Literature Review + Outline - jcom: A JavaScript Object Shell

DeDominic, Anthony
Eastern Connecticut State University
Willimantic, USA
dedominica@my.easternct.edu

Abstract

This is merely the background on my subject, tying together research garnered over the weeks. Below I will discuss the importance of a shell with modern data structures and object like constructs I will also discuss problems that are being solved with such shells. I will also talk about processes that will be used in my project.

1. Introduction

Currently popular shells have no built-in feature complete way of transacting or using structured documents. As a result, working with web oriented services becomes a hassle; A hassle that involves very long and contrived text pipelines full of tools like: sed, grep, awk, xargs, tr, paste, cut, tee, and so on. What results is some sort of string tokenized parser. An example of the problem can be seen below[1].

```
# gets an html webpage from duckduckgo,
# processes it into
# "url - page description" lines
# removes ad links
# and only preserves the first three
curl ${SEARCH_ENGINE}$(rawurlencode ${4}) | \
html2 | \
grep -A 2 "@class=result__a" | \
sed '/^--$/d' | \
sed '/@class/d' | \
grep -Po '(?<=\/a(=|\/)).*' | \
paste -d " " - - | \
sed 's/\(@href\|b\)=//g' | \
sed '/r\.search\.yahoo\.com/d' | \
head -n 3
```

1.1. Related Works

Object Oriented shells isn't an entirely new concept. As others have shown, many projects, some more serious than others, attempted to solve this issue [1] [2].

The power of shells is well understood. Most programming languages have some kind of system() or shell invocation mechanism. Some, such as the developers of shcaml attempted to take it further by including functional combinators; these combinators allowed for slipping in native ocaml code and objects. As a result these code pieces could be parsers, that could take known shell commands, and structure their data into key-value trees [1].

[1] source: https://github.com/GeneralUnRest/neo8ball-irc/blob/master/lib/search.sh

Purely object oriented attempts ultimately result in new instances of bash and explicit message passing with named pipes [2]. Even though offers powerful object-like abstractions it general comes with significant overhead. It also incurs security risks since processes can write and listen to the object named pipes.

Lisp shells have been ideas attempted in the past; examples are like Scheme Shell, and other toy, academic ones [1][3]. The advantage lisp gives over other languages, which makes it suitable for shells, is that the program and data have the same form [3]. This comes in handy in meta programming, where programs can dynamically change to handle certain data. In terms of structural data, strong functional composition and combination can give structural data parsing functionality; of course at great cost of readability and complexity.

Projects like Powershell are built on the .NET runtime and can thus utilize class like features and functions. The streaming pipelines in Powershell are merely .NET classes which allow for some structural correctness and parsing [1][4]. As a result, Microsoft prefers to call more than just a shell, but a whole "automation and configuration management framework." One that is capable of handling structured data such as JSON, XML, CSV, etc, REST APIs, and object models.

Tools like Ansible, aren't quite shells, but are domain specific languages which solve similar problems. Ansible, et al, use code generation and a yaml playbook file to construct python code; since python is a general purpose language, it handles structured and typed data much better than shell code. As a result, powerful modules for system administration, configuration and interaction with web services have made it a formidable automation tool.

1.1. Service-oriented Architecture

Service-oriented Architectures, or SOA, are modern way to construct, highly available, data rich web applications. In short, a service oriented architecture is a way for applications to share state and provide services over middleware and structured, serialized objects. [5] One way of transacting states between services is through Representational state transfer, or REST. With REST, services are able to share well structured data through document structures like JSON, XML, etc. [6]

POSIX-shells are completely out of this domain because of their limitations. Generally when dealing with web APIs it is recommended to build a general purpose language script instead.

1.2. Designing and Defining a Domain Languages

Designing domain specific languages is a arduous problem. Some have suggested designing and developing domain specific languages using meta-models; basically a model, modeling models [7]. This process makes use of advanced features of UML version 2.1. This also has the added benefit of code generation.

Defining a language is ultimately the process of creating formal grammars. Generally this is done empirically, as in from experience and trial and error. Some have used machine learning and bottom-up CYK based parsers to generate grammar [8]. This would be useful for languages built using example strings.

1.3. Events

Some system events may impact other systems. Currently, other than using "triggers", there is no way to define listen events in POSIX shells. Powershell offers such functionality through the power of its .NET classes [4].

Currently to accomplish a similar function, one must use a blocking statement like this:

```
# file descriptor is needed
# for performance reasons
exec <>99 /some/named-pipe
# consumes a process, must be subshelled.
while read -r line; do echo $line; done <&99 &
echo "event" >&99
```

Given that package management tools used in many linux distributions use shell like scripting, and have a post-install process, it may be helpful for a user to define his own post-install event handlers or notifiers.

2. Background

- about the project, overall.
- how it compares to prior work.
- overview of methodologies.

2.1. Materials

- list of technologies.
- libraries.
- machine used in benchmarking/experiment.

3. Objects and Structured Documents (part of methods)

- how the key language feature works.
- implementation.
- examples of it being used.
  - reaffirm key importance.

4. Events (part of methods)

- feature showing off *emit* and *on* keyword.
- describe how it works, nodejs EventEmitter, etc.
- use case examples.
  - show use cases.

5. Command and File Parsers (part of methods?)

- section to show off feature.
- describe how it allows one to create parser modules for commands and common files.
- examples of uses.
  - ideally just a few commands, like ls, etc.
  - show how it can be used to easily configure some program.
  - nagios, apache, something.

6. Further Examples

- compare and contrast to various other tools and shells.
- benchmarking vs other tools/technologies.

7. Results | Findings

- final word, compare to other shells and systems manipulation technologies.
- present benchmarking results.

8. Discussion

- implications of the project.
- state limitations, weaknesses in product.
- further work needed.

Citations

[1] A. Heller and J. Tov, "Caml-Shcaml: An OCaml Library for UNIX Shell Programming," 2008. DOI: 10.1145/1411304.1411316

[2] J. Haemer, "A New Object-Oriented Programming Language: sh," 1994 [Online]. Available: http://https://www.usenix.org/legacy/publications/library/proceedings/bos94/full_papers/haemer.ps

[3] J. Ellis, "A LISP SHELL," 1980. DOI: 10.1145/947639.947642

[4] J. Snover, "Monad Manifesto." Microsoft, Aug-2002 [Online]. Available: http://www.jsnover.com/Docs/MonadManifesto.pdf

[5] Perrey and Lycett, "Service-oriented architecture." IEEE, Jan-2003. DOI: 10.1109/SAINTW.2003.1210138

[6] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with representational state transfer (REST)." Elsevier, 2008. DOI: 10.1016/j.websem.2007.11.002

[7] B. Selic, "A Systematic Approach to Domain-Specific Language Design Using UML." IEEE, May-2007. DOI: 10.1109/ISORC.2007.10

[8] K. Nakamura and T. Ishiwata, "Synthesizing Context Free Grammars from Sample Strings Based on Inductive CYK Algorithm." Springer, Grammatical Inference: Algorithms and Applications, 2000. DOI: 10.1007/978-3-540-45257-7_15