The Metaverse Browser Engine brings spatial computing to the web

Built on open standards for multi-origin compositing, geoposition awareness, self-hosting and hybrid copresence

June 2026

https://metaverse-standards.org/open-metaverse-browser-initative/

The World Needs Spatial Services Delivered via the Web

AI-driven scene understanding is making the world machine-readable

Human users
AR glasses · phones · headsets

Autonomous systems
Vehicles · drones · robots

Both need the same thing
Services from many origins
→ one scene around the client
Web is the only platform with pervasive reach

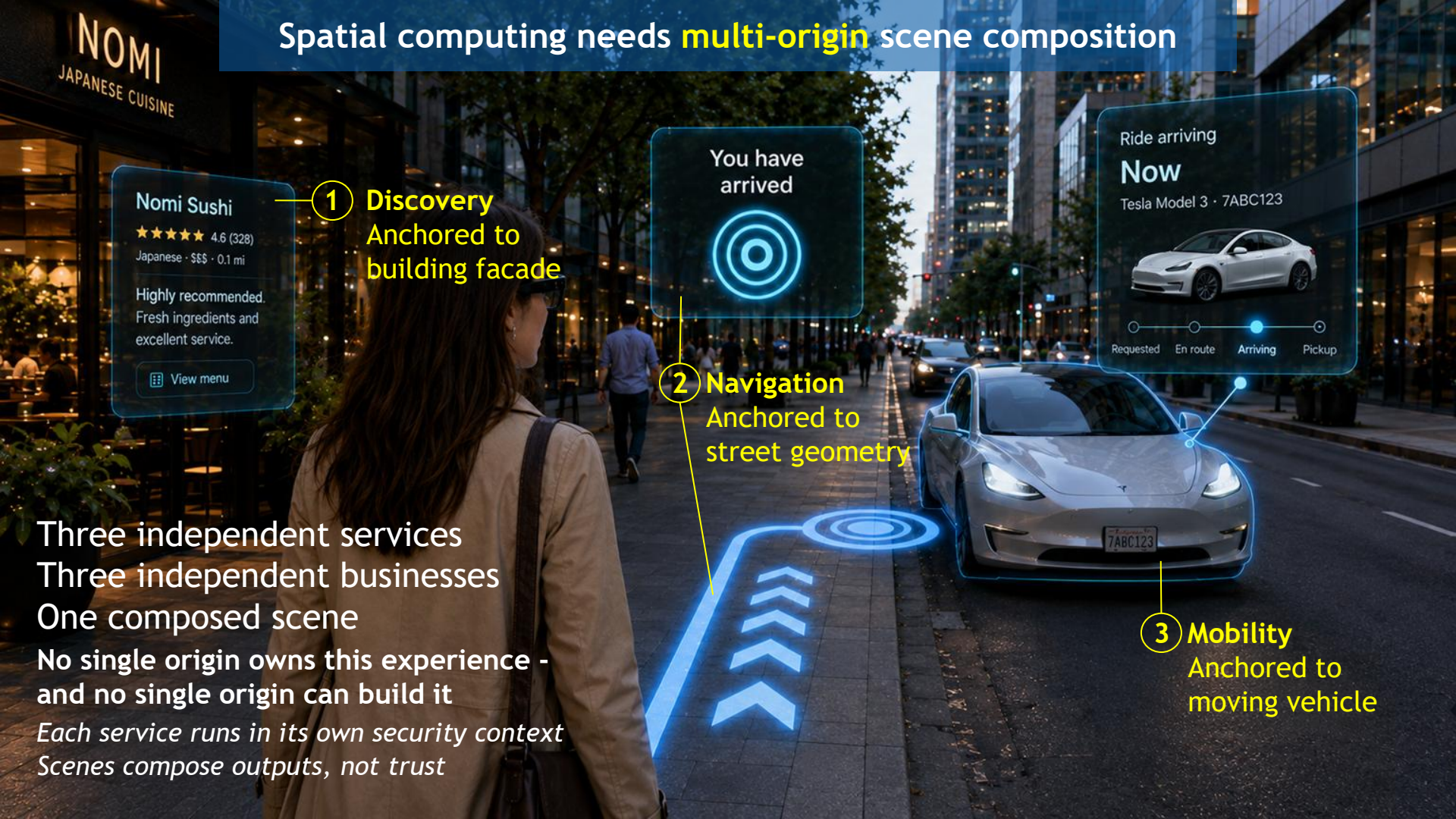
OMBI — an open-source testbed, hosted at MSF *vendor-neutral · accessible to all*

1 — Build consensus on the need and use cases for spatial service composition
2 — Explore deploying spatial services on current web technologies
3 — Identify where current web standards fall short and what new capabilities are needed

OPEN METAVERSE BROWSER INITIATIVE

Smart glasses are the concrete starting point — *though spatial services should run on any device, to the extent of its capabilities*

Spatial computing needs multi-origin scene composition



Three independent services
Three independent businesses
One composed scene

No single origin owns this experience - and no single origin can build it

Each service runs in its own security context
Scenes compose outputs, not trust

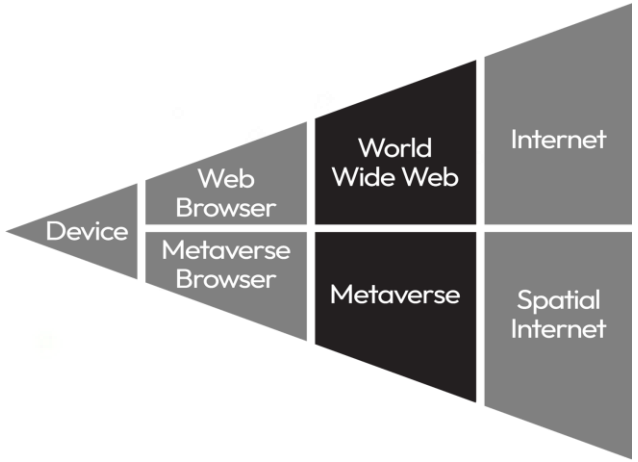
RP1 set out to build a metaverse browser using today's web

RP1 built a global prototype metaverse browser using WebXR
It showed the RP1 team where the opportunities and needs are to build new web spatial capabilities

∞ **An endless map** spanning 1:1 map of Earth

100k+ CONCURRENT — **Solving for scale using 1000x less compute. Concurrent users with full 6DOF and spatial audio** using RP1's proprietary state-sync architecture. Scalable to billions of unsharded concurrent users

90 fps CO-PRESENCE — **Spatial audio + co-presence** across phone, tablet, desktop and VR headset in one shared scene.



The decision to prototype and discuss a METAVERSE BROWSER ENGINE is inspired and informed by the challenges encountered during this hands-on project

OMBI is Solving for these Requirements

1. Permissionless publishing. Spatial services must be publishable the same way as 2D web content — on any server, any origin, with no gatekeeper approval required. Walled gardens are not acceptable. You keep hosting control, flexibility, and full data sovereignty. *Today, you can self-host a WebXR app, but it owns the whole experience; you cannot self-host an independently composable spatial service that the browser can discover, permission, and place into a shared scene*

2. Multi-origin shared 3D scene, with trust: Many origins must be able to contribute to a single scene under browser-enforced origin isolation and explicit spatial capabilities. *The web has cross-origin communication, but no browser-mediated way to compose independent origins into one shared 3D scene. WebXR gives one application control of the XR session — forcing third-party spatial content to be fetched, interpreted, and rendered through the host application*

3. Browser-mediated spatial loading — the "10-foot rule": Spatial services must be seamlessly loaded using the user's location, pose, and permissions to bring nearby services into the scene as they move. Users will not repeatedly install a new app every few steps through the world. *Today, each app implements or integrates its own localization, discovery, and permission flow; the browser does not load spatial content by location*

4. Browser-managed, declarative delivery and display: Authors publish a service to a server; the browser handles transport, permissions, composition, and display — the 2D web paradigm applied to space. The scene is declarative and the engine owns the frame, so authors describe content rather than render it. *WebXR provides an application-controlled XR session, not browser-managed spatial browsing, layout, multi-origin composition semantics, or a shared spatial document model — so every app writes its own render loop and composes its own scene*

Every concept that makes the web work has a direct metaverse equivalent

The Web		The Metaverse
Web Server A server that hosts a website	→	**Metaverse Server** A server that hosts a spatial fabric
Website A page served from one origin	→	**Spatial Fabric** A 3D environment hosted by one operator
Web Browser Chrome, Firefox, Safari	→	**Metaverse Browser** Any browser built with a Metaverse Browser Engine (MBE)
Browser Engine Blink, Gecko, WebKit	→	**Metaverse Browser Engine** Sneeze
HTTP / HTTPS Protocol connecting browser to web server	→	**RMAP** Protocol connecting browser to metaverse server
DOM (Document Object Model) 2D document tree representing a page	→	**SOM (Scene Object Model)** 3D graph representing a collection of spatial fabrics
URL / Web Address Navigate explicitly by typing an address	→	**Proximity + URL** Content loads as you approach

OPEN METAVERSE BROWSER INITIATIVE

OMBI Architecture “Four Ideas”

1. Sneeze Metaverse Browser Engine (MBE)

Geoposition	**CPU Code**	**Rendering**	**OpenXR**	**GPU Shaders**
Location (VPS/GPS)	WASM Execution	Rendering Engine Abstraction	XR Device Abstraction	SPIR-V Execution

2. SOM (Scene Object Model)

Multiple spatial services deliver state into a shared scene

3. RMAP (Remote Model Access Protocol)

Discover and connect to spatial services

Spatial Fabric	**4. Spatial Fabric**	**Spatial Fabric**
3D assets and spatial services	3D assets and spatial services	3D assets and spatial services

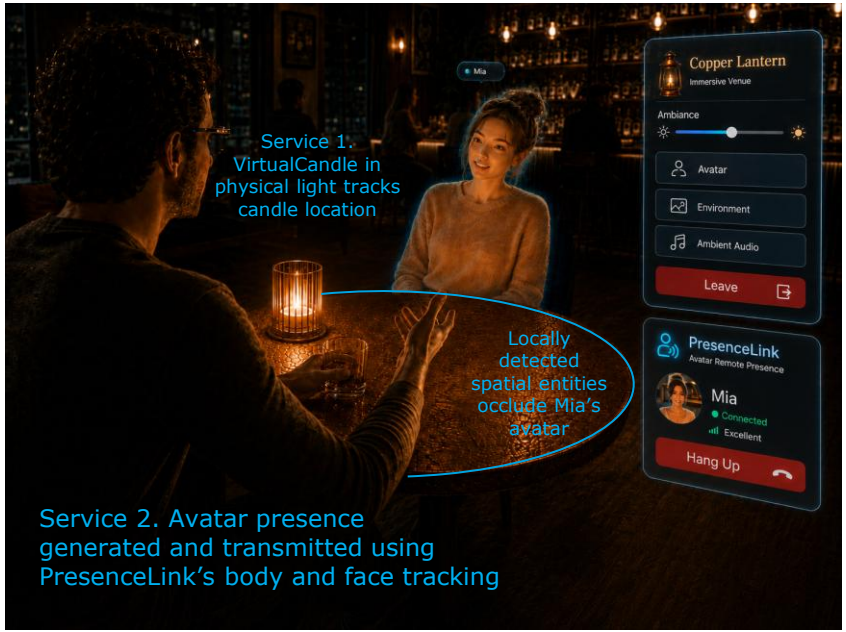
Spatial Fabrics = the 3D evolution of websites

Servers

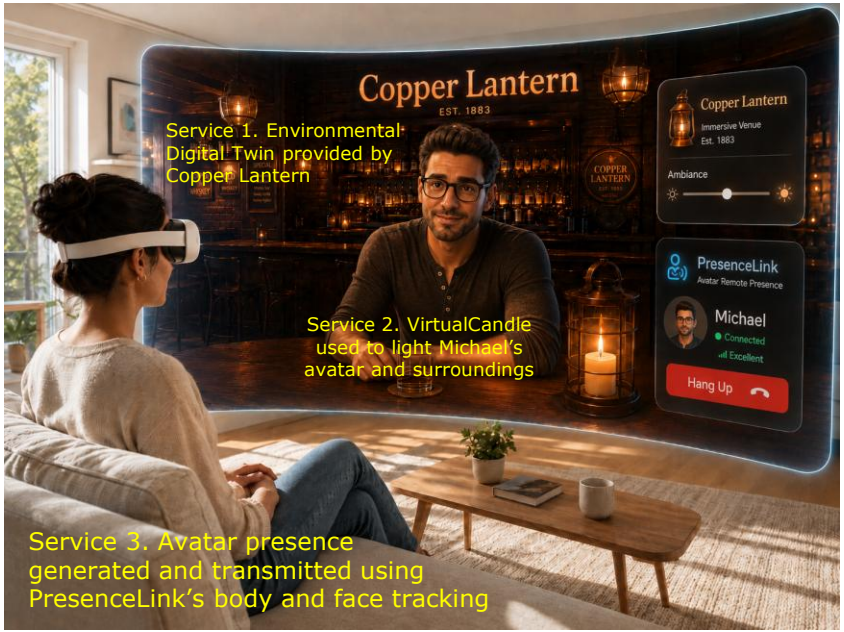
Virtual Presence Use Case Example

Mia and Michael are connecting avatars using ‘PresenceLink’ service
The Copper Lantern Bar provides a environmental digital twin service
VirtualCandle service is an example of cross-origin rendering interaction

Michael is in the **real-life** bar using **AR**



Mia is at home using **immersive VR**

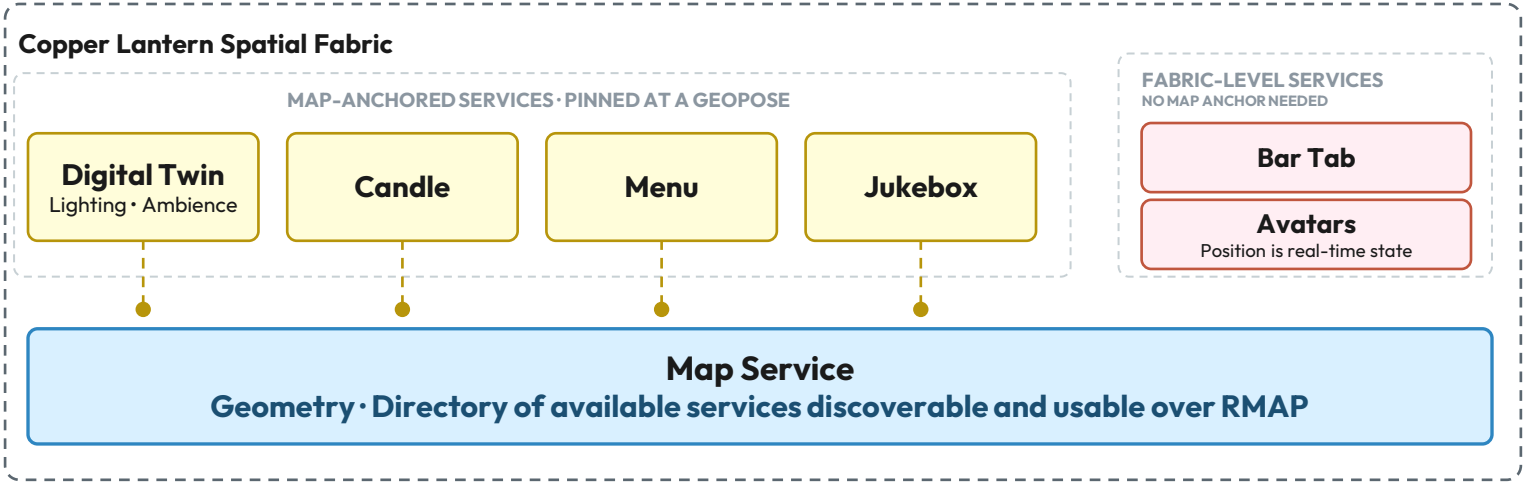


Spatial Fabric — the 3D evolution of a website

A shared 3D coordinate space — public or private — that can be pinned to a real-world place

Map service + list of optional fabric-level services + map-anchored services

Decentralized self-hosting — fabric owners keep control of their data and monetization



Real-time interaction — persistent RMAP connections and live state updates support shared, synchronized experiences

Open service integration — many fabrics and services can connect into one session — composing into one scene, not tabs

Massive scale — from a single room to RP1's 1:1 digital twin of Earth, many concurrent users

Discovering Spatially Located Services

USE CASE
Sneeze + Spatial Fabric service discovery

User opens Sneeze in spatial context by specifying a **primary fabric**

GPS/VPS provide continuous position information to Sneeze

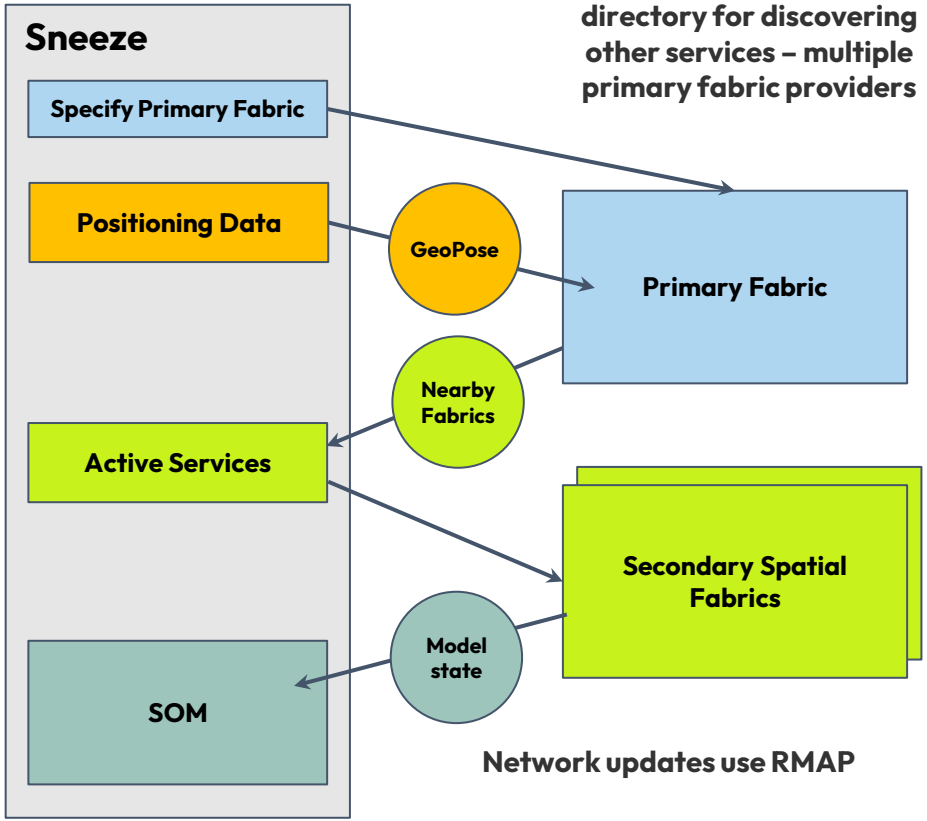
Sneeze fetches proximal map nodes (via RMAP) from loaded fabrics' map services

Incoming map nodes identify secondary spatial fabrics/services, which are loaded into the SOM

Map nodes that become too far away are unloaded from the SOM

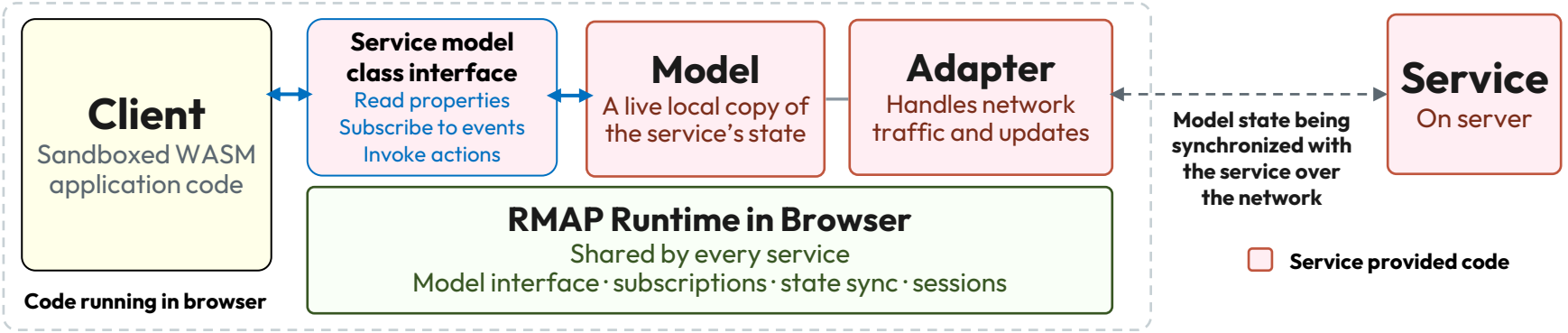
SOM composes the multi-origin scene

OUTCOME Place-aware services rendered, anchored to the user's physical surroundings



Remote Model Access Protocol (RMAP)

Services are modeled as live local objects that the client programs against



The service ships the model class and adapter — so the client needs to write NO integration code

The service controls both ends of the network link — and so can use any language and protocol it wishes

Real-time state — the service pushes changes; the model reflects them live, no polling needed

The on-client Model and Adapter are thin, per-service layers over the browser's shared RMAP runtime

The service model class interface has the same shape for every model — usable with no prior knowledge

CORS sidestepped — no cross-origin fetch surface is needed — security by architecture

What is Sneeze?

Sneeze is a Metaverse Browser **Engine**

Sneeze processes 3D spatial content into the SOM (Scene Object Model) - a 3D scene graph
(instead of a current browser engine processing HTML/CSS/JavaScript into the 2D DOM)

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

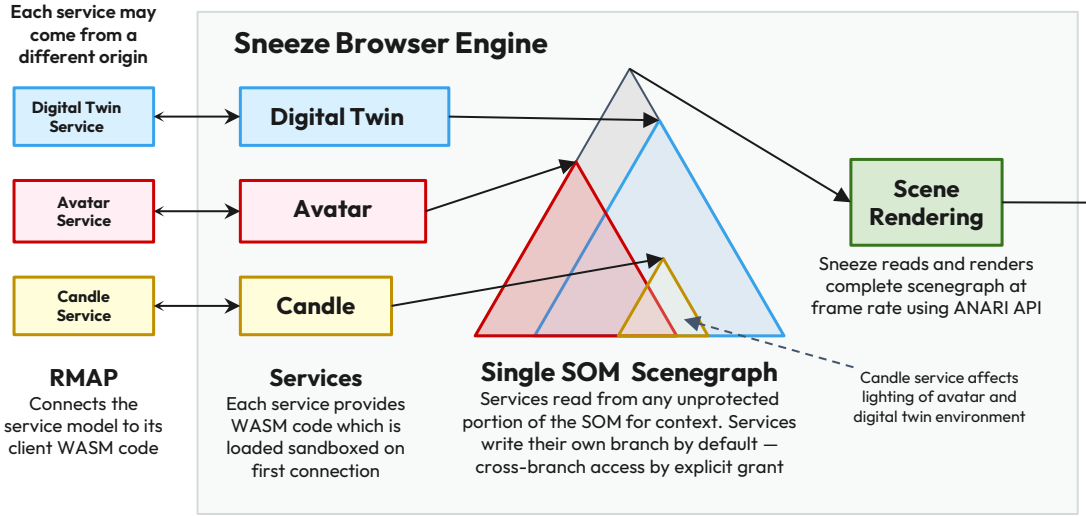
Language: C++

Open-sourced: March 2026, contributed to the Open Metaverse Browser Initiative (OMBI) under the **Metaverse Standards Forum**, where it serves as an exemplar MBE implementation

Rendering: does not implement its own renderer - delegates through a graphics abstraction layer

Sneeze Engine

How multiple spatial services deliver state into a shared scene



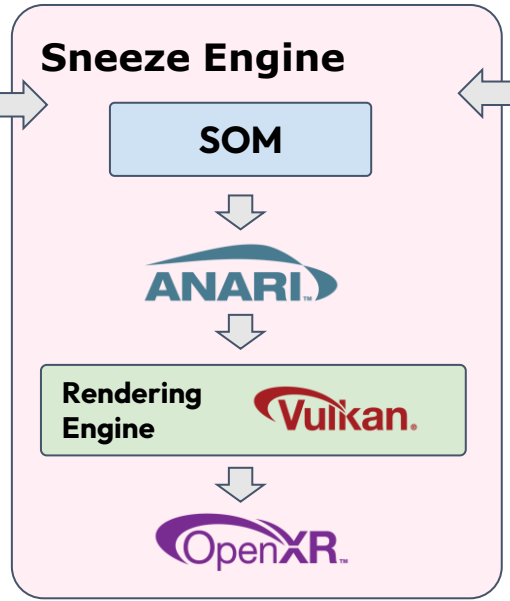
RMAP standardizes service model access
WASM isolates service logic running in the browser
SOM branch ownership composes independent providers into one scene

Sneeze Engine and Khronos Standards

Khronos welcomes use of its standards in OMBI

Drives new requirements into the roadmaps of multiple open standards

Valuable interoperability testbed and proving ground



3D asset import format
Inc. 3D Gaussian splats, Avatars, and glTF 2.1 progressive streaming

Vulkan GPU API
Native rendering with cross-platform portability

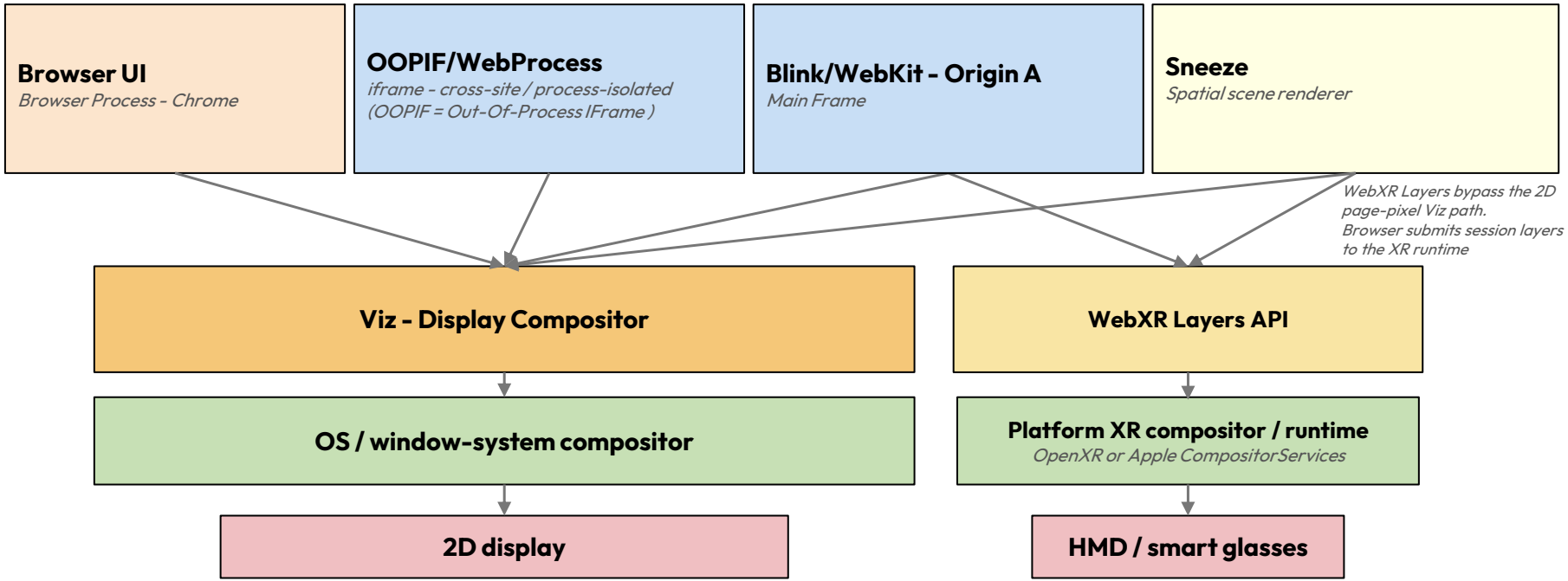
Shaders/kernels
GPU-offload for compute intensive tasks such as inferencing

ANARI Rendering Engine API
Enables implementation agility for choice of rendering engines

OpenXR XR runtime and display abstraction
Portable access to almost all XR devices

Adding Sneeze to the Web Stack (interim)

Cross-site content may render in separate processes - Viz is the shared browser composition stage for page pixels



The Web is not a shared-scene composition model *- web security, process isolation where used, and browser-process checks prevent unauthorized DOM, resource, GPU resource, and script-readable pixel access across origins*

Viz aggregates compositor frames *- surfaces/draw-quads, not as a shared DOM, WebGL/WebGPU, or 3D scene graph*

Immersive XR is single-session *- only one visible immersive XR session owns the XR device display; other origins do not co-compose into that session*

Four Ways to Compose a spatial engine (Sneeze) with a web engine (Blink)

A Full-Viewport Handoff

LIKE *Like opening a PDF, entering a WebXR immersive session, or invoking the proposed W3C <model> element*

MECHANISM Browser switches between Blink-rendered web pages and Sneeze-rendered spatial fabrics

USE CASE Fastest path to open a spatial fabric from the web - similar to opening a PDF or immersive document

LIMITATIONS No simultaneous composition; user is either in a web page or a Sneeze scene. Requires navigation, permissions, identity, and lifecycle handoff

B Sneeze Island Inside a Web Page

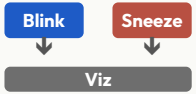
LIKE *Like <iframe>, <canvas>, or a Sketchfab / Google Maps embed - a sandboxed rectangle for 3D*

MECHANISM Blink embeds an opaque Sneeze viewport as a rectangular region inside a normal web page

USE CASE Lets web developers embed live 3D spatial services, fabrics, previews, or mini-worlds in existing websites

LIMITATIONS The 3D scene is boxed inside the page; Blink does not directly access or manipulate the SOM

C Shared Compositor / Sibling Surfaces



LIKE *Like Chrome's Viz combining Blink with browser UI - or an OS window compositor*

MECHANISM Blink and Sneeze each render independent surfaces that are combined by a shared compositor (Viz in Chromium; OS-level on visionOS/Android)

USE CASE Useful for browser chrome, dialogs, permission prompts, side panels, overlays, and mixed UI around a spatial scene

LIMITATIONS Simple rectangular composition only. Does not by itself create true 3D integration unless depth, pose, hit testing, and occlusion are explicitly shared

D Web Surface Inside the Sneeze 3D Scene

LIKE *Like Apple visionOS rendering Safari panels in 3D space, or WebView inside a native AR app*

MECHANISM Blink renders live web content to a surface/texture that Sneeze places as an object in the SOM

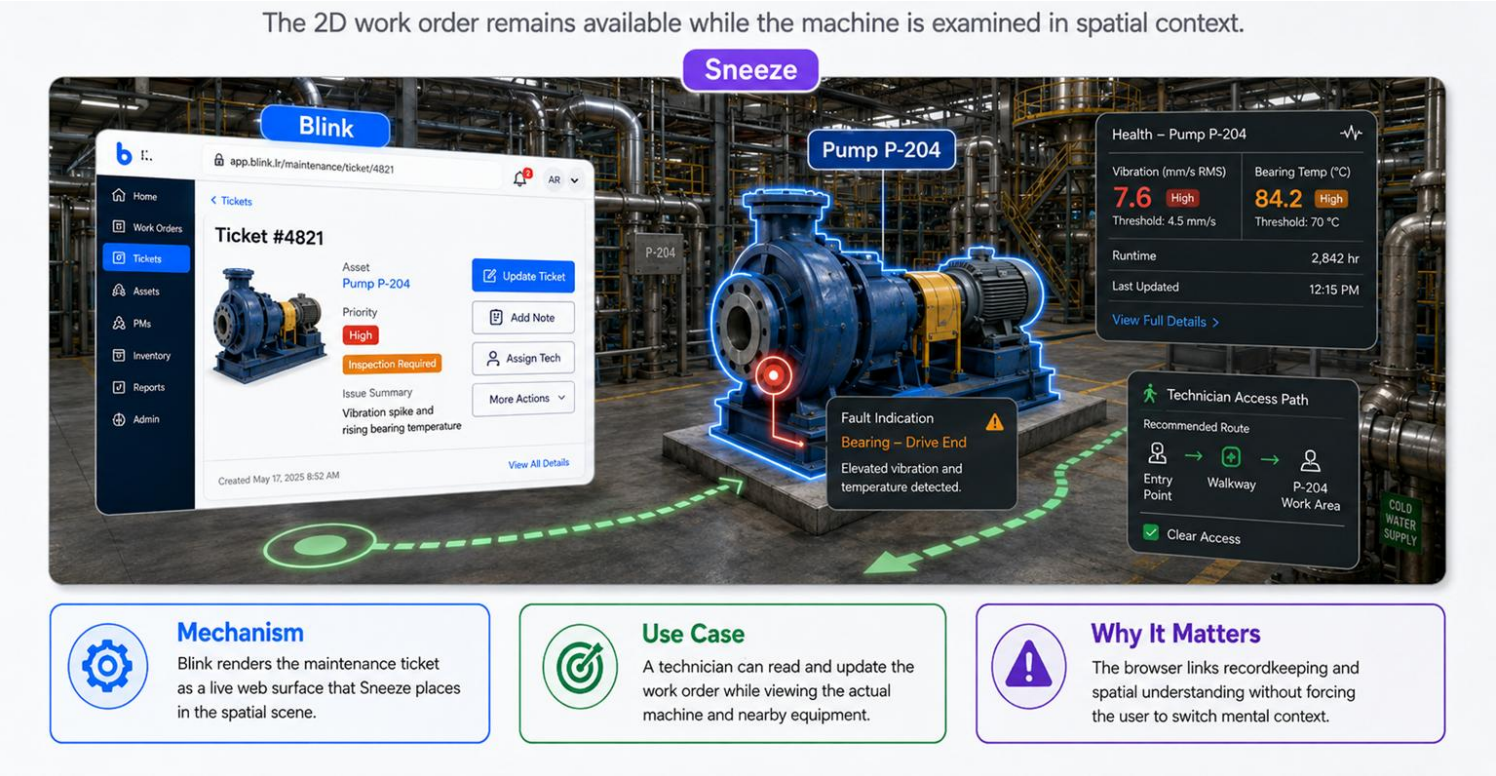
USE CASE Web panels, dashboards, forms, commerce flows, maps, documents, and controls embedded into a true 3D spatial scene with perspective, transforms, occlusion, and lighting

LIMITATIONS Requires careful input routing, focus management, permissions, depth/occlusion handling, and security isolation. Initially likely treats the web page as a rectangle, not individual DOM nodes as SOM objects

Needs more than Viz Composition

Flexible 3D Composition of 2D Web Content

The SOM can bring a 2D Web rendered surface fully into the 3D composited scene



Metaverse Browser Initiative at MSF

Open Source, Open Governance

Open-source project to create ‘Sneeze’ Browser Engine

OSS best practices and permissive Apache 2.0 license

Testbed / exemplar implementation to engage industry and multiple standards organizations

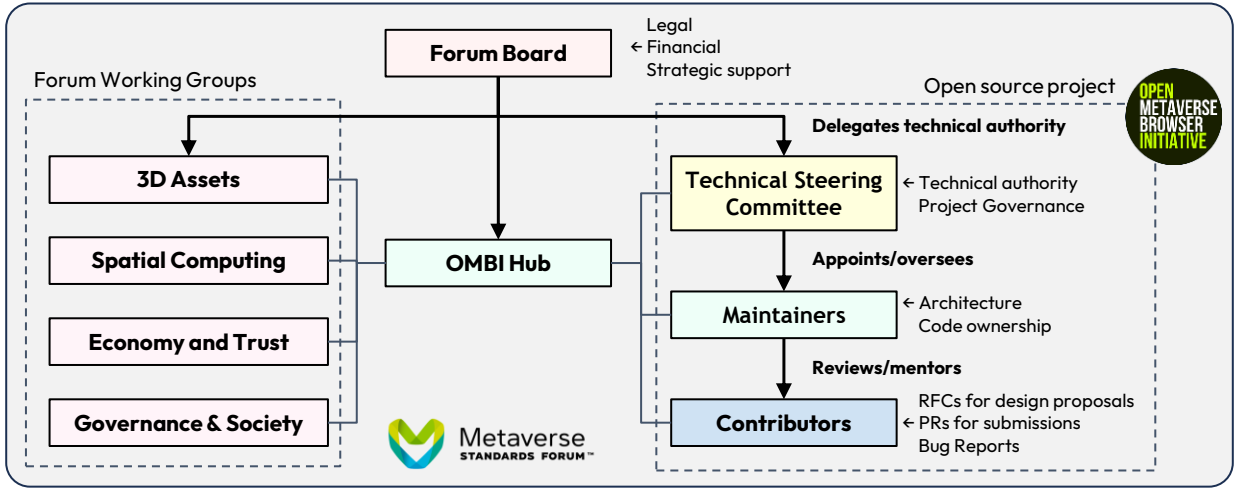
Domain Experts in Forum Working Groups
Provide open source project input guidance and may use OMBI as a testbed. For example:

3D Assets
Avatar formats, including scalability for 1000s of avatars in a scene

Spatial Computing
Portable access to visual positioning systems, indoor positioning, geolocation

Economy and Trust
Identity and transactions, Universal Manifest for persona and possession portability

Governance and Society
EU Digital Identity Wallet (EUDI) alignment under eIDAS 2.0, privacy and child protection compliance



Forum members may participate in both Forum working groups and open-source project

University of Rochester
Center for Extended Reality

ANNOUNCING

Open Metaverse Academic Alliance

Bringing Together Academic Institutions Worldwide Around OMBI Interoperability for the Open Metaverse

Find Out More! Get Involved!

OPEN METAVERSE BROWSER INITIATIVE

Metaverse STANDARDS FORUM™

https://omb.wiki/