UID-Explore Event Bus

Stand 27.09.2025

Architektur im Überblick

UID-Explore nutzt eine lose gekoppelte ESM-Architektur. Die Schichten arbeiten wie folgt zusammen:

- **Input** in `12-1_input/parameters` erzeugt und signalisiert Parameteränderungen.
- **Base** in `12-2_base` normalisiert Werte, berechnet die Simulation und publiziert konsistente Zustände.
- **Presentation** in `12-3_presentation` zeichnet Chart, KPI, Vektorrad und reagiert auf Status und Pointer.
- **Bus** in `12-2_base/bus.js` verbindet alles. Er spiegelt zusätzlich in den DOM, damit externe Tools mithören können.

Statt direkter Funktionsaufrufe laufen Aktualisierungen über Bus-Events. Das reduziert Kopplung, erlaubt Debugging über den DOM und verhindert Render-Loops durch saubere Verantwortlichkeiten.

Der Bus

Datei `12-2_base/bus.js`

Der Bus ist minimal und synchron. Er kennt drei primitive Operationen.

```
on(type, handler)          // registriert einen Listener und liefert eine
unsubscribe-Funktion
off(type, handler)         // entfernt einen Listener
emit(type, payload)        // ruft alle Listener auf und spiegelt
zusätzlich als DOM CustomEvent
```

Wichtige Eigenschaften

- Listener sind pro Event-Typ in einer `Map` organisiert. `on` liefert immer eine saubere Abmeldung.
- `emit` ruft Handler synchron auf. Danach wird ein `CustomEvent(type, {detail: payload})` auf `window` gesendet. So kann man auch mit `window.addEventListener` debuggen, ohne intern am Code zu drehen.
- Es gibt einen kleine Debug-Hook, der ausgewählte Events in der Konsole protokolliert. Beispiel aus dem Paket `uid:e:params:ready`, `uid:e:params:change`, `uid:e:model:update`, `uid:e:sim:data`, `uid:e:sim:pointer`, `uid:e:error`, `uid:e:engine:status`.

Beispiel Zuhören im DOM

```
window.addEventListener('uid:e:sim:data', (ev) => {
  console.table(ev.detail.series)
})
```

Datenfluss

Boot

Die Parameter-UI baut ihr DOM auf und sendet `uid:e:params:ready` mit einem anfänglichen `state` Objekt. Parallel erzeugt der Director in `uid.js` die erste Rechnung. Er publiziert `uid:e:engine:status`, plant eine Berechnung über `requestAnimationFrame` und sendet in der Rechenfunktion zuerst `uid:e:model:update` und dann `uid:e:sim:data`.

Chart, KPI und Vektorrad warten auf die ersten Daten. Sobald `uid:e:sim:data` kommt, zeichnen sie.

Interaktion

Bewegt der Nutzer einen Slider, sendet die Parameter-UI `uid:e:params:change` mit `key` und `value`. Der Director normalisiert das Paar über den Katalog, wendet algebraische Kopplungen an und plant eine neue Berechnung. Die Rechenfunktion publiziert wieder `uid:e:model:update` und `uid:e:sim:data`.

Bewegt der Nutzer den Playhead, sendet das Chart `uid:e:sim:pointer` mit einem Index. Marker im Chart und das Vektorrad setzen ihre Markierungen neu, ohne eine Neuberechnung auszulösen.

Schaltet der Nutzer Serien in der Legende, sendet die Legend den Sichtbarkeitszustand über `uid:e:viz:series:state`. Chart, Marker und Linien übernehmen die Sichtbarkeit und zeichnen gefiltert. Ein separater Schalter für Zusatz-Overlays sendet `uid:e:viz:overlays:enable` und steuert Spitzenlinien und Marker.

Director und Engine

Dateien `12-2_base/uid.js`, `12-2_base/engine.js`, `12-2_base/schema.js`

Der Director ist der zentrale Orchestrator.

- Hört auf `uid:e:params:change` und optional auf `uid:e:integrator:set`.
- Normalisiert Werte über `makeCatalog` und `normalizeParams`.
- Erzwingt Konsistenz über algebraische Kopplungen. Beispiele R0 mit beta und gamma, gamma mit D, sigma mit L.
- Plant Berechnungen gebündelt über `requestAnimationFrame`. Viele schnelle Slider-Events führen damit zu einer einzigen sichtbaren Aktualisierung.

- Sendet `uid:e:model:update` mit dem konsolidierten Parameter-Satz und `uid:e:sim:data` mit den Serien. Liefert zusätzlich `uid:e:engine:status` mit Integrator und Step-Anzahl.
- Sendet `uid:e:error` mit einem beschreibenden Typ, wenn die Invariante verletzt wird. Beispiel Drift größer Toleranz.

Die Engine rechnet deterministisch. Unterstützt SIR, SEIR, SIRD, SIRV sowie optional SIS. Integratoren sind Euler, Heun und RK4. Das Ergebnis enthält `series`, Metadaten und einen Drift-Wert.

Präsentation

Chart

Dateien `12-3_presentation/chart/...`

Das Chart hört auf `uid:e:sim:data` und baut intern eine Brücke zwischen Zeitachse und Pixeln. Ein Overlay zeichnet einen dünnen Playhead und schickt bei Mausbewegung `uid:e:sim:pointer`. Marker und Spitzenlinien sind separate Overlays und reagieren zusätzlich auf `uid:e:viz:series:state` und `uid:e:viz:overlays:enable`. Während Slider-Drag dimmen die Spitzenlinien kurz und nehmen danach den normalen Stil wieder an.

KPI

Datei `12-3_presentation/kpi/index.js`

Die KPI-Decks hören auf `uid:e:model:update` und `uid:e:sim:data` und berechnen aus der Registry die anzuzeigenden Werte. Der Modus unterscheidet statische Karten und Live-Karten.

Vektorrad

Dateien `12-3_presentation/vectors/...`

Ein eigener State hält Serien, Pointer, Modell und Parameter. Das Wiring hört breit auf Modell- und Parameter-Events, liest die Ranges aus dem Katalog und malt auf einem Canvas mit korrekter DPR-Skalierung. Das Vektorrad reagiert weich auf Pointer-Änderungen und nutzt die gleichen Zeitreihen wie das Chart.

Event-Referenz nach Familien

Parameter

`uid:e:params:ready`
Sender Parameter-UI. Enthält `state.params` und `state.meta` mit Sprache, Modus, Modell und einem Treiberschlüssel.

`uid:e:params:change`
Sender Parameter-UI. Entweder ein Paar aus `key` und `value` oder ein `bulk` Objekt. Director normalisiert und koppelt. Chart-Linien hören mit und passen einen dezenten Stil für die Drag-Phase an.

Die folgenden Formen sind im Paket für zukünftige Eingabepfade vorgesehen und werden bereits gehört. Aktuell gibt es im Paket keinen Sender.
`uid:e:params:set`, `uid:e:params:update`, `uid:e:params:commit`, `uid:e:params:apply`, `uid:e:params:value`, `uid:e:params:changed`.

Modell

`uid:e:model:update`
Sender Director. Publiziert den konsolidierten Parameter-Satz samt `method`. Konsumenten sind Parameter-UI für UI-Sync, KPI für Kontexte und Vektorrad für Parameter-State.

Simulation

`uid:e:sim:data`
Sender Director. Enthält die Zeitreihen, die Laufzeit und das Zeitschritt-Raster. Chart, KPI, Marker, Linien und Vektorrad reagieren darauf.

`uid:e:sim:pointer`
Sender Playhead-Overlay. Setzt einen Index oder `null`. Marker und Vektorrad übernehmen die Markierung. Chart zeichnet den Playhead passend zur Skala.

Visualisierung

`uid:e:viz:series:state`
Sender Legend. Enthält Sichtbarkeiten für S, E, I, R. Chart und Overlays filtern damit die Zeichnung.

`uid:e:viz:overlays:enable`
Sender Overlays-Schalter. Aktiviert oder deaktiviert Marker und Spitzenlinien.

Engine

`uid:e:engine:status`
Sender Director. Enthält Modell, Integrator und geplante Steps. Das Vektorrad nutzt dies für Ranges und Kontext.

`uid:e:integrator:set`
Kein Sender im Paket. Externe Steuerung kann darüber Integrator-Wechsel auslösen. Der Director übernimmt das nahtlos und rechnet ohne Remount neu.

Formel

`uid:e:formula:mark` und `uid:e:formula:pulse`
Sender Formula-Bridge. Dient dem Markieren der passenden Formel während des Drags sowie einem kurzen Impuls nach Loslassen des Sliders.

Fehler

`uid:e:error`
Sender Director. Liefert `type` und `context`. Beispiel bei zu großem Drift oder bei ungültigen Parametern.

Payload-Gestalt pro Event

Die wichtigsten Formen in knapper, realer Gestalt

```
// params:ready
{ state: { params, meta: { lang, mode, model, driverKey } } }

// params:change
{ key, value }          // oder
{ bulk: { ...mehrere Werte... } }

// model:update
{ N, I0, R0, beta, gamma, D, sigma, mu, nu, measures, dt, T, method }

// sim:data
{ series, N, dt, T }

// sim:pointer
{ idx }                 // oder { idx: null }

// viz:series:state
{ visible: { S, E, I, R } }

// viz:overlays:enable
{ enabled }

// engine:status
{ model, method, steps }

// error
{ type, context }
```

Kopplungen und Invarianten

Die Katalog-Funktion liefert pro Modell Ranges und Defaultwerte. Die Normalisierung hält Eingaben in den Grenzen. Algebraische Kopplungen halten die Größen konsistent. R0 mit beta und gamma, gamma mit D, sigma mit L. Die Engine prüft am Ende auf Drift. Die Toleranz wächst proportional zur Populationsgröße.

Performance

Der Director fasst viele schnelle Änderungen über `requestAnimationFrame` zusammen. Präsentationsmodule nutzen `ResizeObserver` und eine korrekte DPR-Skalierung. Marker und Playhead zeichnen auf separaten Canvas-Overlays. Das vermeidet teure Voll-Redraws und sorgt für eine sehr flüssige Interaktion.

Erweiterungsvorschläge

- Jede Event-Payload kann optional einen `source` Schlüssel tragen. So lassen sich Auslöser unterscheiden. Beispiel `p` für Parameter-UI, `legend` für Sichtbarkeit, `playhead` für Pointer.
- `model:update` kann das Feld `model` ebenfalls mitgeben, damit alle Konsumenten den Kontext ohne zusätzlichen Status kennen.
- `params:*` Unterformen lassen sich für Tastatureingaben und Formelkarten nutzen. Die Wiring-Module hören bereits breit zu.
- Eine zentrale Typdefinition in JSDoc hilft IDEs beim Entwickeln. Das Paket nutzt bereits Muster, die sich leicht anreichern lassen.

Typische Fehlerbilder und Gegenmittel

- Fehlende Import-Mappings für `@uid/base/...` oder `@uid/presentation/...` führen zu Auflösungsfehlern. Die Import-Map muss die Aliasse korrekt auf die Unterordner zeigen.
- Unsubscribe vergessen führt zu Leaks. Alle `on`-Aufrufe liefern Abmelder. Die Module im Paket nutzen diese Konvention korrekt. Daran bei Erweiterungen festhalten.
- Pointer wird nicht gelöscht. Das Playhead-Overlay sendet `{idx: null}` beim Verlassen. Konsumenten sollten `null` tolerieren und sauber löschen.
- Unsaubere Bulk-Updates. Bei größeren Änderungen besser `bulk` nutzen und eine Berechnung bündeln. Der Director kann beides.

Mini-Rezepte

DOM-Debug ohne Eingriff in den Code

```
for (const ev of
['uid:e:params:ready','uid:e:model:update','uid:e:sim:data','uid:e:sim:poin
ter']) {
  window.addEventListener(ev, e => console.log(ev, e.detail))
}
```

Integrator von außen umschalten

```
import { bus } from '@uid/base/index.js'
bus.emit('uid:e:integrator:set', { method: 'rk4' })
```

Parameter gesammelt setzen

```
bus.emit('uid:e:params:change', { bulk: { R0: 2.2, D: 5.5, N: 1_375_650 }
})
```

Quick-Map Dateien und Rollen

- `12-2_base/bus.js` Bus primitiv mit DOM-Spiegel und Debug-Hook
- `12-2_base/schema.js` Katalog, Normalisierung, Kopplungen
- `12-2_base/engine.js` Numerik und Modelle
- `12-2_base/uid.js` Director mit Planung und Publikation
- `12-1_input/parameters/*` Parameter-UI, Slider, Tabs, Formel-Bridge
- `12-3_presentation/chart/*` Renderer, Legend, Marker, Overlays, Playhead
- `12-3_presentation/kpi/*` KPI-Decks und Registry
- `12-3_presentation/vectors/*` Vektorrad State, Canvas, Wiring

Damit ist klar, wie das System zusammenspielt und an welcher Stelle du ansetzen kannst, ohne andere Teile zu stören.