Creating multiplatform and mobile apps in Scheme with Gambit/SchemeSpheres

[Tutorial abstract]

Álvaro Castro-Castilla
Fourthbit
Madrid, Spain
alvaro.castro.castilla@fourthbit.com

ABSTRACT

Mlkshk 90's sustainable kale chips tousled. Swag mumblecore blog Banksy, try-hard wayfarers Helvetica tousled art party fashion axe plaid Truffaut Kickstarter Neutra fixie. Semiotics bespoke tofu butcher. Yr cliche crucifix, put a bird on it Neutra Portland authentic kale chips leggings Marfa dreamcatcher. Narwhal bicycle rights Bushwick cred. YOLO literally put a bird on it pour-over. Polaroid whatever paleo food truck.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.m [**Software Engineering**]: Miscellaneous—*rapid prototyping, reusable software*

General Terms

Design, Documentation, Experimentation, Languages

Keywords

Scheme Programming Language, multiplatform, mobile, Gambit, SchemeSpheres, automation, software construction, framework

1. INTRODUCTION

1.1 Motivation

Implementation fragmentation and diversity has been a long-standing part of the Scheme community. *Scheme Now!*, *Eggs*, *PLaneT* are the most prominent proposals aiming at defining a package specification for shared code, as well as centralizing packages in common repository. Specially the case of *Scheme Now!*, which has been developed with the intention of serving as a cross-implementation system, might be seen as an approach to the fragmentation problem. Most other package systems aim at solving the distribution and sharing problem for a specific implementation and Scheme ecosystem.

With the advent of mobile technologies, complexity grows as the problem becomes not only of distribution, but of software construction workflows. Building, debugging and deploying code for mobile platforms vary wildly in aspects such as the platform's SDK API language and project structure. With these new added complexities, it becomes particularly hard to make an all-in-one solution that becomes sufficiently accepted by the community. More so if it's supposed to work for a wide range of Scheme systems seamlessly.

Taking these issues into account, the main motivation behind SchemeSpheres is providing a framework for rapidly building applications for multiple platforms. This functionality is similar to what package systems and central repositories provide, but more th emphasis is placed on the workflow and construction, instead of distribution. SchemeSpheres is focused on the project setup and construction, simplifying the creation of projects for multiple platforms and providing a set of tools for the build/debug/release cycle. It is sufficiently flexible to allow the Scheme developer to choose the right libraries and design the application architecture freely, thus preserving handling complete control to the final suer. For this to become possible, a working framework should provide the following key features:

- Project construction automation, in all development platforms, for the different targets available in each.
- Project structure templates and build processes for all platforms, without hiding the details and allowing full customization per-project.
- Dependency handling and some form of package specification, so the system favours reusability and code sharing.
- Additionally, a set of libraries that supports most common needs for all target platforms.

1.2 Justification (current problems)

Mainly:

* Creating a multiplatform project involves many more aspects than just generating the code. Making it easy and readily available is very beneficial to the community. * Structuring, documentation and reusability also help in the process of building software.

Other things to take into consideration:

* The use of many SRFIs implies syntax-rules. Syntax-rules must be handled by the building system as well. * Other libraries require define-macro or syntax-case. * Short names can collide easily.

2. PROPOSED FRAMEWORK

2.1 Requirements

Needs: * Support for syntax-rules * Support for low-level macros (define-macros/syntax-case/rsc-macro-transformer), but those macros should be used only without interacting with syntax-rules. * Modules * Complete control over the workflow and build process * An R5RS+ Scheme compiler that can generate native or portable C code for the target platforms. Gambit has been chosen for this task.

2.2 Approach

1. Syntax expansion, modules, conditional compilation:

* syntax-rules mainly * low-level for isolated cases where it is really necessary to support a library * modules missing * cond-expand for conditional compilation

2. SchemeSpheres is based on the workflow comprised of three tools.

* Sfusion scaffolds out a new application, writing the Ssake configuration and pulling in relevant Grunt tasks and Bower dependencies that you might need for your build.

* Ssake is used to build, preview and test your project, thanks to help from tasks curated by the Yeoman team and grunt-contrib. A program to handle tasks (task runner), with procedures for build automatization in different platforms.

* Sspheres is used for dependency management, so that you no longer have to manually download and manage your scripts.

3. (proto)Ecosystem:

* An structuration and curation of modules into Spheres, with sharing of common and uniform functionality (such as FFI, ...) * documentation of the process * A central repository

3. EXAMPLE OF APPLICATION BUILDING

4. FINDINGS AND GOTCHAS

How to do: * the generators approach? did it work? * handling conditional compilation (cond-expand)

Main issues: * syntax expansion and proper error reporting.

* How easy it is * What advantages it has

5. CONCLUSIONS AND ROADMAP

* Roadmap: * Modules * Javascript backend * Limitations

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for