WebMCP and Accessibility

Putting Assistive Technology on the Highway

Author

Nadav Bernstein

With strategic contribution from

Tamar Schapira

Sean Beld

Version 1.0

February 2026

Organization

Senseit

Executive Summary

WebMCP introduces a capability-level interaction model for web applications.
Rather than navigating rendered interface elements, agents invoke declared actions.
This marks a shift from DOM-level interpretation to action-level integration.

Today, assistive technologies operate primarily at the DOM layer, interpreting structure, semantics, and state after rendering. While effective, this model is inherently reactive.

If WebMCP formalizes action-level schemas for agents without parallel consideration for assistive integration, structural asymmetry emerges:

- Agents operate on declared capabilities.
- Assistive technologies remain constrained to rendered artifacts.

This is not a compliance gap. It is an abstraction-layer gap.

When new integration surfaces are standardized, accessibility must be addressed at the same architectural layer. Capability schemas should not evolve independently of assistive access models.

We propose exploring High-Level Assistive Actions, enabling assistive technologies, where standardized and platform-aligned, to discover and invoke declared application capabilities directly. This approach would be additive to existing accessibility architecture, not a replacement. The accessibility tree, ARIA, platform APIs, and WCAG remain foundational. Task-level abstraction is acceleration, not substitution.

Implications include:

- Structured exposure of workflow states
- Deterministic modeling of error and validation flows
- Clear definition of critical user tasks
- Parity between AI agents and assistive technology users

WebMCP represents an architectural inflection point. Integrating accessibility at the capability layer now may prevent assistive technologies from remaining downstream of a new abstraction surface.

This paper outlines a guardrail-driven approach to action-level accessibility integration and invites technical critique from browser engineers, standards contributors, and assistive technology architects.

The web is gaining a new primitive.

WebMCP allows websites to expose structured, callable tools, not just buttons and forms, but defined capabilities. Instead of navigating the interface, software can invoke named tasks with structured inputs.

This is more than a convenience for AI agents.

If implemented responsibly, it represents a rare opportunity:

To move assistive technology users from navigating roads to driving on highways.

Accessibility has long focused on making interfaces operable. WebMCP gives us the chance to make outcomes directly reachable.

That difference matters.

What WebMCP Changes, and What It Doesn't

WebMCP does not modify the accessibility tree.
It does not replace ARIA.
It does not redefine WCAG.
It does not change how browsers compute roles, names, and states.

The DOM still becomes an accessibility tree.
The accessibility tree still maps to platform APIs.
Assistive technologies still consume those APIs.

In its current early preview form, WebMCP operates entirely within the browser and does not expose new accessibility behavior by default.

What WebMCP introduces is a parallel surface:

A task layer.

Where UI describes what things are,
WebMCP describes what the application can do.

This is not a semantic layer.
It is an action layer.

And that distinction gives us room to innovate without destabilizing the foundations.

From Interface Accessibility to Task Accessibility

Today, accessibility ensures users can:

- Navigate controls
- Activate buttons
- Operate widgets
- Complete workflows step-by-step

That is essential.

But many workflows remain cognitively heavy even when perfectly compliant.

Multi-step forms. Nested navigation. Expanding data tables. Dynamic filters. Modal sequences.

The user must traverse the interface to reach the outcome.

WebMCP allows something different.

Instead of:

- Navigate to claims
- Filter by date
- Expand result
- Download document

The user can express:

“Download my March report.”

The task executes.
The outcome is reached directly.

This reduces:

- Motor demand
- Cognitive branching
- Focus traversal
- Sequential dependency

This is task-level accessibility.

This does not replace well-structured HTML; it shifts the primary entry point from controls to declared tasks.

It does not replace accessible UI.
It elevates the way outcomes can be reached.

Stability: Accessibility That Survives Redesign

The web is not static.

Design systems change.
Layouts are refactored.
Components are replaced.
Frontends are rewritten.

Even well-intentioned redesigns introduce accessibility regressions.

WebMCP introduces something new:

A declared capability.

When a site defines:

submit_application(data)
download_report(range)
checkout_cart(options)

It is not just defining a button.
It is declaring, in a structured way:

“This application can perform this task.”

That declaration is separate from layout, styling, and component structure.

Those declared capabilities can remain stable even as the interface changes.

For users who rely on screen readers and related assistive technologies, that means:

Access to outcomes becomes less fragile across redesign cycles.

That is structural resilience.

The Real Opportunity: High-Level Assistive Actions

Today, platform accessibility APIs expose control-level actions:

Activate. Expand. Set value. Select.

Assistive technologies operate at the widget level.

WebMCP introduces domain-level actions.

The central question is whether declared WebMCP tools could be responsibly mapped to platform accessibility APIs.

The opportunity is not to change the accessibility tree; it is for browsers to explore whether declared WebMCP tools could be exposed to assistive technologies as structured, high-level actions, in a way that builds on existing platform models rather than replacing them.

Assistive technologies could present:

For example, a VoiceOver user on a booking site might hear: “Available actions: Search flights. Change reservation. Download itinerary.”

Not as simulated clicks.
Not as scripted navigation.
But as declared capabilities.

This would not remove interface accessibility.
It would complement it.

The architectural pieces exist in today’s browser and platform models.
What is required is careful, cross-vendor design.

If We Do This, We Must Do It Right

Exploring high-level assistive actions is not a proposal for immediate implementation.

It is a statement that if such a direction is pursued, certain guardrails are essential.

If WebMCP is to elevate assistive technology access, three foundations are non-negotiable.

1. Standardization

Task discovery must be cross-browser.
Exposure rules must be consistent.
No vendor-specific shortcuts.

Assistive technology vendors must be involved early, not retrofitted later.

Fragmentation would harm the very users this is meant to empower.

2. Consent and Control

Domain-level tasks can be powerful:

Submitting applications.
Triggering payments.
Modifying records.

Assistive users must not lose control.

Invocation must be explicit.
Destructive actions must require confirmation.
Execution must remain transparent and predictable.

Acceleration must not come at the cost of autonomy.

3. Deterministic Execution and UX Discipline

High-level actions exposed to assistive technologies must be deterministic.

The same input must produce the same outcome.
Execution must be stable across invocations.
Focus must move predictably.
Changes must be announced clearly.
Event ordering must remain disciplined.

Accessibility depends on reliability.

WebMCP must complement the accessibility pipeline, not destabilize it.

What WebMCP Is Not

It is not an automatic compliance engine.
It is not a semantic repair tool.
It is not a chatbot interface replacing structured interaction.
It is not a replacement for accessible UI.

Accessibility remains grounded in:

- Deterministic semantics
- Testable outcomes
- Stable mappings
- WCAG conformance

WebMCP should augment accessible UI, not substitute for it.

The correct model is dual-path:

Accessible UI as the compliant baseline.
Task-level abstraction as optional acceleration.

If the task layer fails, the UI remains fully operable.

That boundary preserves both innovation and integrity.

A Moment We Should Not Miss

WebMCP is currently experimental and browser-specific.
Its direction is not yet fixed.

That makes this the right time for accessibility voices to engage.

We can treat WebMCP as an AI convenience feature.

Or we can recognize it as a chance to rethink how assistive technology accesses outcomes, without changing the foundations of accessibility architecture.

For decades, accessibility has focused on ensuring users can navigate interfaces.

WebMCP gives us the opportunity to ensure users can reach outcomes directly.

To move assistive technology users from navigating complex roads to driving on structured highways.

Not by bypassing standards.
But by building on them.

If WebMCP becomes part of the web platform, accessibility must be part of its design from the beginning.

The future of accessibility is not AI replacing interfaces.

It is AI elevating how outcomes are reached.

And that elevation must remain deterministic, standardized, and user-controlled.

That is not incremental improvement.

That is structural advancement, designed responsibly.

Appendix: High-Level Assistive Actions

Elevating Assistive Technology to the Task Layer (v1.1)

1. The Structural Limitation We've Lived With

Modern accessibility architecture is built on a powerful and well-established model:

DOM → Accessibility Tree → Platform Accessibility API → Assistive Technology

Specifications such as Core-AAM and HTML-AAM define how roles, states, and properties map into operating system APIs. Platform surfaces such as UIA (Windows), AXAPI (macOS), and AT-SPI (Linux) expose:

- Roles
- States
- Properties
- Control-level actions (Invoke, Expand, SetValue, Select, etc.)

This architecture has made the web operable.

But it is fundamentally control-oriented.

Assistive technologies can activate controls.
They cannot invoke domain-level outcomes.

The model answers:

“What is this element, and how do I operate it?”

It does not answer:

“What can this application accomplish?”

As a result, users must translate:

Interface → Intent → Outcome

Often across many sequential steps, even when fully WCAG-conformant.

2. The Opportunity Introduced by WebMCP

WebMCP allows a site to declare structured tools such as:

- submit_application(data)
- checkout_cart(options)
- download_statement(range)
- filter_claims(criteria)

These are not widgets.
They are declared capabilities.

Today, these tools are designed for agent invocation within the browser.

Architecturally, however, they introduce a new concept:

A domain-level action layer.

The question is not whether WebMCP replaces ARIA. It does not.

The question is whether declared capabilities could, in the future, be responsibly exposed to assistive technologies as structured, high-level actions, in a way that builds on existing accessibility models rather than altering them.

3. A Careful, Platform-Aligned Vision

This does not require changing how the accessibility tree is computed.

It does not require replacing existing platform action models.

The underlying architectural primitives already exist:

Browsers already mediate between the DOM and platform accessibility APIs.
Platform APIs already support action exposure at the control level.

The opportunity is to explore whether domain-level actions could be layered, carefully and additively, above that foundation.

A possible sequence could look like:

Step 1 - Tool Declaration

A site registers WebMCP tools with:

- Name
- Description
- Structured schema
- Deterministic execution logic

Step 2 - Browser Mediation

The browser validates:

- Security constraints
- Execution boundaries
- User consent requirements

Step 3 - Accessibility Exposure (If Standardized)

Eligible tools could be exposed through platform accessibility APIs as structured actions, not replacing existing control actions, but complementing them.

Assistive technologies could then query:

Available actions:
- Submit Application

- Download Report
- Save Draft

Not as simulated clicks.
Not as scripted navigation.
But as declared capabilities.

This would require deliberate, cross-browser coordination and collaboration with assistive technology vendors.

It is not automatic.
It must be intentionally designed.

4. What This Would Change for Assistive Users

This is not about bypassing interfaces.

It is about reducing translation burden.

Today:

Navigate → Interpret → Activate → Repeat

With high-level assistive actions:

Discover → Select Task → Execute → Review Result

This reduces:

- Motor demand
- Sequential navigation fatigue
- Cognitive branching
- Workflow error accumulation

In enterprise, healthcare, government, and financial systems, where workflows are complex and high-stakes, this shift could be meaningful.

Accessible UI ensures users can complete workflows.

High-level assistive actions could help users complete them more directly.

That is not replacement.
It is acceleration.

5. The Guardrails Required

If this future is to be realized responsibly, five conditions are non-negotiable.

1. Additive, Not Replacing

High-level assistive actions must augment accessible UI.

They must never replace deterministic, testable interface paths.

WCAG conformance remains grounded in the DOM and accessibility tree.

Task invocation is acceleration, not substitution.

2. Deterministic Execution

Domain-level actions exposed to assistive technologies must:

- Produce consistent, testable results
- Remain stable across invocations
- Avoid probabilistic or changing semantics

AI may help discover or select an action.
But the action itself must execute deterministically.

Compliance integrity depends on predictability.

If an action cannot be deterministic, it should not be exposed as a first-class accessibility action.

3. Explicit Consent and Confirmation

Domain actions may trigger:

- Legal submissions
- Financial transactions
- Record modifications

The platform must require:

- Clear confirmation models
- Transparent execution descriptions
- Accessible error reporting

Acceleration must not compromise agency.

4. Focus and Event Discipline

When a high-level action executes:

- Focus must move predictably
- Changes must be announced clearly
- Screen reader context must remain stable

Accessibility pipelines are sensitive to event ordering and focus displacement.

Execution discipline must be part of any exposure model.

5. Cross-Vendor Standardization

This cannot be proprietary.

If WebMCP tools are exposed differently across browsers, assistive technologies will fragment.

High-level assistive actions must be:

- Standardized
- Interoperable
- Designed with accessibility participation from the beginning

Browser experimentation must not precede accessibility coordination.

6. The Ecosystem Ask

If WebMCP evolves beyond agent tooling, we call for:

1. Early discussion within accessibility and browser communities about task-level action exposure.
2. Collaboration between WebMCP editors and accessibility contributors before any accessibility-facing rollout.
3. Structured consultation with assistive technology vendors.
4. Clear specification language stating that task abstraction is additive to accessible UI, not a replacement.

Accessibility cannot be retrofitted into a task layer after deployment.

It must be co-designed.

7. The Strategic Moment

For decades, accessibility has ensured:

“You can operate the controls.”

WebMCP creates the possibility, if carefully designed, of also ensuring:

“You can complete the task.”

High-level actions would not remove confirmations, corrections, or required decisions. Those interactions would still occur through accessible browser and assistive technology interfaces, not through an opaque agent loop.

If implemented responsibly, high-level assistive actions would not create a separate AI lane and accessibility lane.

They would place assistive technology users on the same structured task surface as agents.

Not compensating.
Not catching up.
But accessing outcomes directly.

That is not incremental improvement.

It is structural advancement, designed with care.