Parser bauen für domänenspezifische Notationen

Arnold, Eckhart
arnold@badw.de
Bayerische Akademie der Wissenschaftten, Germany

Programmierworkshop: 4h
Benötigte Vorkenntnisse: Python und reguläre Ausdrücke

Domänenspezifische Notationen (DSL) sind auf ein jeweils bestimmtes Anwendungsfeld zugeschnitten und ermöglichen deshalb oft eine schnellere Dateneingabe und übersichtlichere Quelltexte als generalisierte Auszeichnungssprachen. Sie bilden deshalb in vielen Bereichen eine sinnvolle Ergänzung oder sogar Alternative zu XML. Beispiele dafür liefern: (Tinney 2014)(von Stockhausen 2020)(Arnold 2019).

Trotz dieser Beispiele werden DSLs in den Digital Humianities (DH) noch eher selten eingesetzt. Dies kann damit zusammenhängen, dass die Kenntnis der Technologien zum Bau von DSL in diesem Bereich noch wenig verbreitet ist. Ein Ziel des Workshops ist es genau diese Kenntnisse zu vermitteln. Mögliche Einsatzgebiete von DSLs gibt es in vielen Bereichen:

1. Bei der Eingabe bzw. Kodierung von Daten, wo DSLs sich als eine bequemere und übersichtlichere Alternative zu XML anbieten, insbesondere in Fällen, wo XML-Editoren die Eingabe nur begrenzt einfacher machen oder man nicht auf proprietäre Produkte wie Oxygen zurückgreifen möchte.
2. Bei der Daten-Extraktion. Da sich mit EBNF nicht nur reguläre sondern auch kontext-freie Grammatiken spezifizeren lassen, ist mit diesem Ansatz mehr möglich als mit regulären Ausdrücken. Dazu gehört insbesondere auch die Konvertierung von Alt-Daten, die in einem textbasierten Format kodiert sind, wie z.B. LaTeX-Manuskripte, nach XML, um sie für die Weiterverarbeitung mit dem Computer vorzubereiten.
3. Schließlich gibt es noch technischere Einsatzgebiete, die für die DH indirekt relevant werden können, wie z.B. die Übertragung von Datenstrukturen aus einer Programmiersprachenwelt in eine andere, etwa TypeScript-Interfaces zu Python TypedDicts: https://ts2python.readthedocs.io

Allerdings erfordert der Einsatz von DSLs die Programmierung von sog. „Parsern“, die Texte in der DSL in generische Datenbeschreibungssprachen wie XML oder SQL oder direkt in bestimmte Datenstrukturen übersetzen. Während es für kleinere Projekte noch genügt, die Grammatik einer DSL informell zu beschreiben und den Parser aus regulären Ausdrücken zusammenzusetzen, lassen sich größere Projekte oder DSLs, die sich mit der Zeit weiterentwickeln, ohne eine formale Spezifikation der Grammatik einer DSL und automatisierte Tests, die vor Fehlern bei späteren Ergänzungen der DSL schützen, kaum noch realisieren. Üblicherweise werden dafür Parser-Generatoren verwendet, die die Strukturdefinition bzw. Grammatik der DSL in einen ausführbaren Parser übersetzen. Die Grammatik wird dabei mit der Beschreibungsprache EBNF spezifiziert. (Eine EBNF-Spezifikation ist für eine DSL ungefähr das, was eine DTD für XML darstellt.)

Natürlich gibt es auch andere Möglichkeiten, Parser zu bauen, z.B. handgeschriebene Adhoc-Parser oder gestaffelte reguläre Ausdrücke, wie sie etwa von „TextMate-Grammatiken“ für die farbliche syntaktische Hervorhebung von Text-Editoren verwendet werden. Für den Einsatz von Parser-Generatoren und EBNF sprechen diese Gründe:

1. Der EBNF-Formalismus ist seit einigen Jahrzehnten ein stabiler und vielfach genutzter Standard für die Spezifikation formaler Sprachen. Spezifiziert man die Grammatik einer DSL in EBNF, so kann man im Zweifelsfall mit relativ geringem Aufwand auf einen anderen Parser-Generator und eine andere Programmiersprache umziehen. Handgeschriebene Parser sind relativ stärker mit der einmal gewählten Technologie vermählt. Das allein ist ein guter Grund für den Einsatz von EBNF.
2. EBNF erlaubt eine absolut präzise und zugleich konzise Spezifikation der Grammatik einer formalen Sprache. Als Ergänzung zu eine Prosa-Beschreibung mit Beispielen, kann man damit die Regeln einer domänenspezifischen Notation missverständnisfrei kommunizieren.
3. Ab einer gewissen Komplexität erscheint mir der Einsatz eines Parser-Generators, den man mit der formalen Spezifikation (in EBNF) der eigenen DSL füttern kann, bequemer als andere Ansätze, wie etwa gestaffelte reguläre Ausdrücke oder vollständig handgeschriebene Parser. Auch, wenn eine DSL im Laufe der Zeit abgeändert bzw. ergänzt werden soll, zahlt sich eine explizit spezifizierte Grammatik (im Verein mit einer umfassenden Test-Suite) aus - so zumindest meine Erfahrung.

Dieser Workshop ist als Lehrveranstaltung auf 4 Stunden angelegt und vermittelt einen Einstieg in den Bau von Parsern für DSLs. Vermittelt werden:

1. Die formale Spezifikation von Grammatiken für DSLs in der Erweiterten Backus-Naur-Form (EBNF): EBNF ist so etwas wie Reguläre Ausdrücke auf Speed. Während reguläre Ausdrücke wie der Name sagt, nur die sog. „Regulären Sprachen“ verarbeiten können, können mit EBNF Grammatiken für vergleichsweise sehr viel ausdrucksreichere „kontextfreie Sprachen“ festgelegt werden. Insbesondere wird es dadurch sehr viel leichter, verschachtelte Strukturen zuzulassen.
2. Der Bau eines auf dieser Grammatik beruhenden Parsers für die DSL mit Hilfe eines Parser-Generators. Parser übersetzen DSL-Texte in „konkrete Syntaxbäume“. Konkrete Syntaxbäume enthalten in der Regel jedoch noch viele Spuren des Übersetzungsprozesses, die für die weitere Verarbeitung nicht mehr relevant sind. Die Kunst besteht darin, diese konkreten Syntaxbäume zu möglichst schlanken „abstrakten Syntaxbäumen“ zu vereinfachen. Während der Parser selbst automatisch generiert werden kann, hängt die Generierung des abstrakten Syntaxbaums von der Zieldomäne ab und muss daher von Hand festgelegt werden.
3. Die Extraktion von Daten aus Syntaxbäumen bzw. die Umformung von Syntaxbäumen in vorgegebene Zieldatenstrukturen. Auch die abstrakten Syntaxbäume entsprechen in der Regel noch nicht den Zieldatenstrukturen, sondern müssen noch einmal umgewandelt werden. Sollen die Zieldaten in XML vorliegen, so genügt eine einfache Serialisierung. Komplizierter wird es, wenn die Zieldaten gar keine Baumstrukturen haben, sondern z.B. Graphen sind.
4. Der Einsatz von Einheiten-Tests zum schrittweisen Aufbau und der kontinuierlichen Prüfung von Grammatiken. Insbesondere für spätere Änderungen und Ergänzungen einer DSLs sind Einheiten-Tests nahezu unverzichtbar, will man

die Rückwärtskompatibilität der sich weiterentwickelnden DSL sicher stellen.

Als Parser-Generator verwenden wir das Python-Rahmenwerk DHParser. Wer möchte kann sich dort im Vorfeld die „Schritt für Schritt“-Anleitung dazu durchlesen: https://t1p.de/0l6l.

Wohlbemerkt: In dem Kurs geht es um den Bau von Parsern mit Hilfe eines Parser-Generators, was etwas, aber nicht viel schwieriger ist als das Erlernen der Technologie der Regulären Ausdrücke. Es geht also um die Nutzung und nicht um die Entwicklung von Parser-Generatoren (auch wenn einige der Einträge in der Bibliographie sich damit beschäftigen), was ein sehr viel komplizierteres Thema ist. Wen das interessiert, dem lege ich als Einstieg die Blog-Serie von Guido van Rossum dazu ans Herz: t1p.de/y2ko

Bibliographie

Arnold, Eckhart (2021): DHParser. Toolchain for Domain Specific Languages in the Digital Humanities, gitlab.lrz.de/badw-it/DHParser.

Arnold, Eckhart (2019): Dokumentation der Notation für Artikel des Mittellateinischen Wörterbuchs, t1p.de/44x9.

Blaudeau, Clement / Shankar, Natarajan (2020): A verified packrat parser interpreter for parsing expression grammars, CPP 2020: Proceedings of the 9th ACM SIGPLANInternational Conference on Certified Programs and ProofsJanuary 2020 Pages 3–17, DOI: 10.1145/3372885.3373836, t1p.de/otay.

Ford, Brian (2004): Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, Cambridge Massachusetts.

Mascarenhas, Fabio / Medeiros, Sérgio / Ierusalimschy, Roberto (2014): On the relation between context-free grammars and parsing expression grammars, Science of Computer Programming, Volume 89, Part C, Pages 235-250, t1p.de/v5uz.

v an Rossum, Guido (2019): PEG Parsing, t1p.de/y2ko.

von Stockhausen, Annette (2020): Domain Specific Language (DSL) für Transkriptionen, t1p.de/r8yv.

Tinney, Steve et al. (2014): ORACC. Open Richly Annotated Cuneiform Corpus, t1p.de/uryr.

Voelter, Markus et al. (2013): DSL-Engineering. Designing, Implementing and Using Domain-Specific Languages, Stuttgart, t1p.de/wik.