Betrachtung der Einflüsse von Zwischendarstellungseigenschaften auf die Evolution von Werkzeugketten

Timm Felden Felix Krause

Universität Stuttgart, Institut für Softwaretechnologie
Universitätsstr. 38, 70569 Stuttgart
{feldentm,krausefx}@informatik.uni-stuttgart.de

Zusammenfassung

In diesem Papier werden Eigenschaften von Zwischendarstellungen (IRs) auf Evolution und Migration von Werkzeugketten betrachtet. Dabei steht eine rückblickende Betrachtung SKilL-basierter Projekte im Mittelpunkt. Ebenso wird versucht, die Entwurfsentscheidungen hinter SKilL neu zu bewerten.

Einleitung

SKilL [5] ist ein Serialisierungssystem, das dem Zweck dient, große Objektgraphen sprachunabhängig, typsicher und änderungstolerant zwischen Werkzeugen auszutauschen, welche nicht unbedingt dieselbe Spezifikation der enthaltenen Objekte teilen. Motiviert wurde SKilL durch die Erfahrungen mit der Anbindung von Bauhaus [9], einer überwiegend in Ada geschriebenen Programmanalysewerkzeugkette, an Eclipse, welches in Java geschrieben ist. [6] Hier musste ein API verwendet werden, das über zwei Sprachgrenzen Zugriffe auf die IR bot und sowohl untragbar inperformant als auch unvollständig war.

Mittlerweile sind SKilL-Implementierungen für einige Sprachen frei verfügbar [2] und es gibt eine wissenschaftliche Betrachtung des Gesamtsystems [4]. Die Migration der Bauhaus-IR hat 2016 begonnen [8] und ist im Wesentlichen abgeschlossen. Daneben wurde ein knappes Mannjahr in ein noch unveröffentlichtes Programmiersprachenforschungsprojekt (Type Research language, Tyr) investiert, welches mit einer SKilL-basierten IR ein in Scala geschriebenes Front-End mit sich selbst und einem in C++ geschriebenen LLVM-Back-End verbindet. Zudem wurde eine SKilL-basierte LLVM/IR mit LLVM-Bitcode verglichen [7].

In den nachfolgenden Betrachtungen ist die niedrige Kopplung der Komponenten in Werkzeugketten zu beachten. Dadurch ist die Entwicklung in getrennten Programmen wirtschaftlich. Für die Migration von Bauhaus war entscheidend, dass die IR in eine eigene Bibliothek ausgelagert war, deren Implementierung überwiegend aus einer Spezifikation generiert wird.

Sprachunabhängigkeit

Die ursprüngliche Argumentationslinie für SKilL forderte gleichermaßen Sprach- und Plattformunabhängigkeit der Lösung. Tatsächlich hat sich Letztere aus dem sprachunabhängigen Design ergeben, was die gesonderte Forderung zwecklos macht. Daneben wäre für alle von uns durchgeführten Projekte POSIX-Kompatibilität [1] ausreichend gewesen, wenn man von Projekten absieht, die die Funktionsfähigkeit auf anderen Plattformen demonstrieren sollten.

Wie erwartet war es hilfreich, den Studenten bei der Bearbeitung von Abschlussarbeiten die Sprachwahl offen zu lassen. Das motivierende Element für die Studenten war dabei unterschiedlich. Bei Studenten, deren Motivation im Lernen einer neuen Sprache besteht, ist es empfehlenswert, zu Projektbeginn auf die Verwendung einer bekannten Sprache zu drängen. Da das Projekt dann in der Regel schon begonnen hat, bleibt der motivierende Faktor selbst dann bestehen, wenn am Ende Java oder C++ eingesetzt wird.

Innerhalb einer Werkzeugkette mehrere Sprachen einzusetzen, erwies sich als unproblematisch. Bezogen auf ein Werkzeug gilt dies jedoch nicht. Das liegt überwiegend an der schlechteren Werkzeugunterstützung. So funktionieren Entwicklungsumgebungen, Debugger und die Erhebung von Testabdeckung überwiegend nur für eine Sprache oder Sprachfamilie.

Insbesondere wenn die Werkzeugkette über Jahrzehnte entwickelt wird, ist es wünschenswert, jederzeit auf aktuelle Technologie zurückgreifen zu können. So wurde beispielsweise das in C++ implementierte C99-Front-End durch ein neueres ersetzt, welches C11-kompatiblen Code verarbeitet. Dabei wurde der C++-Ada-Wrapper komplett durch ein generiertes SKilL/C++-API ersetzt. Zudem konnte die Anbindung des Werkzeugs an die IR auf das relevante Subset reduziert werden, da die Implementierung der IR nicht mehr auf der von allen Werkzeugen gemeinsam genutzten Bibliothek basiert. Ebenso konnte das Front-End komplett aus dem per Werkzeug sequentiellen Bauhaus-Buildprozess entfernt werden, was nicht nur parallele Builds erlaubt, sondern auch ein rechtliches Problem mit einer genutzten Bibliothek löst. Zudem reduziert sich die Größe des Binaries samt zugelinkter Bibliotheken um über 100MB, da das Werkzeug nur den Zugriff auf einen Teil der IR, nicht aber die zahlreichen Hilfsfunktionen benötigt. Das Verhalten unverändert bestehender Werkzeuge wurde ver-

wendet, um die Korrektheit der Umstellung abzusichern. Bei diesem Vorgehen handelte es sich dennoch nicht um eine Eins-zu-eins-Migration, da während der Migration Fehler der Bestandslösung nur im neuen Front-End korrigiert wurden.

Änderungstoleranz

Reduziert man die Werkzeugspezifikation eines bestehenden Werkzeugs auf das tatsächlich verwendete Subset, so stellt man in Sprachen mit positionaler Parameterübergabe fest, dass man bei Konstruktoraufrufen Standardwerte streichen muss, die zu nie verwendeten Feldern gehören. Abhilfe lässt sich mit einem Erbauer pro spezifiziertem Typ schaffen.

Für SKilL wurde ein Werkzeug entwickelt, das Bestandsdatensätze auf eine neue Spezifikation abbilden kann [3]. Dieses erzeugt einen zur neuen Spezifikation passenden Datensatz, dessen Werte dem alten Datensatz entnommen werden. Dadurch lassen sich auch Anpassungen vornehmen, die nicht automatisch vorgenommen werden können, wie etwa das Umbenennen von Feldern oder Veränderungen innerhalb der bestehenden Typhierarchie.

Das Werkzeug wurde im Rahmen der Bauhaus-Migration noch nicht eingesetzt, da die Originaldaten noch vorhanden sind und das Migrationswerkzeug in das neue Format nach wie vor funktionsfähig ist. Ferner würde eine Neugenerierung der größeren Datensätze mehrere Tage pro Datensatz beanspruchen.

Eine überraschende Beobachtung sind die Effekte von Änderungstoleranz auf die Entwicklung von Tyr. Da sich das Projekt in einem Prototypstadium befindet, verändert sich die IR häufig. Die Änderungen sind aber auch hier überwiegend so, dass man das Back-End auch mit der IR eines weiterentwickelten Front-Ends verwenden konnte. Dadurch konnten Weiterentwicklungen schneller getestet werden. Ebenso konnten Bootstrapping-Probleme überwiegend gelöst werden, indem man eine alte vorübersetzte Version der Standardbibliothek zum Testen verwendete, wodurch komplizierte Sonderfälle von Spracherweiterungen in der Standardbibliothek nicht unmittelbar behandelt werden mussten.

Spezifikationssprache

Der Entwurf der SKilL-Spezifikationssprache orientiert sich in puncto Funktionsumfang an einfach umsetzbaren Eigenschaften vergleichbarer Produkte. Bezüglich Container haben sich dynamische Arrays, Sets und Maps als unbedingt erforderlich erwiesen. Ob man sowohl Listen als auch Arrays braucht, ist unklar. Der Mehraufwand hält sich jedoch in Grenzen.

Die in SKilL vorhandenen Konstanten haben sich als nutzlos erwiesen. Dagegen sind Klassen und polymorphe Zeiger wie erwartet hilfreich. Das verwendete Interface-Konzept mit Vererbung von Klassen hat sich als hilfreich erwiesen. Es lässt sich auf eine reine Vererbung durch Klassen projizieren, was effiziente APIs in Ada und C++ ermöglicht. Die Bauhaus-Spezifikation ist überwiegend klassenorientiert, was teilweise zur Duplizierung von Felddefinitionen führt. Dagegen wurde die Tyr-Spezifikation direkt mit Mehrfachvererbung entworfen. Dadurch konnten gemeinsame Eigenschaften gruppiert werden, ohne den generierten Code ineffizienter zu machen.

Unerwartet hilfreich ist das `view`-Konzept. Dieses erlaubt es, die Sicht auf ein Feld eines Supertyps neu zu definieren. Dadurch kann man Beziehungen in Cohierarchien korrekt darstellen. So hat etwa eine Definition einen Typ, aber eine Funktionsdefinition einen Funktionstyp. In Sprachen, die covariante Getter erlauben, können diese im API generiert werden. Das verbessert die Lesbarkeit der Werkzeugimplementierung deutlich, da unnötige manuelle Typprüfungen und Konvertierungen hinter dem API verschwinden.

Nicht serialisierte Felder in der Werkzeugspezifikation wurden projektunabhängig bei den meisten neuentwickelten Werkzeugen genutzt. Dabei kamen gleichermaßen spezifikationsinterne (`auto`) wie -externe Feldtypen (`custom`) zum Einsatz. Der neu entwickelte Bauhaus-Linker verwendet beispielsweise ein solches Feld anstelle einer globalen Unifikationstabelle.

Fazit

Die motivierenden Probleme der Bauhaus-IR konnten durch Migration auf SKilL tatsächlich gelöst werden. Insgesamt hat SKilL einen deutlich positiven Einfluss auf die Evolution von damit realisierten IRs. Unter Berücksichtigung obiger Beobachtungen wird derzeit ein Nachfolger unter dem Namen *Object Graph Serialization Sytem* (OGSS) entwickelt.

Literatur

[1] Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. In: *IEEE Std 1003.1, 2016 Edition*.

[2] Timm Felden et al. SKilL on Github. `https://github.com/skill-lang/skill`, 2016. (Abgerufen 19.2.2019).

[3] Oliver Brösamle. Manipulation der Typisierung serialisierter SKilL-Graphen. Masterarbeit: Universität Stuttgart, Institut für Softwaretechnologie, 2018.

[4] Timm Felden. *Änderungstolerante Serialisierung großer Datensätze für mehrsprachige Programmanalysen*. Dissertation, Universität Stuttgart, Germany, 2017.

[5] Timm Felden. The SKilL Language V1.0. Technischer Bericht Informatik 2017/01, Universität Stuttgart, Institut für Softwaretechnologie, 2017.

[6] Timm Felden and Torsten Görg. Werkzeugunterstützte Eliminierung von Data-Races in Eclipse. In: *Softwaretechnik-Trends*, volume 33:2, 2013.

[7] Daniel Pfister. Skilled LLVM. Masterarbeit: Universität Stuttgart, Institut für Softwaretechnologie, Programmiersprachen und Übersetzerbau, 2018.

[8] Dennis Przytarski. SKilLed Bauhaus. Masterarbeit: Universität Stuttgart, Institut für Softwaretechnologie, 2016.

[9] Aoun Raza, Gunther Vogel and Erhard Plödereder. Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering. In: *Reliable Software Technologies – Ada-Europe 2006*, *Lecture Notes in Computer Science*, volume 4006, (pages 71–82). Springer Berlin Heidelberg, 2006.