technical analysis rather than original empirical research. Its primary source is a single technical presentation, "the case for a MBE," disclosed by the MSF in connection with the OMBI and RP1, and presented at a W3C Immersive Web Community Group discussion in New York in May 2026 [6]. That material is released under a Creative Commons Attribution 4.0 International license, which permits reuse, adaptation, and redistribution with appropriate attribution; this report relies on that permission and cites the source accordingly rather than presenting the architecture as original work of the present author.

The author read the source presentation in full, extracted its architectural claims and diagrams, and reorganized them into the standard structure expected of a technical report (introduction, architecture description, discussion, conclusion), converting slide-level tables into report tables and slide-level diagrams into original schematic figures redrawn for this report. Where the source material referenced external standards, the W3C WebXR Device API, the OGC GeoPose 1.0 Data Exchange Standard, Apple's ARKit geo-anchoring, and Google's ARCore Geospatial API, those references were cross-checked against the relevant public standards documentation to verify that the characterization in the source material is consistent with current public documentation [7–10].

This report does not include independent implementation, benchmarking, or performance testing of the Sneeze engine or the APOLLO browser referenced in the source material; all performance and scale figures reported below (e.g., concurrency and frame-rate figures) are as disclosed in the source presentation and have not been independently verified by the author. The architecture described reflects the state of the

OMBI/Sneeze project as of the cited presentation (May 2026) and should be read as a snapshot of an early-stage, evolving open-source testbed rather than a finalized specification.

3 | Architecture Overview

3.1 | The Case for Multi-Origin Spatial Service Composition

The source material frames the core problem through a concrete scenario: a pedestrian wearing AR glasses encounters, within a few steps, a restaurant-discovery overlay anchored to a building facade, a wayfinding overlay anchored to street geometry, and a ride-status overlay anchored to a moving vehicle, three independent services, run by three independent businesses, that must be composed into one coherent scene. No single origin owns this experience, and, critically, no single origin can build it alone: each service must run in its own security context, and the browser must compose the outputs of these services without needing to compose or share trust between them.

This requirement, many independent, geolocated services contributing to a single 3D scene around a moving user, is presented as structurally different from what today's web and WebXR stack were designed to do, and is used to motivate a dedicated engine rather than incremental extension of the existing browser rendering pipeline.

3.2 | High-Level Architecture: The Metaverse Browser Engine

The proposed high-level architecture (*Fig. 1*) places a MBE at the client, composed of five cooperating subsystems, geoposition Visual Positioning System/Global Positioning System (VPS/GPS), code execution, rendering abstraction, an OpenXR-based device abstraction, and shader execution, all operating over a central SOM, which functions as the spatial analogue of the DOM in a conventional web browser. Below the MBE, a RMAP synchronizes networked state between the client and independently hosted Spatial Fabrics, servers that expose assets and services, analogous to websites in the two-dimensional web.

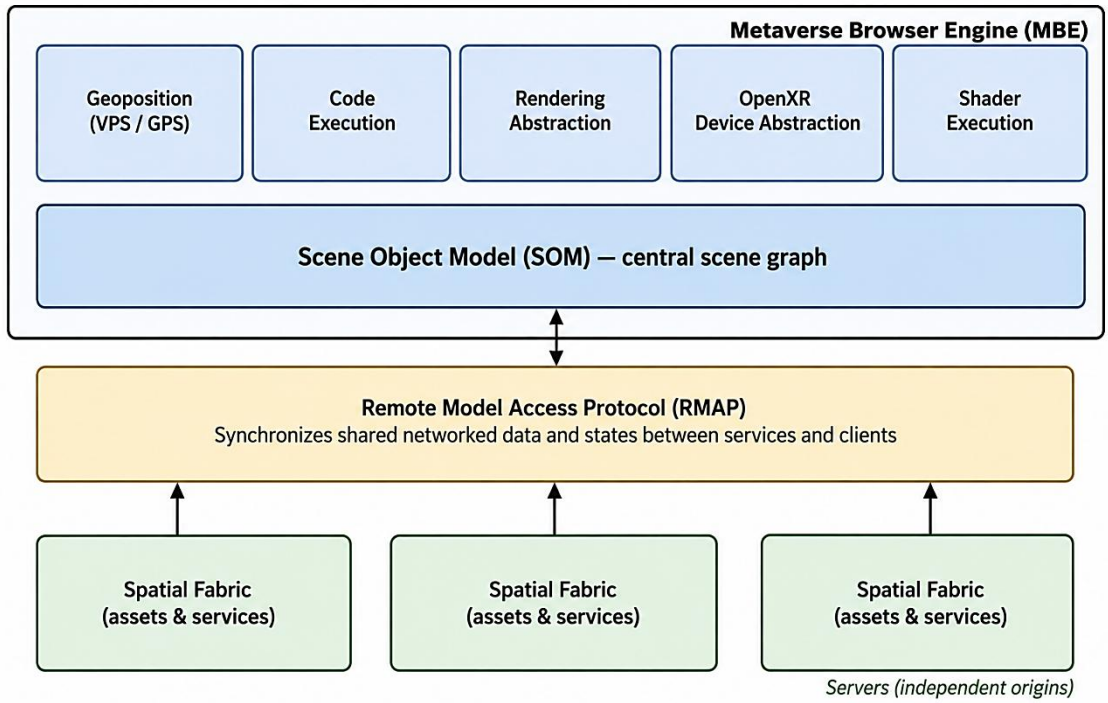

Fig. 1. High-level architecture of the MBE: device-side subsystems over a shared SOM, an RMAP synchronization layer, and independently hosted Spatial Fabrics. Redrawn by the author from the architecture described in [6].

3.3 | Sneeze: An Open-Source Metaverse Browser Engine

Sneeze is described in the source material as a MBE for the metaverse stack, occupying a role comparable to WebKit or Blink for the conventional web-browser stack. Rather than parsing HTML, CSS, and JavaScript into a DOM, Sneeze ingests 3D models and shaders from services and builds a SOM. *Table 1* summarizes the functional correspondence drawn between a conventional web browser engine and Sneeze.

Table 1. Functional correspondence between a conventional web browser engine and the Sneeze spatial engine, as presented in [6].

Function	Web Browser Engine	Sneeze Spatial Engine
Content ingestion	Fetch HTML/CSS from servers	Receive 3D models and shaders from services
Scene composition	Build the DOM	Build the Scene Object Model
Rendering	Layout and paint	Delegate to an abstracted 3D renderer
Script execution	JavaScript (e.g., V8)	WebAssembly, sandboxed runtime
Input handling	Mouse and keyboard	Spatial input via OpenXR

According to the source material, Sneeze is implemented in C++, does not implement its own renderer but delegates rendering through a graphics abstraction layer, and was open-sourced in March 2026 and contributed to OMBI, where it serves as an exemplar (reference) implementation of a MBE rather than the only possible one.

3.4 | Remote Model Access Protocol

A recurring theme in the source material is that conventional service-integration protocols (REST, GraphQL, gRPC, WebSockets, MQTT) require every client developer to write bespoke integration code against every service, that this integration cost scales per client, and that there is no single standard way to implement or document real-time services. RMAP is presented as an inversion of this model: rather than the client writing an adapter against a service's API, the service itself ships a typed RMAP model definition that runs inside the client's WebAssembly store alongside the client's own code. The client then interacts only with these models, which are inherently discoverable and exposed through one uniform action/event mechanism — without needing prior documentation of the underlying service protocol.

The source material argues that this inversion has several downstream consequences: RMAP is runtime-agnostic and portable to any language or platform, not only metaverse browsers; it sidesteps Cross-Origin Resource Sharing (CORS) concerns because there is no cross-origin fetch surface left for a browser to police; bug fixes can propagate through service-side updates without requiring client rebuilds; and a service can change its underlying wire protocol without breaking existing clients.

3.5 | Scene Object Model and Multi-Origin Composition

The SOM is presented as the mechanism through which content from many independent, differently-originated services is composed into a single scene. In the pedestrian scenario introduced in Section 3.1, three services, discovery, wayfinding, and ride status, each ship a sandboxed WebAssembly program that Sneeze loads on first connection. RMAP delivers model updates into each service's own WASM instance, and that instance is permitted to write only into its own, isolated branch of the SOM (*Fig. 2*). Services may read from unprotected portions of the SOM for contextual awareness, but write-branch ownership and verified service identity are used to prevent one service from mutating another's content.

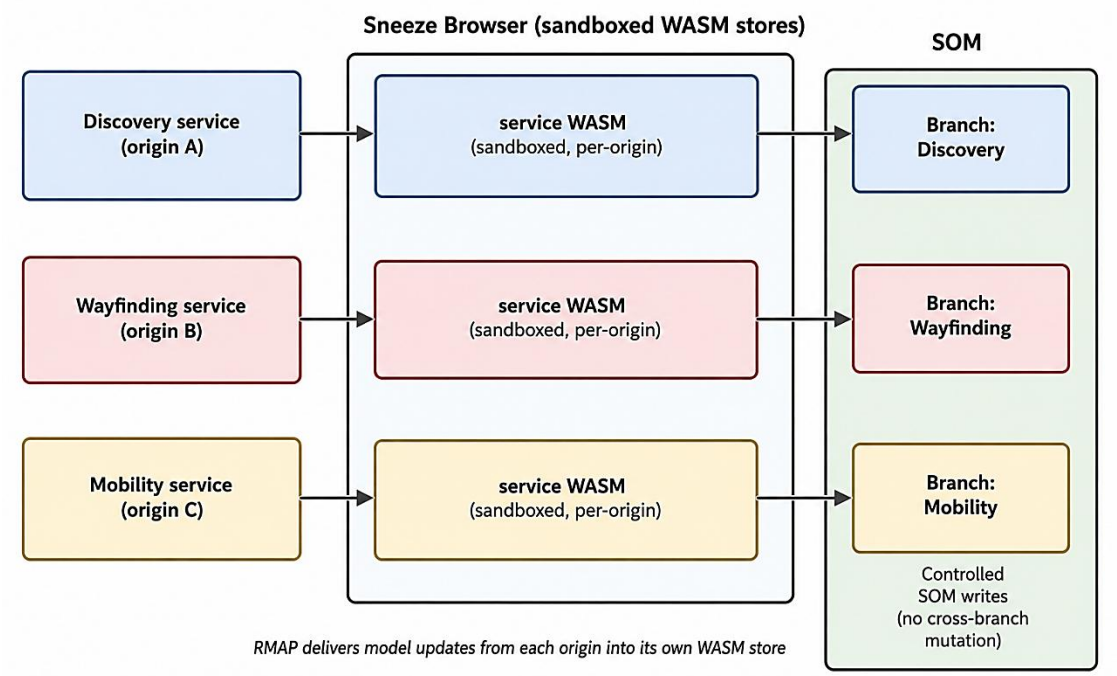

Fig. 2. Multi-origin composition in the SOM: each service's sandboxed WebAssembly instance writes only into its own scene branch. Redrawn by the author from the architecture described in [6].

The stated outcome is a single, browser-composed scene assembled from many independent providers without requiring a per-service application install, while each provider retains the ability to evolve its own backend protocol independently within its sandboxed store.

3.6 | WebAssembly as the Sandboxing and Execution Model

The source material argues for WebAssembly (WASM), rather than JavaScript, as the execution substrate for third-party service logic inside the browser engine, citing five considerations: stronger per-service sandboxing for third-party code; no dependency on a JavaScript engine inside the native browser core, since WASM modules can run directly in an embedded native runtime; language flexibility, since services can be authored in C, C++, Rust, or any other language that compiles to WASM; finer-grained performance control, including CPU budgeting and interruption per service; and a portable execution model consistent with a multi-platform, standards-based architecture. Because Sneeze itself is built around a native C++ scene model rather than a JavaScript runtime, and because real-time engine workloads are argued to be less well suited to JavaScript's execution characteristics than to native code paired with WASM, the source material positions WASM plus native code as a better fit for the engine's sandboxing needs than a JavaScript-based approach.

3.7 | Certificate-Based Security for Multi-Origin Content

The web's trust model ties permissions and isolation to DNS-based origins (domain, scheme, and port), with cross-origin access mediated through exceptions such as CORS; this couples trust to web infrastructure rather than to a portable cryptographic identity. The proposed alternative anchors trust to organizational, certificate-based identity instead. A Metaverse Spatial Fabric (MSF) file is described as a JSON structure signed with JWS, embedding the full X.509 certificate chain of the signer; the signing organization's identity is derived as the SHA-256 hash of the signing certificate's public key, a value that survives certificate renewal and is independently verifiable by any party.

Under this model, grouped services run inside a WASM store scoped to the organization's certificate fingerprint, with isolated memory, CPU budgets, and controlled host-API access; persistent storage is scoped to both the individual spatial fabric and to its owning organization, so that one organization cannot read another's stored data. The browser verifies the signature and certificate chain before trusting any content, and users are able to filter out untrusted, unverified, or unwanted organizations and content categories using

standardized content tags (for example, advertisement, adult, game, point-of-sale, navigation, safety, or entertainment). The stated result is that multiple origins can coexist by design, each cryptographically isolated, without reliance on CORS or DNS-based domain coupling.

3.8 | Geopositioning and Location Awareness

The MBE architecture combines GPS/VPS positioning with OpenXR tracking to ground the scene in the physical world and maintains a live SOM of nearby Spatial Fabrics, services, and objects so that content can be activated automatically based on proximity. As the user moves, the engine is described as continuously updating the scene through proximity queries, streaming, and coordinate transforms, while treating VPS access as a pluggable, replaceable positioning backend rather than a fixed dependency on any single provider. Location and input handling are described as browser-mediated, with fabric-level ownership of VPS data and permission controls placed around sensitive positioning signals.

A representative use case in the source material walks through this lifecycle: the user opens the browser within a specified primary spatial fabric; GPS/VPS positioning locates the user within the SOM; the browser fetches proximal map nodes, via RMAP, from the map services of already-loaded fabrics; incoming map nodes identify secondary Spatial Fabrics and services, which are loaded into the SOM; nodes that fall outside a relevance radius are unloaded; and the SOM composes the resulting multi-origin scene. The stated outcome is that place-aware services are rendered and anchored to the user's physical surroundings without the user needing to manually discover or install each one.

3.9 | Composing Spatial and Conventional Web Rendering

Because a spatial engine such as Sneeze and a conventional web engine such as Blink are expected to coexist rather than one replacing the other, the source material sets out four architectural patterns for combining them, summarized in *Table 2*.

Table 2. Four architectural patterns for composing a spatial engine (Sneeze) with a web engine (Blink), as presented in [6].

Pattern	Mechanism	Representative Use Case	Main Limitation
A: Full-viewport handoff	The browser switches between a Blink-rendered page and a Sneeze-rendered spatial fabric.	Opening a spatial fabric from a web link, analogous to opening a PDF or an immersive document.	No simultaneous composition; requires navigation, permission, identity, and lifecycle handoff between the two engines.
B: Spatial island inside a web page	Blink embeds an opaque Sneeze viewport as a rectangular region within a normal web page.	Embedding a live 3D spatial service, preview, or mini-world inside an existing website.	The 3D scene remains boxed inside the page; Blink cannot directly access or manipulate the SOM.
C: Shared compositor/sibling surfaces	Blink and Sneeze each render independent surfaces combined by a shared, OS- or browser-level compositor.	Browser chrome, dialogs, permission prompts, side panels, and mixed UI layered around a spatial scene.	Only simple rectangular composition; true 3D integration (depth, pose, hit-testing, occlusion) is not shared unless explicitly implemented.
D: Web surface inside the Sneeze 3D scene	Blink renders live web content to a surface or texture that Sneeze places as an object inside the SOM.	Web panels, dashboards, forms, commerce flows, maps, or documents embedded as objects with real 3D perspective, occlusion, and lighting.	Requires careful input routing, focus management, permissions, and depth/occlusion handling; the web page is initially treated as a single rectangle rather than as individually addressable DOM nodes inside the SOM.

The source material notes that pattern D, a live web surface placed as an object inside the 3D scene, illustrated in the material through a factory-maintenance digital-twin example in which a 2D work-order ticket is rendered as a panel beside the physical equipment it describes, requires more than simple compositor-level (pattern C) composition to support properly, since it needs coordinated handling of depth, occlusion, lighting, and cross-engine input routing rather than the aggregation of independent rectangular surfaces.

4 | Discussion

The architecture reviewed in Section 3 is best understood as a response to five specific, structural gaps in the current web and WebXR stack, rather than as a general critique of the web. First, the web defines strong cross-origin isolation but no browser-level model for multiple origins to contribute governed branches into one shared spatial scene graph; WebXR provides a rendering surface, not multi-origin spatial composition. Second, although the web already provides URLs, DNS, and geolocation APIs, there is no interoperable, browser-mediated lifecycle for discovering, authorizing, loading, composing, and unloading nearby spatial services as a user moves, the "ten-foot rule" problem of not wanting to install a new application every few steps. Third, geolocation and WebXR anchors exist independently, but there is no widely deployed browser primitive for persistent, shared, six-degrees-of-freedom geospatial anchors that independent services can agree on. Fourth, there is no standard abstraction for a URL-addressable, live 3D service exposing properties, events, actions, subscriptions, and permissions to a composing client, comparable to what RMAP proposes. Fifth, immersive WebXR sessions do not support DOM overlays, popups, or cross-origin trust UI in a consistent way, which constrains web-style authentication, payment, or permission flows from inside an immersive session.

These five gaps, together with device-side constraints around memory budgets, compute capacity, frame-rate consistency, and the absence of a browser-safe, speed-first datagram protocol comparable to raw UDP (WebTransport is noted as a partial, browser-safe substitute), form the technical case for a purpose-built engine rather than incremental extension of existing web APIs. This case is, however, made by the same organizations proposing the alternative, and the report itself should be read with that provenance in mind: it is disclosed, structured, industry material rather than independently peer-reviewed research, and its performance and scale claims (for example, six-degrees-of-freedom co-presence for over 100,000 concurrent users) are asserted rather than demonstrated in the source presentation available to this author.

From a standardization perspective, the source material's own appendix on geolocation interoperability is instructive: it lists existing, largely non-interoperable building blocks, the W3C Geolocation API, the WebXR Device API, an incubating WebXR geo-alignment reference space, the OGC GeoPose 1.0 Data Exchange Standard, Apple's ARKit geo-anchoring, and Google's ARCore Geospatial API, and proposes four collaborative actions across standards bodies: extending Geolocation toward a continuous geospatial pose API; advancing a geospatially aligned WebXR reference space; adopting a GeoPose-based wire serialization between client, runtime, and spatial fabric; and exploring an OpenXR extension binding device poses to OGC coordinate-reference-system semantics. *Table 3* summarizes this landscape.

Table 3. Summary of existing geolocation-related standards and platform APIs and the interoperability gap each leaves open, as characterized in [6].

Existing Building Block	Owning Body	Gap Relative to a Full Geospatial Pose/Anchor Interface
Geolocation API	W3C DAS/WebApps	Provides latitude/longitude and related data, but no six-degrees-of-freedom pose or XR-frame synchronization.
WebXR device API	W3C Immersive Web WG	Provides continuous XR poses in local reference spaces, with no standard binding to a geospatial or world coordinate reference system.
WebXR geo-alignment/geospatial reference space	W3C Immersive Web (incubation)	Would align WebXR local reference spaces to geospatial frames, but remains an incubating proposal without renewed championing.
GeoPose 1.0 data exchange standard	OGC	Defines location and orientation of real/virtual objects in Earth-anchored frames, but is not yet adopted as a wire format between clients, runtimes, and Spatial Fabrics.
ARKit ARGeoAnchor/ARCore geospatial API	Apple/Google	Provide shipping, VPS-backed geographic anchors, but are platform-specific rather than cross-vendor.

A further point worth surfacing in discussion is governance: the source material situates the open-source Sneeze project inside a layered governance model, in which MSF working groups (3D assets, spatial computing, economy and trust, and governance and society) provide domain input and may use OMBI as a testbed, while a technical steering committee, appointed maintainers, and contributors hold day-to-day technical authority over the codebase under an Apache 2.0 license. This is a conventional open-source governance structure, and its presence does not by itself resolve the deeper standardization questions raised above; it does, however, indicate that the proposal is intended as a testbed for building cross-organization consensus rather than as a single-vendor product.

For researchers working on adjacent problems in cooperative and spatial computing, for example, trust and confidence evaluation in Vehicle-to-Everything (V2X) coordination messages, or multi-sensor cooperative perception, the architectural pattern most directly relevant is the certificate-based, organization-scoped trust model described in Section 3.7, together with the RMAP model-synchronization pattern described in Section 3.4: both address, in a spatial-web context, the same general problem of establishing verifiable trust and low-friction interoperability between independently operated origins that must share a common operational picture in real time.

5 | Conclusion

This report has synthesized publicly disclosed, Creative Commons–licensed material from the MSF's OMBI describing the architectural case for a dedicated MBE. The reviewed architecture, centered on the Sneeze engine, a SOM, a RMAP, WebAssembly-sandboxed service execution, and certificate-based multi-origin trust, is presented as a response to specific, well-defined gaps in the current web and WebXR stack around multi-origin scene composition, automatic geolocated service discovery, persistent shared geospatial anchoring, and standardized live-service access, rather than as a wholesale replacement for the web browser.

Four patterns for combining spatial and conventional web rendering, and a concrete four-item cross-standards-body roadmap for geospatial pose interoperability, indicate that the proposal is intended to complement, rather than displace, existing web and immersive-web standards. At the same time, the architecture remains an early-stage, single-organization-led testbed whose performance claims have not been independently verified, and whose governance and standardization outcomes will depend on the extent to which other standards bodies and industry participants engage with the OMBI process going forward. Future work, from this author's perspective, would benefit from independent implementation and benchmarking of the SOM/RMAP model against the requirements set out in Section 3.1, and from closer examination of how

the certificate-based trust model described in Section 3.7 might generalize to other multi-origin, real-time coordination problems in spatial and cooperative computing.

Acknowledgments

The author gratefully acknowledges the MSF, the OMBI, and RP1 for disclosing and licensing the source technical presentation under a Creative Commons Attribution 4.0 International license, which made this report possible.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data Availability

The primary source material analyzed in this report is publicly available at https://omb.wiki/sneeze and https://omb.wiki/. No new empirical data were generated during this study.

Conflicts of Interest

The author has no financial or non-financial interests to disclose and no affiliation with the MSF, the OMBI, or RP1.

Use of Artificial Intelligence

An AI language model assistant (Claude, Anthropic) was used to help structure and draft this report from the source presentation identified in Section 2, including reorganizing slide content into standard report sections, converting slide tables into report tables, and producing the schematic figures in *Figs. 1* and *2* as original redrawings of the source architecture. The author reviewed, verified, and takes full responsibility for the accuracy, integrity, and originality of the content of this manuscript.

References

[1] Mourtzis, D. (2025). Industrial metaverse towards human-centric smart manufacturing. In *Human-Centric Smart Manufacturing Towards Industry 5.0* (pp. 3–44). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-82170-7_1

[2] Xing, Y., Shell, J., Fahy, C., Xie, T., Kwan, H. Y., & Xie, W. (2022). Web XR user interface research: Design 3D layout framework in static websites. *Applied Sciences*, *12*(11), 5600. https://doi.org/10.3390/app12115600

[3] Martí-Testón, A., Muñoz, A., Gracia, L., & Solanes, J. E. (2023). Using WebXR metaverse platforms to create touristic services and cultural promotion. *Applied Sciences*, *13*(14), 8544. https://doi.org/10.3390/app13148544

[4] Metaverse Standards Forum. (2026). *Open metaverse browser initiative (OMBI)*. https://metaverse-standards.org/open-metaverse-browser-initiative/

[5] Metaverse Standards Forum. (2026). *The metaverse standards forum. Where leading standards organizations and companies cooperate to foster interoperability standards for an open metaverse*. https://metaverse-standards.org/

[6] Open Metaverse Browser Wiki. (2026). *Sneeze documentation*. https://omb.wiki/sneeze

[7] World Wide Web Consortium. (2026). *W3C candidate recommendation draft*. https://www.w3.org/TR/webxr/

[8] Open Geospatial Consortium. (2023). *OGC GeoPose 1.0 data exchange standard*. https://www.ogc.org/standard/geopose/

[9] Apple Inc. (2026). *ARGeoAnchor and geo-tracking documentation*. https://developer.apple.com/documentation/arkit

[10] ARCore. (2026). *Build global-scale, immersive, location-based AR experiences with the ARCore Geospatial API*. https://developers.google.com/ar/develop/geospatial