Agile Construction of Data Science

DSLs (Tool Demo)

Artur Andrzejak, Kevin Kiefer, Diego Costa, Oliver Wenz

Heidelberg University
http://pvs.ifi.uni-heidelberg.de/
DOMAIN SPECIFIC LANGUAGES:
Blessings and Curses
Disadvantages of (External) DSLs

A. Effort of DSL implementation
B. DSL extension needed

Using only classical programming languages (GPL like Python)
Ideal DSL Engineering

Solution Postulates:

• A. Low initial “price” for implementing a DSL
• B. No extra overhead for tasks outside DSLs
• C. Integration with existing code base
• D. Support for common IDEs / editors
TOOL NLDNL: KEY CONCEPTS
Our tool NLDSL consists of:

- A library *.lib for implementing pipeline-oriented DSLs
- An environment *.edit for DSL editing and in-editor code generation for IDEs supporting LSP
D. Support for common IDEs / editors

- Our tool uses a **Language Server Protocol (LSP)**
  - Separates editor “frontend” from services (completion, lint, ...)

---

**Theia, Eclipse Che, ...**

**Visual Studio Code**

**IntelliJ IDEA / PyCharm**

**Generic Code Recommender**

**User-defined DSLs**

**Library for DSL dev.**

**textX**

**pyglS**

**DSL-based Lang. Server**

**NLDSL.edit**

**NLDSL.lib**

---
Development Phases with External DSLs

External DSL

DSL editing: isolated from GPL

Host Program (GPL)

Target GPL or IR

Intermediate form, not editable by user

Executable
B: No Overhead for Tasks Outside DSLs

- GPL code is generated on-demand during editing
- User enters DSL code in comments, assisted by editor
C. Integration with existing code base

- Final code is pure GPL (Python) code
- DSL in comments only, no dependencies on tool/DSL
A. Low Effort for Implementing a DSL

- We created a framework for fast creation of families of DSLs (not only for data science)
- DSL families: syntax based on fluent API

Two ways of implementing/extending DSL

- 1. Fast creation/updates of core DSL operations
- 2. Instant extensions during editing by DSL users
A. Low Effort for Implementing a DSL

• We focus on family of constrained DSLs which model chains or pipelines of operations (similar to “fluent API” syntax)
• A compact implementation as a set of Python functions which bundle together DSL syntax description and code generation.
• We allow defining DSL-level functions as ad-hoc DSL extensions
Fluent API

• Fluent API:
  – Object oriented APIs based on method chaining
  – Popular due to higher readability of the source code

• Supported DSL syntax:
  – Object | Op1 | Op2 | Op3 | ...

• Maps to GPLs/libs in data science, e.g. Pandas
  ```python
  wine.rename(columns={"color_intensity": "ci"})
  .assign(color_filter=lambda x: np.where((x.h > 1), 1, 0))
  .query("alcohol > 14 and color_filter == 1")
  ```
Implementing DSL Operations

• Each DSL element implemented as a Python method
  – Python docs: specify DSL syntax + parameters
  – Python code: generating target code

```python
gb_doc = """Grammar:
    group by $columns[$col] apply $aggregation
    aggregation := { min, max, sum, avg, mean, count }
Type:
    Operation
"""
@grammar(gb_doc)
def group_by(code, args):
    cols = list_to_string(args["columns"])
    return code + ".groupby({}).{()}.format(cols,
        args["aggregation"])
...
PandasCodeGenGenerator.register_function(group_by)
```
Instant DSL Extensions

• Example of a new DSL command:
  – Divide a column by another column in the dataframe

• Rule definition:

  #$ \texttt{div columns} \ $col1 \ $col2 \ \texttt{as} \ \$res = \texttt{append column} \ $col1 / \ $col2 \ \texttt{as} \ \$res$

• Rule usage (same file)

  ## on df | \texttt{div columns} \ df.totalRooms \ df.households \ \texttt{as} \ 'roomsPerHousehold'

  df.assign(**{'roomsPerHousehold': df.apply(lambda row: row.totalRooms / row.households, axis=1).values})
Proof of Concept: Data Science DSLs

One DSL, multiple targets
• DSL for dataframe operations (~ SQL)
• Targets: Pandas, Spark

Other DSLs:
• Visualization: seaborn
• Deep learning: Keras, TF
TOOL NLDSL: DEMO
Examples (PySpark) and Demo

```python
## start_session named "Example"
spark = SparkSession.builder("Example").getOrCreate()

## df = load from '../data/cal_housing.data' as csv
df = spark.read.csv('../data/cal_housing.data')

## largeDelay = df | select_cols carrier, flight, arr_delay | select_rows "arr_delay" > 20
largeDelay = df.select(carrier, flight, arr_delay).filter(col("arr_delay") > 20)

## on largeDelay | show
largeDelay.show()
```
Tool Availability

• Online editor (Theia) at http://129.206.61.41:3000/


• NLDSL.lib for DSL implementations:
  – GitLab: https://gitlab.com/Einhornstyle/nldsl
  – Docs: https://einhornstyle.gitlab.io/nldsl/
  – PyPI package: https://pypi.org/project/nldsl/
    • pip install nldsl

• Coming soon: Visual Studio Code extension
Thank you.

QUESTIONS ARE WELCOME!

Contact:

Email: artur@uni-hd.de
WWW: pvs.ifi.uni-heidelberg.de
Web search: “PVS Heidelberg”