Graydon Hoare im Interview zur Programmiersprache Rust

Sprachen 12.07.2013 13:09 Uhr – Julia Schmidt – 59 Kommentare

Mozillas Rust steht in Version 0.7 zur Verfügung. Doch was ist der Sinn hinter einen weiteren Programmiersprache? Wann kann man mit ihrer Stabilität rechnen? Und wohin geht die Reise für das optisch an C erinnernde Konstrukt? Frank Müller hat für heise Developer mit Sprachentwickler Graydon Hoare gesprochen.

heise Developer: *Graydon, was war beziehungsweise ist deine Motivation hinter der Entwicklung von Rust?*

Graydon Hoare: Die Unzufriedenheit mit den Kompromissen, die man beim Schreiben in C++ eingehen muss. Wie viele andere Leute, die im Umfeld von System- und Desktop-Programmen arbeiten, musste ich aus Gründen der Performance, des Speicher- und Laufzeitmodells sowie entwicklungstechnischer Einschränkungen mit dieser Sprache arbeiten. Aber ich finde C++ ziemlich fehlerträchtig.

Es ist schwierig, damit Code zu schreiben, von dessen Fehlerfreiheit und Speichersicherheit ich überzeugt bin. Das gilt doppelt, wenn ich Dinge schreibe, die hochgradig nebenläufig sind oder mit vielen Threads arbeiten.

heise Developer: *Was sind die besonderen Stärken von Rust?*

Hoare: Meiner Meinung nach ist es die Speichersicherheit. Wenn du dich an die sichere Sprache hältst, also keine *unsafe { ... }* Blöcke, wird sich dein Programm auch speichersicher verhalten – auch bei Problemen, die sich aus dem parallelen und nebenläufigen Verschachteln ergeben. Das betrifft Dinge wie das Falschverstehen der Lebensdauer von Zeigern, Nullzeigern, Schreiben wilder Zeiger, Scheitern beim Freimachen des Speichers, Data Racing, etwas verändern, was eigentlich unveränderlich sein sollte, und so weiter.

Diese grundlegenden Sachen lassen sich mit "einfacheren" und mehr an der Brute-Force-Methode ausgerichteten Sprachtechniken wie ubiquitärem Verpacken und Garbage Collection erreichen, allerdings wollten wir die volle Bandbreite der C++-Idiome zur Nutzung des Speichers bei voller Geschwindigkeit und geringen Kosten unterstützen (RAII, Stack Allocation, Structure Nesting, interne Zeiger, Zero-Cost References). Statt also den einfachen Weg der GC-zentrischen Sprachen zu gehen, haben wir eine mächtige (und hoffentlich mundende) Kombination aus Speicherabstraktionen gewählt, die auf affinen und regionalen Typensystemen aufbaut. Diese sind in Sprachen aus der Forschung sehr verbreitet, in industriellen allerdings weniger. Aber sie sind großartig. Durch sie können wir die gewünschten Idiome und darüber hinaus viele weitere faszinierende Dinge

unterstützen, bei denen wir nicht sicher waren, ob wir sie umsetzen würden können:

- RAII-Objekte, die statisch nur auf Dinge mit höherer Lebensdauer verweisen können.
- Objekte, die statisch ihre Mutator-Methoden verlieren, wenn sie von einer veränderlichen zu einer unveränderlichen Variable verschoben werden.
- Iteratoren, die nicht während der Iteration ungültig gemacht werden können.
- eine geprüft sichere Weitergabe der Eigentumsrechte, Read-only- und Read-Write-Ordnungen zwischen Koroutinen und Threads.

Auch ist das System der Traits gelungen: Bei ihm verhält es sich ein bisschen wie mit der Arbeit mit Templates in C++, aber mit überprüften Schnittstelleneinschränkungen bei den Parametern der Typen. Dahinter steckt eine sehr bequeme Methode, um zwischen statischem und dynamischen Dispatching in Abhängigkeit vom Nutzungskontext zu wählen, statt das in der Definition zu tun.

Schließlich noch eine Anmerkung zur Ausdrucksmächtigkeit: Ich habe viel Zeit in Nebenprojekten und gelegentlich in bezahlter Arbeit damit verbracht, mit funktionalen Sprachen zu programmieren, und wir haben viele schöne Teile dieser Sprachen eingebunden: leichtgewichtige algebraische Datentypen mit Typinterferenz, Pattern Matching, Lambdas und leichtgewichtige Koroutinen sorgen für eine sehr ausdrucksstarke Erfahrung, sobald man sich an den Stil gewöhnt hat.

heise Developer: *Für welche Einsatzszenarien ist Rust gedacht?*

Hoare: Für alles, was die Leute heutzutage mit C++ schreiben. Server- und Desktop-Programme. Seltsamerweise haben wir herausgefunden, dass sich viele Spieleentwickler dafür interessieren, was aber letztlich Sinn ergibt. Sie waren eigentlich nicht unsere ursprüngliche Zielgruppe, aber viele von ihnen sind ja C++-Nutzer.

heise Developer: *Welche Art von Projekten gibt es momentan und wie groß sind sie?*

Hoare: Der Rust-Compiler ist selbst in Rust geschrieben, allerdings hängt sein Stil etwas hinter der Entwicklung der Sprache her, da er ein eigenständiges Projekt ist: Der Compiler ist in einem älteren Dialekt geschrieben, und viele Idiome müssen noch entfernt werden. Das Projekt umfasst etwa 100.000 Codezeilen, plus etwa 100.000 Zeilen in den Standardbibliotheken.

Die wichtigste Zielanwendung ist eine Browser Engine namens Servo, deren Entwicklung der Grund für die Mozilla Foundation ist, in Rust zu investieren. Sie umfasst etwa 35.000 Zeilen. Servo ist noch ein junges Projekt.

Auf unserer Website sind noch einige andere Projekte gelistet, an der die breitere Community arbeitet: einige Spiele, ein paar Demos, verschiedene Bindings und Bibliotheken. Noch nichts wahnsinnig Großes. Die Sprache befindet sich immer noch im Alpha-Stadium. Wir hoffen, dass die Größe der Projekte wächst, wenn sich die Sprache und die Bibliotheken stabilisieren.

Vorherige 1 2 Nächste

Kommentare lesen (59 Beiträge)

Forum zum Thema:
C/C++

Themenseiten:

MOZILLA

https://heise.de/-1916345 Drucken