CE2 Refactoring Plan: FFI to Declarative + Data Unification

Status: Plan **Created**: 2026-01-20 **Target**: Thursday 2026-01-23 **Category**: Architecture

Overview

The CE2 scale transitions work (Powers of Ten zoom) has validated the UX through prototype JavaScript FFI code. This plan outlines refactoring that code to exemplary declarative PureScript using psd3-selection, plus unifying the data model at the database layer.

Part 1: FFI Debt Assessment

Current State

The `ce2-website/src/Viz/` directory contains ~2000+ lines of imperative D3.js in FFI files:

File	Lines	Purpose	Violation
`ScaleTransition.js`	722	Transition animations	Direct DOM mutation, imperative D3
`PackageSetBeeswarm.js`	~300	Beeswarm layout	Imperative render, manual force sim
`BubblePackBeeswarm.js`	~250	Bubble pack view	Same
`PackageSetTimeline.js`	~200	Timeline view	Same
`PackageSetView.js`	~150	Package set view	Same
`Triptych/*.js`	~400	Chord/Matrix/BubblePack panels	Same

Why This Violates Design Philosophy

From CLAUDE-CONTEXT.md:

> "PSD3 takes control from D3" - We don't write imperative D3 code that mutates the DOM directly
> "D3 as calculation engine only" - We use D3's mathematical algorithms but manage state and rendering declaratively

The FFI files contain:

- `d3.select()` / `selectAll()` - should be `T.select` / `T.selectAll`
- `.attr()` / `.style()` chains - should be declarative `T.attr` / `T.style`
- `.transition().duration()` - should use `psd3-simulation` tick system
- `.data().join()` - should be `T.joinData` from psd3-selection
- Manual `.remove()` calls - should be GUP exit via data join

What The Prototype Validated

Despite the architectural violations, the UX is proven:

1. **FadeOut phase**: Non-relevant packages fade to opacity 0, then removed from DOM
2. **PopIn phase**: Remaining packages grow, modules appear inside
3. **MoveToTreemap phase**: Packages glide to treemap positions
4. **Crossfade phase**: Treemap backdrop fades in, beeswarm fades out
5. **Zoom out**: Instant jump back (no animation)

This UX should be preserved while refactoring the implementation.

Part 2: Refactoring Strategy

Principle: Data Drives DOM

Instead of:

```
// WRONG: Imperative
svg.selectAll("g.package-group")
   .filter(d => fadingIds.has(d.id))
   .transition()
   .style("opacity", 0)
   .on("end", function() { d3.select(this).remove(); });
```

Use:

```
-- RIGHT: Declarative
-- State change: remove fading packages from data
let visiblePackages = Array.filter (\p -> not (Set.member p.id fadingIds))
packages
-- Re-render: data join handles exit
T.joinData "packages" "g.package-group" visiblePackages packageTemplate
```

Phase-by-Phase Refactoring

FadeOut Phase

Current (FFI):

- `setOpacityByIdsImpl` sets opacity on matching groups
- `removeFadedPackagesImpl` manually removes DOM elements

Target (Declarative):

1. Transition state includes `fadingIds :: Set Int`
2. Each tick, compute opacity: `if fading then (1.0 - progress) else 1.0`
3. Use `T.attrFn "opacity"` with lookup function

4. At end of phase, update data source to exclude faded IDs
5. Re-join with smaller dataset - GUP exit handles removal

Library needs:

- `T.attrFn` that takes `(d -> a)` already exists
- May need `T.styleFn` for opacity specifically

PopIn Phase

Current (FFI):

- `growCirclesImpl` scales radii
- `renderPackedModulesImpl` adds module circles inside groups

Target (Declarative):

1. Pre-compute packed positions in PureScript (already done via psd3-layout)
2. Package radius is function of progress: `lerp originalR targetR progress`
3. Module circles are nested data join inside package groups
4. Use `T.joinData` for both levels

Library needs:

- Nested data joins (group -> children pattern)
- May already be supported via `T.selectAll` within a join

MoveToTreemap Phase

Current (FFI):

- `moveGroupsToPositions` interpolates transform
- `renderTreemapBackdrop` creates rectangle elements

Target (Declarative):

1. Position is function of progress: `lerp sourcePos targetPos progress`
2. Use `T.attr "transform"` with computed translate
3. Treemap rectangles are separate data join
4. Both driven by tick progress in Halogen state

Library needs:

- Transform attribute helpers (or just string interpolation)

Crossfade Phase

Current (FFI):

- `renderTreemapBackdrop` with opacity
- `setPackageGroupsOpacity` fades out beeswarm

Target (Declarative):

1. Backdrop opacity = progress
2. Package group opacity = 1 - progress
3. Both via `T.styleFn "opacity"`
4. At end, remove package groups by updating data source

Beeswarm/Timeline Views

These are larger refactors but follow same pattern:

1. Layout computation stays in PureScript (force simulation params)
2. D3 force simulation can stay as FFI (it's "calculation engine")
3. Rendering of simulation state becomes declarative data join
4. Tick updates come through Halogen subscription (already have this pattern)

Part 3: Data Unification

Current Data Model

Two separate data sources with different schemas:

packageSetData (from API):

```
type PackageSetPackage =
  { id :: Int
  , name :: String
  , version :: String
  , depends :: Array String
  , topoLayer :: Int
  }
-- 568 packages (full registry)
```

modelData (from static JSON):

```
type SimNode =
  { id :: Int
  , name :: String
  , nodeType :: NodeType  -- PackageNode | ModuleNode
  , package :: String     -- parent package for modules
  , targets :: Array Int  -- dependency IDs
  , ...
  }
-- 139 packages + 645 modules (project analysis)
```

The Problem

Client must reconcile:

- Which of 568 registry packages match our 139 project packages?
- Which are local-only (ce-, *psd3-*) vs registry?
- What's the transitive closure for "Project + Dependencies"?

This computation is scattered across:

- `computeRelevantPackageIds` in App.purs
- `filterNodesByScale` in App.purs
- ID matching by name (brittle)

Proposed Unified Model

Single API endpoint returns pre-computed groupings:

```
GET /api/unified-packages?project=1&packageSet=1
```

Response:

```
{
  "packages": [
    {
      "id": 1,
      "name": "prelude",
      "version": "1.0.0",
      "depends": ["prim"],
      "topoLayer": 0,
      "source": "registry",        // "registry" | "local"
      "inProject": true,           // Is this in our project's deps?
      "moduleCount": 0,            // Modules we analyzed (0 for registry-
only)
      "totalLoc": 0
    },
    {
      "id": 1000,
      "name": "ce2-website",
      "version": "0.0.0",
      "depends": ["prelude", "halogen", ...],
      "topoLayer": 10,
      "source": "local",
      "inProject": true,
      "moduleCount": 45,
      "totalLoc": 12000
    }
  ],
  "groupings": {
    "fullSet": [1, 2, 3, ...],           // All 568 registry
    "transitive": [1, 5, 10, ...],       // Project's transitive deps
(~122)
    "local": [1000, 1001, ...],          // Local packages with source
(~17)
```

```
    "projectExtra": [1000, 1001, ...]    // Same as local for now
  },
  "modules": [
    // Only for local packages
    { "id": 2000, "name": "Main", "package": "ce2-website", "loc": 500,
... }
  ]
}
```

Database Changes

Add view or materialized query:

```
CREATE VIEW unified_packages AS
SELECT
  psp.id,
  psp.name,
  psp.version,
  psp.depends,
  psp.topo_layer,
  'registry' as source,
  EXISTS(SELECT 1 FROM modules m WHERE m.package = psp.name AND
m.snapshot_id = ?) as in_project,
  (SELECT COUNT(*) FROM modules m WHERE m.package = psp.name AND
m.snapshot_id = ?) as module_count,
  (SELECT COALESCE(SUM(loc), 0) FROM modules m WHERE m.package = psp.name
AND m.snapshot_id = ?) as total_loc
FROM package_set_packages psp
WHERE psp.package_set_id = ?

UNION ALL

SELECT
  1000 + ROW_NUMBER() OVER () as id,  -- Synthetic IDs for local packages
  m.package as name,
  '0.0.0' as version,
  '[]' as depends,  -- Would need spago.yaml parsing
  99 as topo_layer,  -- Local packages are "top" of dependency tree
  'local' as source,
  true as in_project,
  COUNT(*) as module_count,
  SUM(m.loc) as total_loc
FROM modules m
WHERE m.snapshot_id = ?
  AND m.package NOT IN (SELECT name FROM package_set_packages WHERE
package_set_id = ?)
GROUP BY m.package;
```

Client Simplification

After unification, client code becomes:

```
-- Load unified data once
unifiedData <- loadUnifiedPackages projectId packageSetId

-- Scale filtering is just set membership
packagesForScale :: ScaleLevel -> Array UnifiedPackage
packagesForScale = case _ of
  PackageSetScale -> filter (\p -> p.source == "registry")
unifiedData.packages
  ProjectDepsScale -> filter (\p -> p.inProject) unifiedData.packages
  ProjectOnlyScale -> filter (\p -> p.source == "local")
unifiedData.packages
```

No more:

- `computeRelevantPackageIds` with fallback heuristics
- Matching packages by name between different ID spaces
- Client-side transitive closure computation

Part 4: Execution Plan for Thursday

Morning: Data Unification (2-3 hours)

1. **Add unified packages endpoint** to ce-server
    - SQL query joining package_set_packages with modules
    - Compute groupings server-side
    - Return single cohesive response
2. **Update Loader.purs** in ce2-website
    - New `fetchUnifiedPackages` function
    - New `UnifiedPackage` type
    - Deprecate separate `fetchPackageSet` / `loadModel`
3. **Simplify App.purs state**
    - Replace `packageSetData` + `modelData` with `unifiedData`
    - Simplify `computeRelevantPackageIds` to set membership
    - Remove name-based ID matching

Afternoon: FFI Refactoring (3-4 hours)

4. **Start with FadeOut phase** (smallest scope)
    - Replace `setOpacityByIdsImpl` with declarative `T.styleFn`
    - Replace `removeFadedPackagesImpl` with data-driven GUP exit
    - Verify transition still works

5. **MoveToTreemap phase**
    - Replace `moveGroupsToPositions` with declarative transform
    - Replace `renderTreemapBackdrop` with data join
6. **PopIn phase** (if time permits)
    - Replace `growCirclesImpl` with declarative radius
    - Replace `renderPackedModulesImpl` with nested data join
7. **Document patterns** for remaining FFI files
    - Checklist for future refactoring sessions
    - Note any library additions needed

End of Day

- All scale transitions work with new unified data
- FadeOut and MoveToTreemap phases are declarative
- Remaining FFI files documented for future refactoring
- Worklog updated with accomplished/remaining

Success Criteria

1. **No new FFI added** - all fixes use library functions
2. **ScaleTransition.js shrinks** - at least FadeOut + MoveToTreemap removed
3. **Client data model simplified** - single unified source
4. **Transitions still work** - UX preserved
5. **Code is exemplary** - could be shown as library usage example

Risks and Mitigations

Risk	Mitigation
Library missing needed function	Add to psd3-selection, don't work around
Performance regression	Benchmark before/after, optimize if needed
Complex nested joins	Start simple, iterate
Database query complexity	Use view/CTE, keep readable

References

- `docs/kb/plans/snazzy-growing-hammock.md` - Original transition plan
- `CLAUDE-CONTEXT.md` - Design philosophy
- `psd3-selection` source - Available declarative primitives
- Current FFI files - What to replace