Abstract

In 1978, Niklaus Wirth surveyed the set of most commonly used algorithmic tasks in function of the accompanying data structures and vice versa. He published this survey in a book titled “Algorithms + Data Structures = Programs”. After more than three decades, it can still be considered to be an accurate title: Even today, software engineering practices focus on finding the “best” data structure for a program (i.e., the interplay of a set of algorithms). A single program often consists of multiple algorithms that operate on the same data. In such a program, the “best” data structure is the data structure that yields the best overall performance, where performance can be interpreted as shorter execution times, lower memory consumption, higher throughput, or any other desirable non-functional feature.

However, we observe that for some programs, relying on a single data representation can be less performant than changing the representation of the data at runtime. Hence, we argue in favour of using the “best” data representation for each phase of a program, and to enable changes between the different data representations at runtime. We call this just-in-time representation changes. Implementing representation changes is a tedious and error-prone task, for which little to no support exists in contemporary programming languages.

In this dissertation we explore the idea of just-in-time representation changes and taxonomize the design space of programming languages in which such representation changes can be implemented. From this taxonomy, we identify a vacuum in the landscape of programming languages, to allow developers to express data representations changes in order to realise non-functional features, such as for instance improving performance. Hence, the core contribution of this work is the design of a new programming language called JITds, which fills this vacuum. Besides representation changes themselves, another core design feature of JITds is that it is possible to disentangle general application logic from logic responsible for the representation changes. In short, with JITds we want to promote the shift from choosing a single data structure to choosing a set of data representations, in order to improve performance.