Contents

SATO SPATIAL DOJO - COMPREHENSIVE PATENT SPECIFICATION

Title: Cross-Store Polymorphic Application System with Spatial Navigation and Universal Component Architecture
Applicant: Chrry AI (Iliyan Velinov)
Filing Date: January 2026
Patent Type: Software Process, System Architecture, UI/UX Pattern & Cross-Platform Framework
Classification: G06F 8/77 (Software Testing), G06F 3/0481 (Spatial Navigation), G06N 3/00 (AI/ML), G06F 9/445 (Program Loading)

ABSTRACT

A revolutionary polymorphic application system that generates infinite independent applications across multiple platforms (web PWA, Chrome extension, desktop via Tauri, mobile via Capacitor) from a single codebase through:

1. **Cross-Store Spatial Navigation**: Apps exist in "stores" (domains) AND "base apps" (contexts), navigable via X-axis (app-to-app), Y-axis (store-to-store), and Z-axis (code depth via `.sushi` directory)
2. **Polymorphic Build System**: Single codebase generates 12+ variants per platform via environment variables (MODE=atlas, MODE=vault, etc.)
3. **Universal Component Library**: 151+ components work across web, native, and extensions via SCSS-to-universal-styles converter
4. **Inheritance-Based Architecture**: Apps `extend` parent apps (e.g., FightClub extends Popcorn extends Chrry), inheriting features, tools, and system prompts

5. **Domain-Based Context Switching**: Same app behaves differently based on domain (vex.chrry.ai vs chrry.ai/vex)

Live System: Production deployment at https://chrry.ai with 12+ apps × 5 platforms = 60+ unique installable applications from one repository.

UNIFIED CLAIMS

Claim 1: Cross-Store Polymorphic Architecture with Spatial Coordinates

A computer-implemented system for generating infinite independent applications from a single codebase, comprising:

A. Store Hierarchy (Y-Axis Navigation)

```
interface Store {
  id: string
  slug: string // URL identifier
  name: string // Display name
  domain: string // Primary domain (vex.chrry.ai)
  parentStoreId?: string // Nested store support
  appId: string // Default "base app" for this store
  visibility: "public" | "private"
}

// Example Hierarchy:
// Blossom (chrry.ai) → Compass (atlas.chrry.ai) → Amsterdam (amsterdam.atlas.chrry.ai)
```

B. App Positioning (X-Axis Navigation)

```
interface App {
  id: string
  slug: string // URL identifier
  name: string // Display name
  storeId: string // Primary store (Y-axis position)
  extends: string[] // Parent apps (inheritance chain)
  domain?: string // Optional dedicated domain

  // Cross-platform builds
  installType: "web" | "hybrid" | "native"

  // Platform-specific
  appStoreUrl?: string // iOS App Store
  playStoreUrl?: string // Google Play
  bundleId?: string // iOS identifier
  packageName?: string // Android identifier
}

// Example:
// FightClub (movies.chrry.ai/fightClub)
// extends: [Popcorn, Chrry]
// Available at: movies.chrry.ai/fightClub AND fightclub.chrry.ai
```

C. Code Depth Navigation (Z-Axis)

```
interface SushiDirectory {
  appId: string
```

```
  path: ".sushi/"

  contents: {
    "DNA.md": string // Project structure
    "mutations/": Mutation[] // Test results
    "agents/": AgentState[] // AI agent XP/level
  }
}

// Deep linking:
// vex.chrry.ai → Surface (app interface)
// vex.chrry.ai/.sushi → Depth (project metadata)
// vex.chrry.ai/.sushi/mutations/2026-01-08.json → Specific mutation
```

D. Spatial Coordinate System

```
interface SpatialCoordinate {
  x: string // App position (vex, atlas, vault)
  y: string // Store position (chrry.ai, vex.chrry.ai, atlas.chrry.ai)
  z: string // Depth level (/, /.sushi, /.sushi/mutations)

  // Computed properties
  url: string // Full URL combining x+y+z
  context: AppContext // Resolved app behavior
}

function resolveCoordinate(coord: SpatialCoordinate): AppContext {
  const store = getStore(coord.y)
  const app = getApp(coord.x) || store.defaultApp
  const depth = parseDepth(coord.z)

  return {
    app: resolveInheritance(app), // Merge parent features
    store,
    depth,
    url: `${coord.y}${coord.x}${coord.z}`,
  }
}
```

Technical Implementation:

1. Domain-Based Store Resolution

```
// Server-side routing (apps/api/index.ts)
app.get("*", (req) => {
  const hostname = req.headers.get("host") // vex.chrry.ai
  const path = new URL(req.url).pathname // /atlas

  // Y-axis: Resolve store from domain
  const store =
    stores.find((s) => s.domain.includes(hostname)) ||
    getStoreBySlug(hostname.split(".")[0])

  // X-axis: Resolve app from path or store default
  const appSlug = path.split("/")[1] || store.appId
  const app = getApp(appSlug)
```

```
  // Z-axis: Check for depth navigation
  const depth = path.includes("/.sushi") ? parseSushiPath(path) : null

  return renderApp({ app, store, depth })
})
```

2. Cross-Platform Polymorphic Builds

```
// apps/extension/package.json
{
  "scripts": {
    "build:chrome:atlas": "MODE=atlas vite build && cd dist && zip -r ../dist-chrome-atlas.zip .",
    "build:chrome:vault": "MODE=vault vite build && cd dist && zip -r ../dist-chrome-vault.zip .",
    "build:chrome:vex": "MODE=vex vite build && cd dist && zip -r ../dist-chrome-vex.zip .",
    "build:chrome": "npm run build:chrome:atlas && npm run build:chrome:vault && npm run build:chrome:v
  }
}

// Vite config reads MODE env var
const mode = process.env.MODE || "vex"
const app = getApp(mode)

export default defineConfig({
  define: {
    "import.meta.env.APP_NAME": JSON.stringify(app.name),
    "import.meta.env.APP_ICON": JSON.stringify(app.icon),
    "import.meta.env.APP_THEME": JSON.stringify(app.themeColor)
  }
})
```

Result: 12 apps $\times$ 5 platforms = 60+ unique builds from one codebase!

3. App Inheritance Chain

```
// Database schema (packages/db/src/schema.ts)
export const apps = pgTable("apps", {
  id: text("id").primaryKey(),
  slug: text("slug").notNull().unique(),
  extends: text("extends").array(), // Parent app IDs
  systemPrompt: text("system_prompt"),
  tools: text("tools").array(), // ["calendar", "location"]
  features: jsonb("features"), // {moodTracking: true}
})

// Runtime inheritance resolution
async function resolveInheritance(app: App): Promise<ResolvedApp> {
  if (!app.extends?.length) return app

  const parents = await Promise.all(app.extends.map((id) => getApp(id)))

  return {
    ...mergeDeep(parents), // Merge parent features
    ...app, // Override with child features
  }
}
```

```
// Example:
// FightClub extends Popcorn extends Chrry
// Inherits: Chrry's calendar tools + Popcorn's movie analysis + FightClub's philosophy
```

4. Store-App Installation Matrix

```
// packages/db/src/schema.ts
export const storeInstalls = pgTable("store_installs", {
  storeId: text("store_id").references(() => stores.id),
  appId: text("app_id").references(() => apps.id),
  featured: boolean("featured").default(false),
  displayOrder: integer("display_order"),
  customDescription: text("custom_description"),
})

// Apps can exist in multiple stores with different descriptions!
// Example:
// Atlas installed in:
// - Blossom (chrry.ai) - "Travel planning app"
// - Compass (atlas.chrry.ai) - "Your primary navigation tool"
// - LifeOS (vex.chrry.ai) - "Integrated travel assistant"
```

Claim 2: Universal Component System with Platform Abstraction

A cross-platform UI component library that enables write-once-run-anywhere development:

A. SCSS to Universal Styles Conversion

```
// scripts/scss-to-universal.js
function convertScssToUniversal(scssPath: string) {
  const scss = fs.readFileSync(scssPath, "utf-8")
  const ast = parseSCSS(scss)

  const universalStyles = {
    web: convertToCSS(ast),
    native: convertToReactNativeStyles(ast),
    extension: convertToCSS(ast, { scopePrefix: "chrry-" }),
  }

  // Generate platform-specific style files
  fs.writeFileSync(
    scssPath.replace(".scss", ".styles.ts"),
    generateUniversalStylesCode(universalStyles),
  )
}

// Generated output:
export const buttonStyles = Platform.select({
  web: { padding: "12px 24px", borderRadius: "8px" },
  native: { padding: 12, borderRadius: 8 },
  default: { padding: "12px 24px", borderRadius: "8px" },
})
```

B. Platform Detection & Conditional Rendering

```
// packages/ui/src/Platform.ts
```

```
export const Platform = {
  OS: detectPlatform(),  // "web" | "ios" | "android" | "windows" | "macos"

  select<T>(options: {
    web?: T
    native?: T
    ios?: T
    android?: T
    default: T
  }): T {
    return options[this.OS] || options.native || options.default
  }
}

// Usage in components:
const Button = ({ label, onClick }) => {
  const Component = Platform.select({
    web: "button",
    native: Touchable,
    default: "button"
  })

  return <Component onClick={onClick}>{label}</Component>
}
```

C. Universal Component Library (151+ Components)

```
// packages/ui/index.ts exports:
export { Button, Input, Card, Modal, Dropdown, Tabs, Table, Form, ... } // 151+ components

// Each component has:
// 1. ComponentName.tsx - Logic (platform-agnostic)
// 2. ComponentName.scss - Styles (converted to universal)
// 3. ComponentName.stories.tsx - Storybook demo
// 4. ComponentName.test.tsx - Unit tests

// Example: Button component works everywhere
import { Button } from "@chrryai/chrry"

// Web PWA: renders <button> with CSS
<Button label="Click me" />

// React Native: renders <Touchable> with RN styles
<Button label="Click me" />

// Chrome Extension: renders <button> with scoped CSS
<Button label="Click me" />

// Tauri Desktop: renders <button> with CSS
<Button label="Click me" />
```

D. Build Pipeline Integration

```
// turbo.json - Monorepo build orchestration
{
  "pipeline": {
```

```
    "build": {
      "dependsOn": ["^build"], // Build dependencies first
      "outputs": ["dist/**"]
    },
    "s:all": {
      // Convert all SCSS to universal styles
      "cache": false,
      "dependsOn": []
    }
  }
}

// Development workflow:
// 1. Edit Button.scss
// 2. Run: pnpm s:all (converts to universal styles)
// 3. Run: turbo build (builds all platforms)
// 4. Result: Button works on web, native, extension, desktop
```

Claim 3: Polymorphic Build System with Environment-Based Variants

A build system that generates multiple independent applications from a single codebase using environment variables and dynamic configuration:

A. Environment-Driven App Configuration

```
// apps/extension/vite.config.ts
const mode = process.env.MODE || "vex" // atlas, vault, vex, focus, etc.
const app = await getApp({ slug: mode })

export default defineConfig({
  define: {
    // Inject app metadata at build time
    "import.meta.env.APP_ID": JSON.stringify(app.id),
    "import.meta.env.APP_NAME": JSON.stringify(app.name),
    "import.meta.env.APP_ICON": JSON.stringify(app.icon),
    "import.meta.env.APP_THEME": JSON.stringify(app.themeColor),
    "import.meta.env.APP_BACKGROUND": JSON.stringify(app.backgroundColor),
    "import.meta.env.APP_DOMAIN": JSON.stringify(app.domain),
  },

  plugins: [
    // Dynamic manifest generation
    {
      name: "dynamic-manifest",
      generateBundle() {
        this.emitFile({
          type: "asset",
          fileName: "manifest.json",
          source: JSON.stringify({
            name: app.name,
            icons: { "128": app.icon },
            theme_color: app.themeColor,
            background_color: app.backgroundColor,
          }),
        })
```

```
      },
    },
  ],
})
```

B. Runtime App Context Resolution

```
// apps/flash/src/entry-server.jsx (Vite SSR)
export async function render(url: string, manifest: string) {
  const { hostname, pathname } = new URL(url)

  // Resolve spatial coordinates
  const store = await getStoreByDomain(hostname)
  const appSlug = pathname.split("/")[1] || store.appId
  const app = await getApp({ slug: appSlug })

  // Resolve inheritance chain
  const resolvedApp = await resolveInheritance(app)

  // Generate HTML with app-specific metadata
  const html = renderToString(
    <App
      app={resolvedApp}
      store={store}
      manifest={manifest}
    />
  )

  return {
    html,
    head: `
      <title>${app.name} - ${app.subtitle}</title>
      <meta name="theme-color" content="${app.themeColor}" />
      <link rel="icon" href="${app.icon}" />
    `
  }
}
```

C. Multi-Platform Build Matrix

```
# Chrome Extension Builds (12 variants)
MODE=atlas vite build → dist-chrome-atlas.zip
MODE=vault vite build → dist-chrome-vault.zip
MODE=vex vite build → dist-chrome-vex.zip
MODE=focus vite build → dist-chrome-focus.zip
MODE=popcorn vite build → dist-chrome-popcorn.zip
MODE=chrry vite build → dist-chrome-chrry.zip
MODE=zarathustra vite build → dist-chrome-zarathustra.zip
MODE=search vite build → dist-chrome-search.zip
MODE=grape vite build → dist-chrome-grape.zip
MODE=burn vite build → dist-chrome-burn.zip
MODE=pear vite build → dist-chrome-pear.zip
MODE=sushi vite build → dist-chrome-sushi.zip

# Tauri Desktop Builds (12 variants × 3 OS)
MODE=atlas tauri build --target universal-apple-darwin → Atlas.app (macOS)
```

```
MODE=atlas tauri build --target x86_64-pc-windows-msvc → Atlas.exe (Windows)
MODE=atlas tauri build --target x86_64-unknown-linux-gnu → Atlas.AppImage (Linux)
... × 12 apps = 36 desktop builds

# React Native Mobile (12 variants × 2 platforms)
MODE=atlas npm run build:ios → Atlas.ipa
MODE=atlas npm run build:android → Atlas.apk
... × 12 apps = 24 mobile builds

# PWA Builds (12 variants, single deploy)
All apps served from single Vite SSR server with dynamic routing

Total: 12 extensions + 36 desktop + 24 mobile + 12 PWA = 84 unique applications from ONE codebase!
```

D. Shared Component Reusability

```
// packages/ui/App.tsx - Works across ALL platforms
export const App = ({ app, store }) => {
  return (
    <AppProvider app={app} store={store}>
      <Navigation />
      <ChatInterface />
      <Sidebar />
    </AppProvider>
  )
}

// Platform-specific entry points just wrap the universal App:

// apps/flash/src/entry-client.jsx (Web PWA)
hydrateRoot(document.getElementById("app"), <App {...props} />)

// apps/extension/src/main.tsx (Chrome Extension)
createRoot(document.getElementById("app")).render(<App {...props} />)

// apps/browser/src/main.tsx (Tauri Desktop)
createRoot(document.getElementById("app")).render(<App {...props} />)

// apps/mobile/App.tsx (React Native)
registerRootComponent(() => <App {...props} />)
```

Claim 4: Cross-Store Installation and Discovery System

A system for installing applications across multiple stores with customized descriptions and positioning:

A. Store-App Many-to-Many Relationship

```
// An app can be installed in multiple stores
// A store can have multiple apps
// Each installation has custom metadata

interface StoreInstall {
  storeId: string
  appId: string
  featured: boolean // Show in featured section
  displayOrder: number // Position in store
```

```
  customDescription?: string // Store-specific description
  customIcon?: string // Store-specific icon
}

// Example: Atlas app exists in 3 stores with different descriptions
await createStoreInstall({
  storeId: "blossom", // chrry.ai
  appId: "atlas",
  featured: true,
  displayOrder: 3,
  customDescription:
    "Your intelligent travel companion for exploring the world",
})

await createStoreInstall({
  storeId: "compass", // atlas.chrry.ai
  appId: "atlas",
  featured: true,
  displayOrder: 0, // Primary app in this store
  customDescription:
    "Plan trips, discover destinations, and navigate like a local",
})

await createStoreInstall({
  storeId: "lifeOS", // vex.chrry.ai
  appId: "atlas",
  featured: true,
  displayOrder: 5,
  customDescription: "Integrated travel planning for your AI-powered life",
})
```

B. Store Discovery Flow

```
// User navigation flow:
// 1. Visit chrry.ai (Blossom store)
// 2. Browse apps (Vex, Atlas, Bloom, Vault)
// 3. Click "Atlas" → Navigate to atlas.chrry.ai OR chrry.ai/atlas
// 4. Atlas appears with compass.chrry.ai branding (primary store)
// 5. User can install Atlas PWA, extension, or navigate back to Blossom

// Store switching logic
function navigateToApp(app: App, currentStore: Store) {
  if (app.domain) {
    // App has dedicated domain, navigate there
    window.location.href = app.domain
  } else if (app.storeId !== currentStore.id) {
    // App belongs to different store, navigate to that store
    const targetStore = getStore(app.storeId)
    window.location.href = `${targetStore.domain}/${app.slug}`
  } else {
    // Same store, just navigate to app path
    router.push(`/${app.slug}`)
  }
}
```

C. Cross-Store App Relationships

```
// Example: Books store (zarathustra.chrry.ai)
const booksStore = {
  id: "books",
  slug: "books",
  domain: "https://zarathustra.chrry.ai",
  appId: "zarathustra", // Default app for this store
  parentStoreId: "blossom",
}

// Zarathustra is the BASE app for Books store
// But Books store also installs other apps:
await createStoreInstall({
  storeId: "books",
  appId: "1984",
  customDescription: "Orwell's dystopian warning through Zarathustra's lens",
})

await createStoreInstall({
  storeId: "books",
  appId: "meditations",
  customDescription: "Marcus Aurelius's Stoic wisdom meets Nietzsche",
})

await createStoreInstall({
  storeId: "books",
  appId: "dune",
  customDescription: "Herbert's epic examined through philosophical depth",
})

// Now Books store has 4 apps:
// - Zarathustra (base app, philosophy guide)
// - 1984 (extends Zarathustra, dystopian analysis)
// - Meditations (extends Zarathustra, Stoic wisdom)
// - Dune (extends Zarathustra, sci-fi philosophy)
```

D. Inheritance Chain Example

```
// Movies store (popcorn.chrry.ai)
const moviesStore = {
  appId: "popcorn", // Base app
  domain: "https://popcorn.chrry.ai",
}

// Popcorn extends Chrry (marketplace features)
const popcorn = {
  extends: ["chrry"],
  systemPrompt: "You are Popcorn, cinema universe curator...",
  tools: ["calendar", "location", "weather"],
}

// FightClub extends Popcorn (inherits cinema features) + Chrry (marketplace)
const fightClub = {
  extends: ["popcorn", "chrry"],
```

```
  systemPrompt: "You are Fight Club, underground cinema companion...",
  tools: ["calendar", "location", "weather"], // Inherited from Popcorn
  highlights: fightClubInstructions, // Custom to FightClub
}

// Runtime resolution:
const resolvedFightClub = await resolveInheritance(fightClub)
// Result:
// {
//   systemPrompt: "You are Fight Club..." (FightClub's custom prompt),
//   tools: ["calendar", "location", "weather"] (inherited from Popcorn),
//   highlights: [...chrryInstructions, ...popcornInstructions, ...fightClubInstructions],
//   features: { ...chrry.features, ...popcorn.features, ...fightClub.features }
// }
```

Claim 5: Domain-Based Context Switching and URL Routing

A routing system that changes application behavior based on domain and path without separate codebases:

A. Multi-Domain Single-Server Architecture

```
// apps/api/index.ts - Single Hono server handles ALL domains
const app = new Hono()

app.get("*", async (c) => {
  const hostname = c.req.header("host") // vex.chrry.ai, atlas.chrry.ai, etc.
  const path = new URL(c.req.url).pathname

  // Store resolution (Y-axis)
  let store = await getStoreByDomain(hostname)
  if (!store && hostname.includes(".chrry.ai")) {
    // Subdomain like atlas.chrry.ai
    const subdomain = hostname.split(".")[0]
    store = await getStore({ slug: subdomain })
  }
  if (!store) {
    store = await getStore({ slug: "blossom" }) // Default to chrry.ai
  }

  // App resolution (X-axis)
  const appSlug = path.split("/")[1] // /atlas → "atlas"
  let app = appSlug ? await getApp({ slug: appSlug }) : null
  if (!app) {
    app = await getApp({ id: store.appId }) // Use store's default app
  }

  // Render app with store context
  return renderApp({ app, store, path })
})

// Examples:
// chrry.ai → Blossom store, Chrry app
// vex.chrry.ai → LifeOS store, Vex app
// atlas.chrry.ai → Compass store, Atlas app
// chrry.ai/atlas → Blossom store, Atlas app (different context!)
```

```
// vex.chrry.ai/atlas → LifeOS store, Atlas app (yet another context!)
```

B. Context-Aware App Behavior

```
// Same Atlas app behaves differently based on context

// Context 1: atlas.chrry.ai (primary store, standalone)
{
  app: "Atlas",
  store: "Compass",
  navigation: ["Amsterdam", "Tokyo", "Istanbul", "NewYork"],  // Other travel apps
  branding: "Compass" store theme
}

// Context 2: chrry.ai/atlas (marketplace context)
{
  app: "Atlas",
  store: "Blossom",
  navigation: ["Chrry", "Vex", "Vault", "Bloom"],  // Other marketplace apps
  branding: "Blossom" marketplace theme
}

// Context 3: vex.chrry.ai/atlas (integrated into Vex)
{
  app: "Atlas",
  store: "LifeOS",
  navigation: ["Vex", "Atlas", "Bloom", "Vault", "Focus"],  // LifeOS suite
  branding: "LifeOS" integrated theme
}

// Implementation:
function renderApp({ app, store, path }) {
  const context = {
    currentApp: app,
    currentStore: store,
    navigation: getNavigationForStore(store),  // Different per store!
    theme: store.theme,
    showMarketplaceFeatures: store.slug === "blossom"
  }

  return <App context={context} />
}
```

C. URL Pattern Mapping

```
// URL patterns and their resolutions:

// Pattern 1: Dedicated domain (primary store context)
"atlas.chrry.ai" → Store: Compass, App: Atlas, Context: Primary

// Pattern 2: Subdomain + path (integrated context)
"chrry.ai/atlas" → Store: Blossom, App: Atlas, Context: Marketplace

// Pattern 3: Store domain + app path (integrated context)
"vex.chrry.ai/atlas" → Store: LifeOS, App: Atlas, Context: Integrated
```

```
// Pattern 4: Nested store + app
"movies.chrry.ai/fightClub" → Store: Movies, App: FightClub, Context: Cinema

// Pattern 5: Deep link with Z-axis
"vex.chrry.ai/.sushi/mutations" → Store: LifeOS, App: Vex, Depth: Mutations

// All patterns handled by SINGLE server with SINGLE codebase!
```

D. Cross-Store App Installation Discovery

```
// User journey:
// 1. Visit chrry.ai (Blossom store)
// 2. See Atlas in app grid (installed in Blossom via storeInstalls)
// 3. Click Atlas
// 4. Browser checks: Does Atlas have dedicated domain?
//    - Yes: Navigate to atlas.chrry.ai (primary store context)
//    - No: Navigate to chrry.ai/atlas (marketplace context)

function handleAppClick(app: App, currentStore: Store) {
  // Check if app has dedicated domain
  if (app.domain) {
    // Navigate to primary store
    window.location.href = app.domain
    return
  }

  // Check if app belongs to different store
  if (app.storeId !== currentStore.id) {
    const targetStore = getStore(app.storeId)
    if (targetStore.domain !== window.location.hostname) {
      // Navigate to app's primary store
      window.location.href = `${targetStore.domain}/${app.slug}`
      return
    }
  }

  // Same store, just navigate to app path
  router.push(`/${app.slug}`)
}
```

TECHNICAL DRAWINGS

Figure 1: Cross-Store Spatial Navigation

```
Spatial Coordinate System (X, Y, Z Axes)


 Y-AXIS (Stores/Domains)
   chrry.ai (Blossom - Marketplace)
     /chrry (Chrry app)
     /vex (Vex in marketplace context)
     /atlas (Atlas in marketplace context)
```

```
  vex.chrry.ai (LifeOS - AI Suite)
    / (Vex app - default)
    /atlas (Atlas in LifeOS context)
    /focus (Focus productivity app)

  atlas.chrry.ai (Compass - Travel Hub)
    / (Atlas app - primary)
    /amsterdam (Amsterdam guide)
    /tokyo (Tokyo guide)
    /istanbul (Istanbul guide)

  movies.chrry.ai (Popcorn - Cinema)
     / (Popcorn app - default)
     /fightClub (Fight Club analysis)
     /inception (Inception guide)
     /pulpFiction (Pulp Fiction analysis)

X-AXIS (Apps within stores)
  Chrry (marketplace creator)
  Vex (general AI assistant)
  Atlas (travel companion)
  Popcorn (cinema guide)
  FightClub (extends Popcorn)

Z-AXIS (Code depth)
  / (surface - app interface)
  /.sushi (depth - project metadata)
  /.sushi/mutations (deep - test results)
```

Figure 2: Polymorphic Build Matrix

```
Single Codebase → 84 Unique Applications


ONE Repository (github.com/chrryai/vex)
  apps/
    flash/        (PWA - 12 apps)
    extension/    (Chrome - 12 apps × 1 = 12)
    browser/      (Tauri - 12 apps × 3 OS = 36)
    mobile/       (Capacitor - 12 apps × 2 = 24)

  packages/
     ui/           (151+ universal components)
     db/           (Drizzle ORM + schema)
     pepper/       (Universal router)

BUILD PROCESS:

  MODE=atlas vite build
    ↓
  Dynamic app config injection
    ↓
```

```
  Platform-specific optimizations
    ↓
    dist-chrome-atlas.zip (Chrome extension)
    Atlas.app (macOS)
    Atlas.exe (Windows)
    Atlas.AppImage (Linux)
    Atlas.ipa (iOS)
    Atlas.apk (Android)


  RESULT: 6 platforms × 12 apps = 72 builds
  (Plus 12 PWA variants served dynamically = 84 total)
```

Figure 3: App Inheritance Chain

```
  Multi-Level App Inheritance


  Chrry (Base)
    Features: App marketplace, store creation
    Tools: calendar, location, weather
    System Prompt: "You are Chrry, AI App Marketplace..."

      → Vex (extends Chrry)
           Inherits: marketplace features, basic tools
           Adds: multi-agent chat, artifacts, memory
           System Prompt: "You are Vex, AI-powered life..."

              → Focus (extends Vex + Chrry)
                   Inherits: Vex's chat + Chrry's tools
                   Adds: task management, timers, mood
                   Prompt: "You are Focus, productivity..."

              → Atlas (extends Vex + Chrry)
                   Inherits: Vex's chat + Chrry's tools
                   Adds: travel planning, location search
                   Prompt: "You are Atlas, travel..."

                      → Amsterdam (extends Atlas + Vex)
                           Inherits: All parent features
                           Adds: Local knowledge, tips
                           Prompt: "Amsterdam Guide..."

      → Popcorn (extends Chrry)
           Inherits: marketplace features
           Adds: cinema analysis, scene breakdown
           System Prompt: "You are Popcorn, cinema..."

              → FightClub (extends Popcorn + Chrry)
                   Inherits: Cinema analysis tools
                   Adds: Philosophy, psychology
                   Prompt: "Fight Club underground..."
```

```
    → Zarathustra (extends Chrry)
         Inherits: marketplace features
         Adds: Philosophy, Nietzsche, life guidance
         System Prompt: "You are Zarathustra, prophet..."

            → 1984 (extends Zarathustra + Chrry)
                 Inherits: Philosophy framework
                 Adds: Dystopian analysis, Orwell
                 Prompt: "1984 Guide, dystopian..."

            → Meditations (extends Zarathustra)
                 Inherits: Philosophy framework
                 Adds: Stoic principles, Marcus Aurelius
                 Prompt: "Meditations Guide, Stoic..."

Runtime Resolution:
FightClub.resolveInheritance() → {
  tools: ["calendar", "location", "weather"],  // From Chrry
  features: {
    marketplace: true,         // From Chrry
    sceneAnalysis: true,       // From Popcorn
    philosophy: true,          // From FightClub
  },
  systemPrompt: "You are Fight Club..." // FightClub's own
}
```

Figure 4: Cross-Store Installation Matrix

```
Store-App Many-to-Many Relationship


STORES (Y-axis):

  Blossom      LifeOS       Compass      Movies
  chrry.ai     vex.chrry    atlas.chrry  popcorn.


APPS (X-axis) installed across stores:

Chrry:
  Blossom   (primary, featured, order=0)
  LifeOS   (featured, order=1, "Create AI apps")
  Compass   (order=2, "Build travel apps")
  Movies   (order=1, "Create cinema apps")

Vex:
  Blossom   (featured, order=2, "AI-powered platform")
  LifeOS   (primary, featured, order=0)
  Compass   (order=5, "General AI assistant")
  Movies   (order=5, "Non-cinema tasks")
```

```
Atlas:
  Blossom   (featured, order=3, "Travel companion")
  LifeOS   (featured, order=5, "Integrated travel")
  Compass   (primary, featured, order=0)

Popcorn:
  Blossom   (featured, order=2, "Cinema universe")
  Movies   (primary, featured, order=0)

FightClub:
  Blossom   (not installed)
  Movies   (featured, order=1, "Underground cinema")

NAVIGATION BEHAVIOR:

 User on: chrry.ai (Blossom)
 Clicks: Atlas
 → Navigates to: atlas.chrry.ai (Atlas's primary store)

 User on: vex.chrry.ai (LifeOS)
 Clicks: Atlas
 → Navigates to: vex.chrry.ai/atlas (integrated view)

 User on: movies.chrry.ai (Movies)
 Clicks: Vex
 → Navigates to: vex.chrry.ai (Vex's primary store)


RESULT: Same app, different contexts based on store!
```

PRIOR ART DIFFERENTIATION

Feature	Heroku Apps	Docker Compose	Vercel Projects	**Sato Spatial System**
Single Codebase	(one app)	(multi-service)	(one app)	
Cross-Platform	(web only)	(containers)	(web only)	(web, native, ext, desktop)
Spatial Navigation				(X/Y/Z axes)
App Inheritance				(extend parent apps)
Store Hierarchy				(nested stores)
Cross-Store Install				(many-to-many)
Universal Components				(151+ components)
Dynamic Builds				(MODE env var)
Domain-Based Context				(same app, diff behavior)
84+ Apps from One Repo				

LIVE SYSTEM EVIDENCE

Production Deployment: https://chrry.ai
Source Code: https://github.com/chrryai/vex
Extension Builds: 12 Chrome extensions published
Desktop Builds: Tauri apps for macOS/Windows/Linux
Mobile Builds: React Native apps for iOS/Android

Demonstrated Features:

1. **Cross-Store Navigation**: chrry.ai → vex.chrry.ai → atlas.chrry.ai
2. **App Inheritance**: FightClub extends Popcorn extends Chrry
3. **Polymorphic Builds**: 12 apps × 7 platforms = 84 unique applications
4. **Universal Components**: 151+ components work everywhere
5. **Domain-Based Context**: atlas.chrry.ai vs chrry.ai/atlas
6. **Store-App Matrix**: Atlas in Blossom, LifeOS, and Compass stores

---

COMPETITIVE SAFEGUARDS

1. Proprietary Build Pipeline

```
// scripts/build-all-platforms.sh (encrypted in production)
export MODE=atlas
pnpm build:chrome      # Chrome extension
pnpm build:desktop     # Tauri (macOS/Windows/Linux)
pnpm build:mobile      # React Native (iOS/Android)
# Result: 6 platforms built with one command

# Pipeline requires:
# - Monorepo structure (Turbo)
# - Universal component library (@chrryai/chrry)
# - Database schema (Drizzle ORM)
# - Dynamic routing (Hono)
# All proprietary, all integrated
```

2. Database Schema Lock

```
// packages/db/src/schema.ts
// Stores, apps, storeInstalls, inheritance chains
// Without this schema, the system doesn't work
// Schema is AGPL-3.0 (copyleft - share modifications)
```

3. Domain Verification

```
const AUTHORIZED_DOMAINS = [
  "chrry.ai",
  "vex.chrry.ai",
  "atlas.chrry.ai",
  "zarathustra.chrry.ai",
  // ... all subdomains
]

function validateDomain() {
  const hostname = window.location.hostname
  if (!AUTHORIZED_DOMAINS.some((d) => hostname.endsWith(d))) {
```

```
    throw new Error("Unauthorized domain")
  }
}
```

FILING STRATEGY

Recommended Approach

Phase 1: Immediate Protection

1. File this comprehensive provisional ($130 micro-entity)
2. Publish on GitHub (open source, defensive publication)
3. Submit to Prior Art Archive (free, legally recognized)

Phase 2: Market Validation (6-12 months)

- Monitor adoption and competitor activity
- Track unique aspects being copied
- Gather user testimonials and case studies

Phase 3: Decision Point (Month 12)

- **If high traction**: Convert to full patent ($10k-15k)
- **If moderate traction**: File PCT for international ($5k+)
- **If low traction**: Let provisional expire, keep trade secret

Budget Path

- **Month 0**: Provisional filing ($130)
- **Month 6**: Public disclosure (GitHub + blog posts)
- **Month 12**: Decide full patent vs trade secret

INVENTOR DECLARATION

I, **Iliyan Velinov**, declare that I am the sole inventor of the "Sato Spatial Dojo" cross-store polymorphic application system. This invention combines:

1. **Spatial Navigation**: X-axis (apps), Y-axis (stores), Z-axis (code depth)
2. **Polymorphic Builds**: 84+ unique apps from one codebase
3. **Universal Components**: 151+ components across 5 platforms
4. **App Inheritance**: Extend parent apps (FightClub → Popcorn → Chrry)
5. **Cross-Store Installation**: Apps exist in multiple stores with custom contexts
6. **Domain-Based Routing**: Same app, different behavior based on domain

All components are live at: https://chrry.ai with source code at: https://github.com/chrryai/vex

Signature: *Iliyan Velinov*
Date: January 21, 2026

CONCLUSION

The **Sato Spatial Dojo** represents a paradigm shift in application architecture:

One Codebase, 84+ Apps: Polymorphic builds across 5 platforms
Spatial Navigation: X-axis (apps), Y-axis (stores), Z-axis (depth)

Universal Components: 151+ components work everywhere
App Inheritance: Extend parent apps like OOP classes
Cross-Store Discovery: Apps exist in multiple stores
Domain-Based Context: Same app, different behavior per domain

The result: A unified system where stores (domains), apps (contexts), and code (depth) form a 3D spatial navigation system enabling infinite variations from finite code.

Document Status: Ready for USPTO Provisional Filing
Estimated Cost: $130 (micro-entity)
Priority Date Established: January 21, 2026
Next Step: File online at https://patentcenter.uspto.gov/
Contact: iliyan@chrry.ai

"*The map is the territory, the territory is the code, and the code is everywhere.*" - Sato Philosophy