UVL Playground - Erfahrungsbericht

Stefan Vill, Jannis Dommer

January 12, 2024

1 Ansatz: WebAssembly

Wir hatten die Absicht den UVLS nach WebAssembly zu kompilieren. Dies hätte die Möglichkeit geboten, den Language Server auf Client-Seite auszuführen. Folgende Probleme sind dabei aufgekommen:

- Systemaufrufe sind nicht WebAssembly kompatibel. Das ist sowohl für den Code des UVLS selbst problematisch, da dieser viel mit dem Dateisystem agiert. Weiter sind aber auch viele der Dependencies davon betroffen, gerade durch die Verwendung von Tokio. Ein Ersatz für die Bibliotheken zu finden stellt sich als Herausforderung dar, vor allem da große Teile des UVLS dann hätten umgeschrieben werden müssen.
- Die Komponenten Tree-Sitter und Z3 sind nicht direkt Teil des Rust-Projektes. Tree-Sitter ist eine C-Library die per unsafe-Rust angesprochen wird und seperat kompiliert werden muss. Z3 wird nur die Binary verwendet, was für uns die Notwendigkeit geschaffen hätte selbst ein Kompilat für WebAssembly zu erzeugen.
- Es hätten viele Bindings erstellt werden müssen. Generell hätte sich viel an den Kommunikations-Interfaces ändern müssen.

Insgesamt hätte der Umbau einen Fork des UVLS erzwungen, wovon wir abgesehen hatten, da der UVLS ja durchaus noch in Entwicklung ist/war.

2 Ansatz: UVLS als Server hosten

Die UVLS-Binary wird unverändert genutzt. Allerdings wird ein JavaScript-Wrapper verwendet, der die Kommunikation von Standard-In/Standard-Out auf WebSockets tunnelt.

Damit bleibt die Frage offen, ob weiterhin pro User ein UVLS-Prozess genutzt wird, oder ob sich auch ein UVLS für mehrere User nutzen lässt. Letzteres wäre der Skalierbarkeit zuträglich.

Um einen "geteilten" UVLS umzusetzen, muss der Wrapper dann die Antworten des Servers den entsprechenden Clients zuordnen. Um das zu realisieren, hatten wir versucht einen Multiplexer zu konstruieren, der anhand der IDs aus den JSON-RPC Requests die Zugehörigkeit ableitet. Das Problem dabei war, dass sich nicht alle Responses ihrer Requests zuordnenbar waren. Dies war meist auf fehlende IDs zurückzuführen, sei dies auf Grund der Implementierung des UVLS oder der JSON-RPC 2.0 Spezifikation. Zwar gab es nach einiger Zeit einen "Working Prototype", allerdings war dieser recht instabil, was durch die Konsolidierung alle Clients betraf.

Auf Grund dieser Schwierigkeiten, haben wir uns entschlossen weiterhin auf einen UVLS Prozess pro Client zu setzen. Dies ist auch der Grund, weshalb die Konfigurations-Ansicht nicht Teil des Playground wurde, da jeder UVLS-Prozess einen WebServer spawnt um diese darzustellen. In Folge hätte damit pro User ein weiterer Port auf der Host-Maschine freigeschaltet werden müssen, was sich bezüglich der Containerisierung mit Docker und Traefik als Reverseproxy nicht realisieren lies. Allerdings hätte der Ansatz mit einem geteilten UVLS hier auch nicht geholfen, da durch den State-Share alle Ansichten über Clients hinweg aktualisiert werden würden.

3 Ausblick

Skalierbarkeit stellt vermutlich durch die Implementierung eine Herausforderung dar. Allerdings würden wir hier eher den Ansatz verfolgen den UVLS doch irgendwie in den Client zu portieren, anstatt den Backend-Server darauf zu optimieren. Dies könnte abgesehen von dem WebAssembly Ansatz, den wir vorsahen, auch durch einen JavaScript Parser oder andere Wege umgesetzt werden.

Weiter sind die Implikationen in Richtung Security durch die Verbindung des UVLS an eine Netzwerkschnittstelle durch uns nicht absehbar.