Contents

Chapter 1

Evaluation

Section **??** described the implementation of the Nickel Language Server addressing the first research question stated in sec. **??**. Proving the viability of the result and answering the second research question demands an evaluation of different factors.

Earlier, the most important metrics of interest were identified as:

Usability What is the real-world value of the language server?
Does it improve the experience of developers using Nickel? NLS offers several features, that are intended to help developers using the language. The evaluation should assess whether the server does improve the experience of developers.
Does NLS meet its users' expectations in terms of completeness, correctness and behavior? Labeling NLS as a Language Server, invokes certain expectations built up by previous experience with other languages and language servers. Here, the evaluation should show whether NLS lives up to the expectations of its users.

Performance What are the typical latencies of standard tasks? In this context latency refers to the time it takes from issuing an LSP command to the reply of the server. The JSON-RPC protocol used by the LSP is synchronous, i.e. requires the server to return results of commands in the order it received them. Since most commands are sent implicitly, a quick processing is imperative to avoid commands queuing up.
Can single performance bottlenecks be identified? Single commands with excessive runtimes can slow down the entire communication resulting in bad user experience. Identified issues can guide the future work on the server.
How does the performance of NLS scale for bigger projects? With increasing project sizes the work required to process files increases as well. The evaluation should allow estimates of the sustained performance in real-world scenarios.

Answering the questions above, this chapter consists of two main sections. The first section sec. 1.2 introduces methods employed for the evaluation. In particular, it details the survey (sec. 1.2.2) which was conducted with the intent

to gain qualitative opinions by users, as well as the tracing mechanism (sec. 1.2.3) for factual quantitative insights. Section 1.4 summarises the results of these methods.

1.1 Evaluation Considerations

Different methods to evaluate the abovementioned metrics were considered. While quantifying user experience yields statistically sound insights about the studied subject, it fails to point out specific user needs. Therefore, this work employs a more subjective evaluation based on a standardized experience report focusing on individual features. Contrasting the expectations highlights well executed, immature or missing features. This allows more actionable planning of the future development to meet user expectations.

On the other hand it is more approachable to track runtime performance objectively through time measurements. In fact, runtime behavior was a central assumption underlying the server architecture. As discussed in sec. **??** NLS follows an eager, non-incremental processing model. While incremental implementations are more efficient, as they do not require entire files to be updated, they require explicit language support, i.e., an incremental parser and analysis. Implementing these functions exceeds the scope of this work. Choosing a non-incremental model on the other hand allowed to reuse entire modules of the Nickel language. The analysis itself can be implemented both in a lazy or eager fashion. Lazy analysis implies that the majority of information is resolved only upon request instead of ahead of time. That is, an LSP request is delayed by the analysis before a response is made. Some lazy models also support memoizing requests, avoiding to recompute previously requested values. However, eager approaches preprocess the file ahead of time and store the analysis results such that requests can be handled mostly through value lookups. To fit Nickels' type-checking model and considering that in a typical Nickel workflow, the analysis should still be reasonably efficient, the eager processing model was chosen over a lazy one.

1.2 Methods

1.2.1 Objectives

The qualitative evaluation was conducted with a strong focus on the first metric in [sec:metrics]. Usability proves hard to quantify, as it is tightly connected to subjective perception, expectations and tolerances. The structure of the survey is guided by two additional objectives, endorsing the separation of individual features. On one hand, the survey should inform the future development of NLS; which feature has to be improved, which bugs exist, what do users expect. This data is important for NLS both as an LSP implementation for Nickel (affecting the perceived maturity of Nickel) and a generic basis for other projects. On the other hand, since all features are essentially queries to the common linearization data structure (cf. [sec. **??**), the implementation of this central structure is an essential consideration.“‘ The survey should therefore also uncover apparent problems with this architecture. This entails the use of language abstractions (cf.

sec. **??**) and the integration of Nickel core functions such as the type checking procedure.

The quantitative study in contrast focuses on measurable performance. Similarly to the survey-based evaluation, the quantitative study should reveal insight for different features and tasks separately. The focus lies on uncovering potential spikes in latencies, and making empirical observations about the influence of Nickel file sizes.

1.2.2 Qualitative

Inspired by the work of Leimeister in (**leimeister?**), a survey aims to provide practical insights into the experience of future users. In order to get a clear picture of the users' needs and expectations independently of the experience, the survey consists of two parts – a pre-evaluation and final survey.

1.2.2.1 Pre-Evaluation

The pre-evaluation introduced participants in brief to the concept of language servers and asked them to write down their understanding of several LSP features. In total, six features were surveyed corresponding to the implementation as outlined in sec. **??**, namely:

1. Code completion Suggest identifiers, methods or values at the cursor position.
2. Hover information Present additional information about an item under the cursor, i.e., types, contracts and documentation.
3. Jump to definition Find and jump to the definition of a local variable or identifier.
4. Find references List all usages of a defined variable.
5. Workspace symbols List all variables in a workspace or document.
6. Diagnostics Analyze source code, i.e., parse and type check and notify the LSP Client if errors arise. The item for the "Hover" feature for instance reads as follows:

   > Editors can show some additional information about code under the cursor. The selection, kind, and formatting of that information is left to the Language Server.
   >
   > What kind of information do you expect to see when hovering code? Does the position or kind of element matter? If so, how?

Items first introduce a feature on a high level followed by asking the participant to describe their ideal implementation of the feature.

1.2.2.2 Experience Survey

For the final survey, interested participants at Tweag were invited to a workshop introducing Nickel. The workshop allowed participants unfamiliar with the Nickel language to use the language and experience NLS in a more natural setting. Following the workshop, participants filled in a second survey which focused on three main aspects:

First, the general experience of every individual feature. Without weighing their in expectations, the participants were asked to give a short statement of their experience. The item consists of a loose list of statements with the aim to achieve a rough quality classification:

☐ The feature did not work at all
☐ The feature behaved unexpectedly
☐ The feature did not work in all cases
☐ The feature worked without an issue
☐ Other

The following items survey the perceived performance and stability. The items were implemented as linear scales that span from "Very slow response" to "Very quick response" and "Never Crashed" to "Always Crashed" respectively. The second category asked participants to explicitly reflect on their expectations:

○ The feature did not work at all
○ The feature behaved unexpectedly
○ The feature did not work in all cases
○ The feature worked without an issue ○ Other

In the final part participants could elaborate on their answers.

Why were they (not) satisfied?
What is missing, what did they not expect?

1.2.3 Quantitative

To address the performance metrics introduced in sec. **??**, a quantitative study was conducted, that analyzes latencies in the LSP-Server-Client communication. The study complements the subjective reports collected through the survey (cf. sec. 1.2.2.2). The evaluation is possible due to the inclusion of a custom tracing module in NLS. The tracing module is used to create a report for every request, containing the processing time and a measure of the size of the analyzed document. If enabled, NLS records an incoming request with an identifier and time stamp. While processing the request, it adds additional data to the record, i.e., the type of request, the size of the linearization (cf. sec. **??**) or processed file and possible errors that occured during the process. Once the server replies to a request, it records the total response time and writes the entire record to an external file.

The tracing approach narrows the focus of the performance evaluation to the time spent by NLS. Consequently, the performance evaluation is independent of the LSP client (editor) that is used. Unlike differences in hardware which affects all operations similarly, LSP clients may implement different behaviors that may cause editor-specific biases. For instance, the LSP does not specify the frequency at which file changes are detected, which in turn can lead to request queuing depending on the editor used.

1.3 Process

1.4 Results

1.4.1 Qualitative

1.4.2 Quantitative