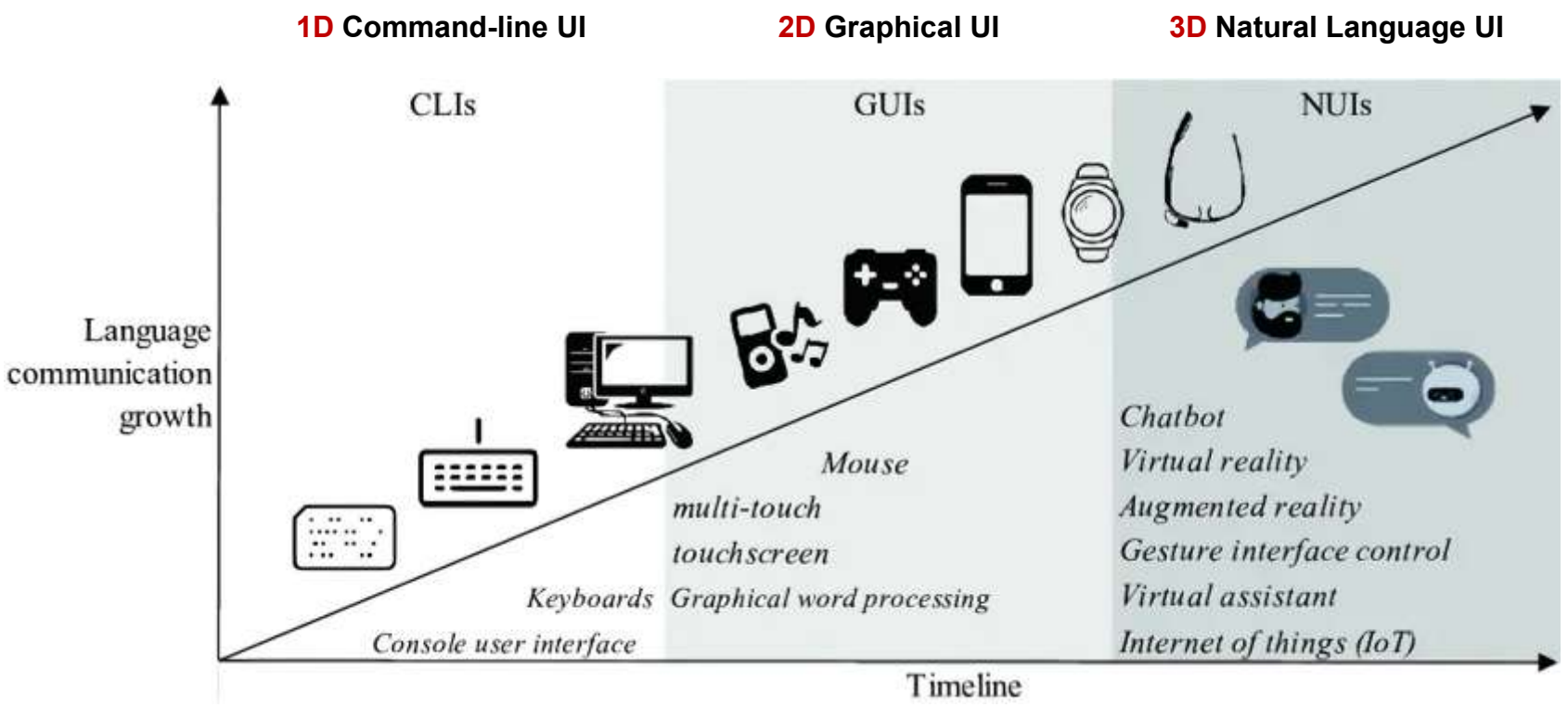
Generative UI & MCP Server
The Future of Web AI

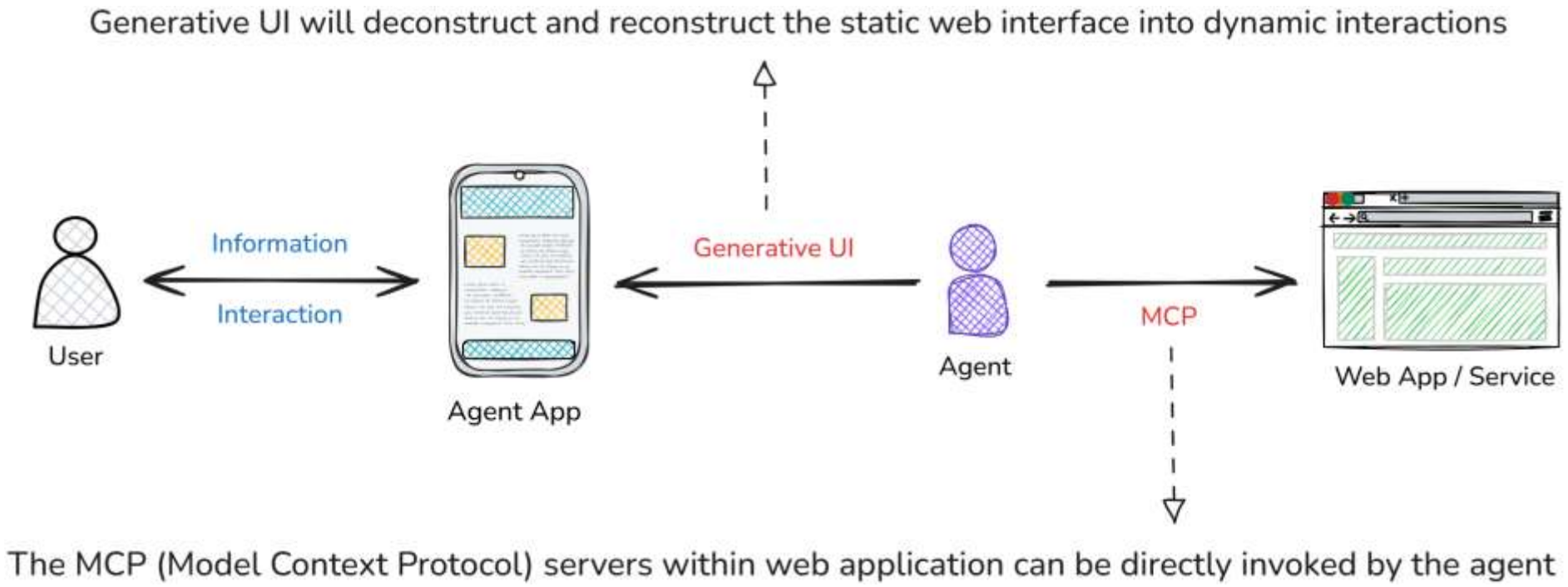
Chunhui Mo (mochunhui@huawei.com)

Exploration IG, September 26, 2025

Natural Language Become the Main User Interface for Web AI Apps



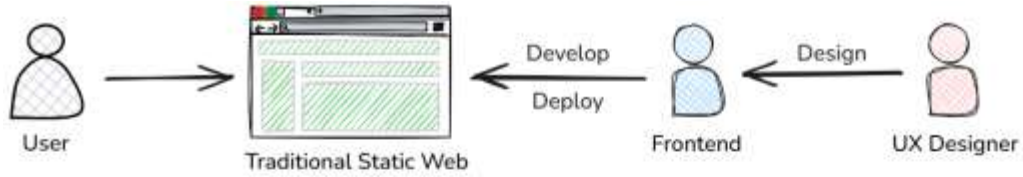
Two Critical Technologies for Agent App Development



Generative UI for Web

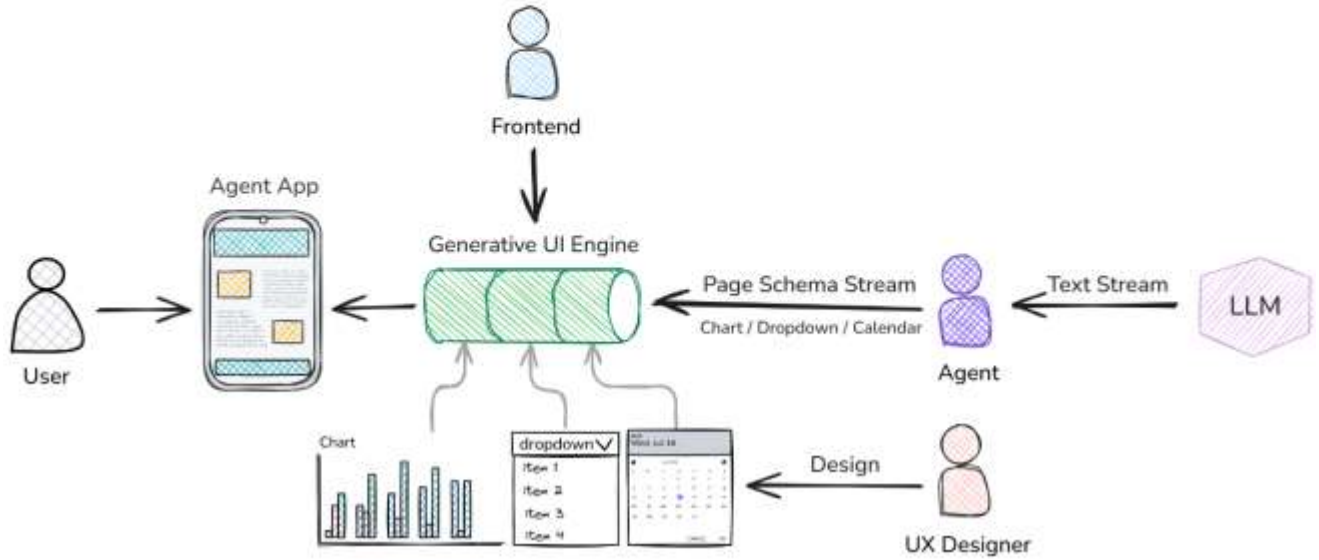
Differences from Traditional Static Web App

Traditional Static Web App



After a web page is designed, developed, and deployed, the interactive interface seen by the user does not change.

Dynamic Generative Agent App

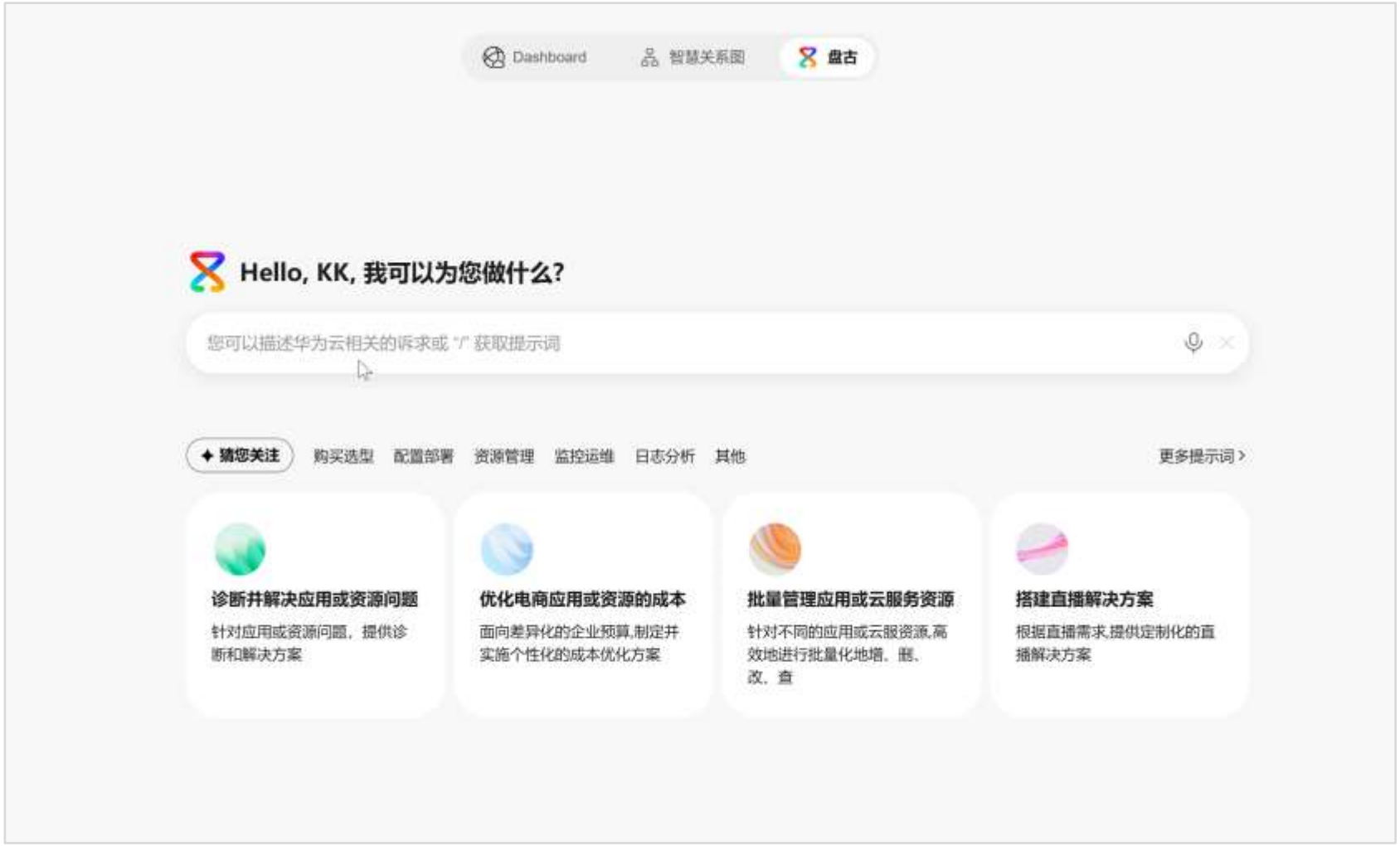


The text flow is converted into a page description schema flow, and the interactive interface seen by the user is generated dynamically in real-time.

Dynamic Generative Agent App Example

Traditional Static Web App

Dynamic Generative Agent App



Dynamic generative Agent apps completely deconstruct & reconstruct the UI of traditional static web apps

Why Generative UI is Important for Web AI

1. LLMs aren’t yet fast enough at generating interactive interfaces in real time. Especially for on-device models, where users often have to wait a long time before they see the interface, and that’s not a good user experience.

2. Only the dynamic generative UI is achieved in the natural language dialogue box. But in the future, the entire application interface will be dynamically generated.

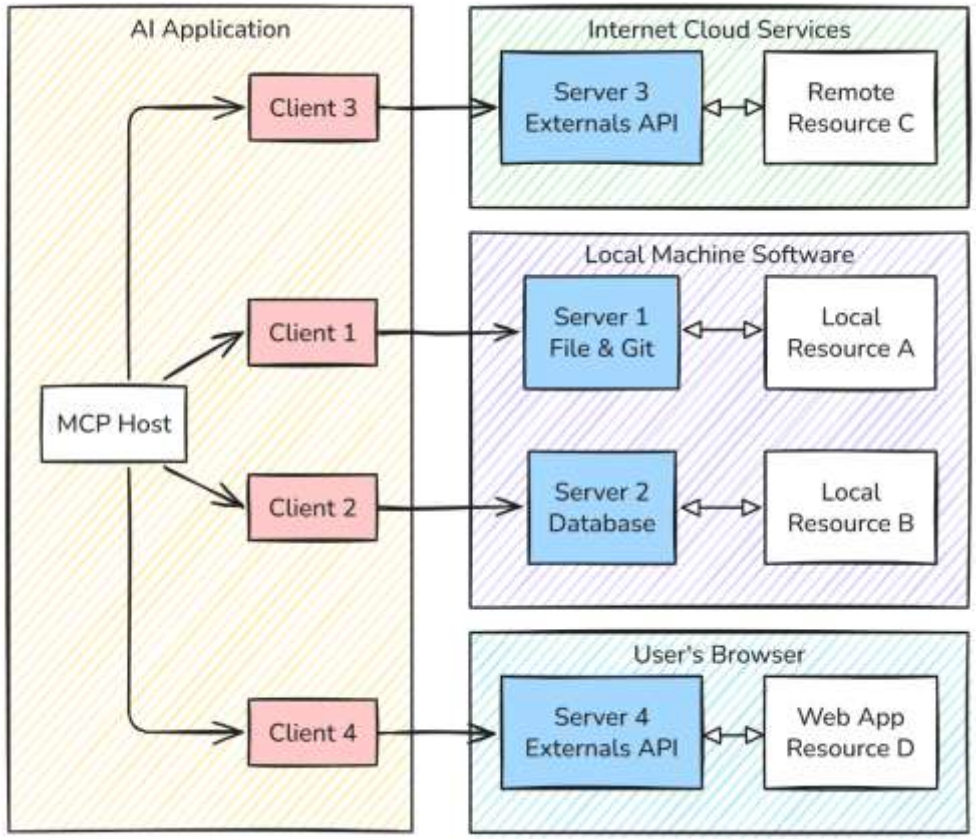
3. Browser engines were never designed or optimized for generative UI. For example, when components are rendered one by one, the web page is forced to repaint again and again. And browsers don’t natively support rendering the individual parts of a complex component piece by piece.

MCP Servers for Web

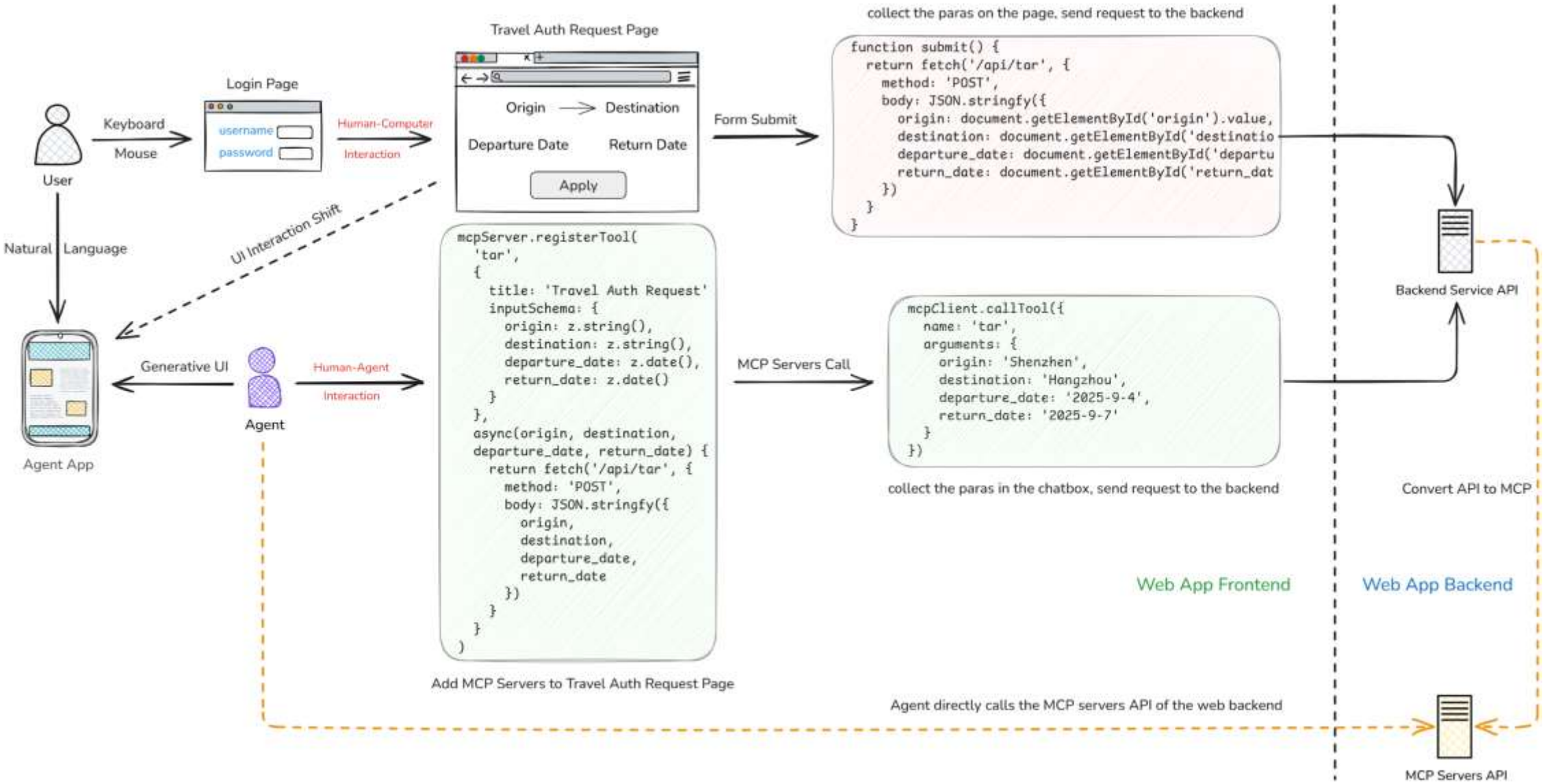
MCP Servers From Native Apps to Web Apps

Blender MCP allows an LLM to directly control
the Blender modeling software via the MCP protocol

Web app becomes an MCP Server option



MCP Server Running Inside A Web Page



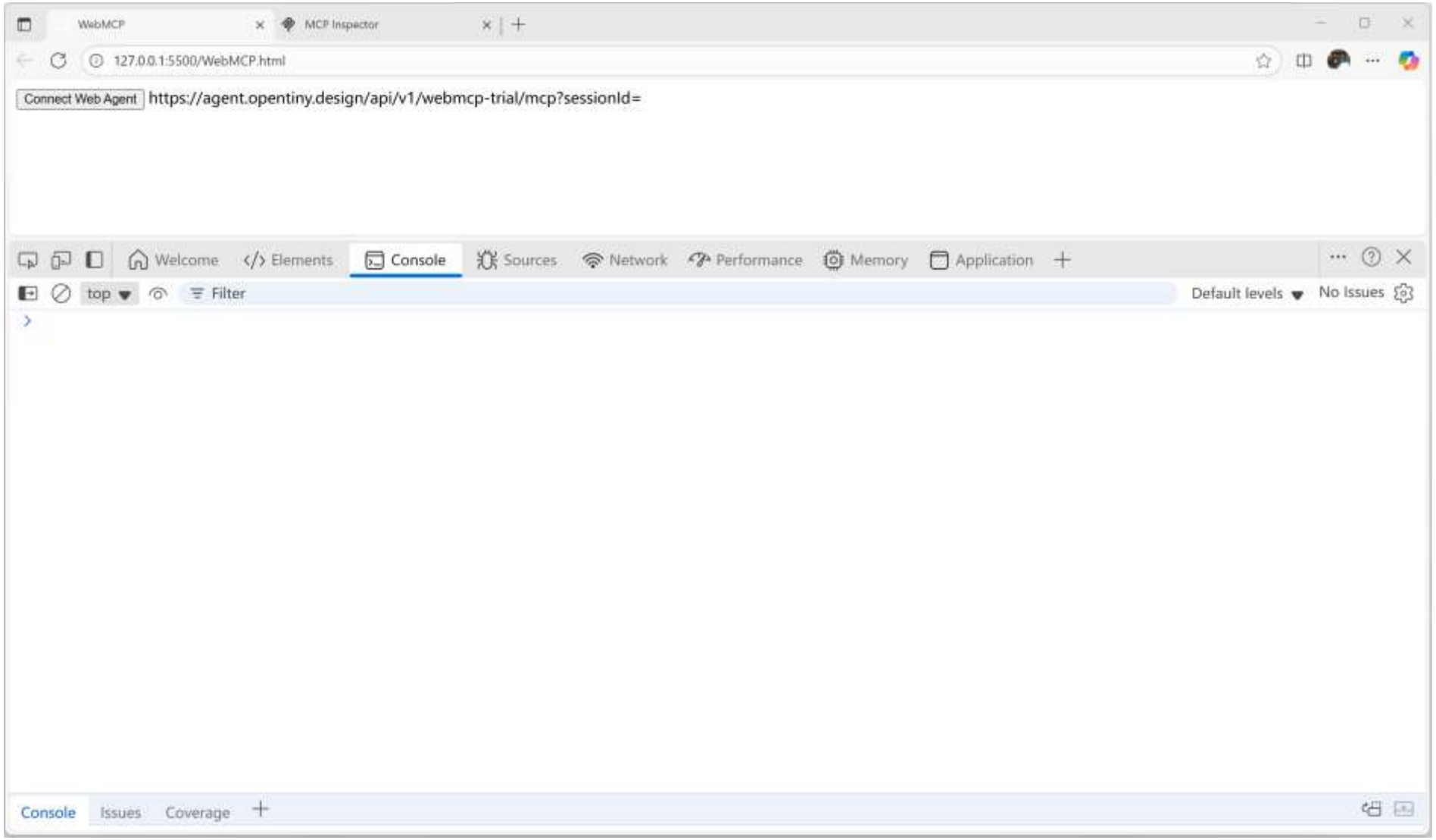
Our WebMCP Proposal: SDK Example

Fully compatible with MCP SDK Create an MCP Server and MCP Client using standard MCP SDK methods. It not only supports registering MCP Tools, but also supports registering MCP Prompts, MCP Resources, and more. It can connect to a remote Web Agent service, allowing both the page Client and the remote MCP Host to call the page MCP Server's Tools, Prompts, and Resources.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>WebMCP</title>
    <script src="webmcp.js"></script>
    <script>
      const { createMessageChannelPairTransport, WebMcpServer, WebMcpClient, ResourceTemplate, z } = WebMCP;

      async function connect() {
        // Create pair MCP transports
        const [serverTransport, clientTransport] = createMessageChannelPairTransport();
        // Create an MCP server
        const server = new WebMcpServer({ name: 'demo-server', version: '1.0.0' });

        // Add an addition tool
        server.registerTool(
          'add',
          {
            title: 'Addition Tool',
            description: 'Add two numbers',
            inputSchema: { a: z.number(), b: z.number() }
          },
          async ({ a, b }) => ({
            content: [{ type: 'text', text: String(a + b) }]
          })
        );

        // Add a dynamic greeting resource
        server.registerResource(
          'greeting',
          new ResourceTemplate('greeting://{name}', { list: undefined }),
          {
            title: 'Greeting Resource',
            description: 'Dynamic greeting generator'
          },
          async (uri, { name }) => ({
            contents: [{ uri: uri.href, text: `Hello, ${name}!` }]
          })
        );

        // Add a code review prompt
        server.registerPrompt(
          'review',
          {
            title: 'Code Review',
            description: 'Review code for best practices and potential issues',
            argsSchema: { code: z.string() }
          },
          ({ code }) => ({
            messages: [{ role: 'user', content: { type: 'text', text: `code: ${code}` } }]
          })
        );

        // Create an MCP Client
        const client = new WebMcpClient({ name: 'demo-client', version: '1.0.0' });

        // Connect the client and server
        await server.connect(serverTransport);
        await client.connect(clientTransport);

        // Client callTool, readResource, and getPrompt
        console.log(await client.callTool({ name: 'add', arguments: { a: 5, b: 6 } }));
        console.log(await client.readResource({ uri: 'greeting://John' }));
        console.log(await client.getPrompt({ name: 'review', arguments: { code: 'x' } }));

        // Connect to the Web Agent server
        const { transport, sessionId } = await client.connect({
          url: 'https://agent.opentiny.design/api/v1/webmcp-trial/mcp',
          agent: true
        });

        // Display the session ID
        document.getElementById('sessionId').innerText = sessionId;

        window.addEventListener('pagehide', async () => {
          await transport.terminateSession();
        });
      }
    </script>
  </head>
  <body>
    <button onclick="connect()">Connect Web Agent</button>
    https://agent.opentiny.design/api/v1/webmcp-trial/mcp?sessionId=<span id="sessionId"></span>
  </body>
</html>
```

Remote MCP Host add the above MCP Server url

Our WebMCP Proposal: SDK Demo



Generative UI and MCP servers are the future of Web AI

THANKS