Multiplatform and mobile app development in Scheme with Gambit/SchemeSpheres

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

ABSTRACT

This document explains the motivation and fundamental ideas behind the SchemeSpheres[1] project. It outlines the conceptual approach taken by this framework, given a set of requirements that are considered essential. The different sides of the key aspects that SchemeSpheres tackles are summarized, focusing especially on the set of tools supporting multiplatform/mobile app development that it provides.

Current implementation supports most of the required functionality, which will be explored along with the process of building an example app. Limitations of the framework will be discussed during the tutorial, exploring possible future solutions as well current workarounds.

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, Gambit, multiplatform and mobile development, workflow automation, software construction

1. INTRODUCTION

Implementation fragmentation and diversity have been a long-standing attributes 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, and 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 multiple 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. Moreover, mobile platforms usually involve very particular toolchains that are rapidly evolving over time. With these new added complexities, it becomes particularly hard to make an all-in-one solution that becomes sufficiently accepted by the community. It is even more challenging 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 to provide a framework for software construction targeting multiple platforms, including mobile. This functionality is similar to what package systems and central repositories provide, but more emphasis is placed on the application construction workflow. Consequently, SchemeSpheres is focused on the project setup, dependency handling, and construction 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 user. To make this 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.
- A common documentation system, manuals and guides.
- Additionally, a set of libraries that supports most common needs for all target platforms.

[1]The project was initiatied by Fourthbit, and comprises code by many authors. Project homepage: http://schemespheres.org

2. SCHEMESPHERES FRAMEWORK

2.1 Requirements

In addition to the key features previously mentioned, the system should provide a solution to a number of issues that can be considered orthogonal to these. Namely, a Scheme framework with the purpose explained above needs to solve some aspects that are not part of the language standard nor considered commonplace among implementations.

1. Complete control over the workflow and build process, as opposed to a black-box model of compilation where the user obtains a library/executable without access to object files or generated code.
2. An R5RS+ Scheme compiler that can generate native or portable C code for the target platforms. Preferably, it should provide a more general backend that can eventually support compilation to other languages as well, opening the possibility of accessing certain platforms SDKs directly without bridging technologies such as JNI.
3. Support for syntax-rules. Several SRFIs require this macro system, not always readily available in Scheme implementations.
4. Support for low-level macros (*define-macros*/*syntax-case*/*rsc-macro-transformer*) for very specific, but important, cases.
5. Avoid name clashes through a modules system, namespaces or names convention.

Some of these are solved by the Gambit compiler; others are addressed by SchemeSpheres or customized third-party libraries (like macros and modules).

2.2 Approach

SchemeSpheres is built upon the Gambit Scheme compiler. This compiler has been chosen for its backend capabilities, performance and versatility. Being an R5RS implementation without full syntax-rules integration, it requires the SchemeSpheres system to provide it. The approach proposed by this framework on top of Gambit is divided into three fronts:

Hygienic macros expansion, modules and conditional compilation. These features must be usually solved together, as they all transform the code at compile-time. Many compilers, as is the case with Gambit, leave hygienic macros and related syntax expansion to the user implementation.

Workflow tool-chain. This is perhaps the most distinctive feature of SchemeSpheres in comparison with package systems. Essentially, the framework provides a set of tools for creating workflows, instead of forcing the user through a generic one. The workflow is divided into three parts: **project skeleton generation**, **task running**, and **dependency management**. These are handled currently by three tools:

- **Sfusion** scaffolds out a new application, writing the Ssake configuration and pulling in relevant Ssake tasks and dependencies that you might need for your build.
- **Ssake** is used to build, preview and test your project, thanks to help from tasks provided by the SchemeSpheres project for automating the workflow.
- **Sspheres** is used for installing dependencies from a curated set of modules and packages (called *spheres*, so the user can easily bring compatible code into an existing project.

Ecosystem. The set of aspects that help SchemeSpheres build community and collaboration around the project. Such a project benefits from the contribution of any Scheme developer that may desire to, so it must be prepared to integrate modules, tests, generators and tasks. The main points that fall under this category are:

- An structuration and set of curated modules into Spheres, with common and uniform functionality (such as FFI, ...)
- Documentation of the internal workings, plus extensions capabilities.
- A central repository.

3. TUTORIAL: EXAMPLE APP

The tutorial will explain the process of building an example application for various targets using OpenGL/ES graphics and the SDL library for application setup. The process will be divided into the following steps:

1. Set up a new host OS application. This will generate the base skeleton of application. Prototype functionality in the host OS.
2. Add support for Android. Conditionally compiled code for mobile (currently only affecting Android). App deployment.
3. Add support for iOS. Conditionally compile code only for iOS. App deployment.
4. Appendix. Extending SchemeSpheres: Ssake tasks and Sfusion generators.

4. CONCLUSIONS AND ROADMAP

This tutorial shows the current state of the tools that SchemeSpheres facilitates to the Scheme developer and how they may be used in the software construction process. As an organically growing software project, many parts will probably evolve substantially. The next natural steps for this frameworks will be:

- Improving syntax expansion integration with Gambit.
- Full modules support.
- Project generators for the JavaScript/HTML5 target.