Javascript as an Intermediate Language for FRP

Subtitle Text, if any

Name1
Affiliation1
Email1

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords keyword1, keyword2

1. Introduction

Functional Reactive Programming (FRP) is an exciting and popular way to make interactive applications in a functional style. However, creating production level applications with FRP remains prohibitivly difficult, especially for mobile platforms. While many functional programmers have the programming experience to write FRP applications, technicial issues have kept this circle to only the most dedicated FRP researchers. Using Javascript as an intermediate language for FRP code eliminates most of the technical hurdles to publishing applications written with FRP, allowing more programmers to use FRP in production.

Development of interactive apps tends to be faster and easier using FRP. One particular strength is that prototypes can be generated very quickly, and these prototypes are easily extented to full applications. This has been used in many interactive multimedia applications such as music, robotics, and games[].

2. Background

FRP implementations include many libraries in Haskell, the dedicated FRP language Elm, and the Sodium project which ports FRP to more language like Java and C. FRP often feels most natural in high-level functional languages like Haskell and Elm, but mobile development requires device specific languages (e.g Java for Android). Ideally, FRP programs could be cross-compiled to the target platform without significant interference in the development process.

For development on Apple's iOS platform, a cross-compiler exists for GHC that could provide a way to use a Haskell FRP library to develop iOS apps. This platform has not yet been a focus for FRP researchers, likely due to a community preference for Android, but also the difficultly installing the cross-compiler. Windows phone development remains equally unexplored. Perez, in work with Keera Studios, has succesfully cross-compiled Haskell to Android and published the game *Magic Cookies*[]. Although promising, the source code for this remains closed source and the setup has been difficult to reproduce due to many installation steps.

To simplify the process, we compile FRP program to an intermediate language, then use a *platform agnostic* tool to compile the intermediate language to any target device desired. The intermediate language presented here is Javascript and platform agnostic tool is Phonegap. For Haskell, ghcjs can compile program to Javascript, while Elm is compiled to Javascript by default. Phonegap is a program developed to allow users to create apps for all platforms by writing Javascript. We instead send the generated (and incomprehensible) Javascript to Phonegap to create a package for any mobile device. Here Javascript is functioning as an intermediate language - the user does not need to look at it for the system to work.

Performance is a concern when using an intermediate language without optimizations. A common measure for performance in FRP systems is the frames per second (fps) that can be displayed, or the frame rate. Any number above 32 fps is generally acceptable, and in our proof of concept app we were able to acheive a frame rate of XX fps. However, this is only for a 2D game, so performance measures for more complex system will be needed in future work.

BEGIN NOT FINISHED SEC Mobile development also relies on the use of two kinds of APIs, software and hardware. Software APIs (e.g. google maps) would require a .jar file loaded to the android project - now we just use the javascript API for that library. Hardware APIs (e.g. acceleration data) use device specific APIs. Some javascript APIs exist for functionality most devices have, like aceleraometers.

Use a double-level FFI, just as with compiling. If native code is necessary, Phonegap provides Plugin code can be called via the FFI in javscript, which is in turn called through the source languages FFI. Both ghcjs and Elm provide a javascript FFI.

Development may also be split between any combination of the FRP language, javascript, and the target language. END NOT FINISHED SEC

This approach is unique because it treats the high-level language of one tool as an intermediate language for another. It is unique in its simplicity because it leverages Phonegap to unify the workflow for all platforms - iOS, Android and Windows apps are all created from the same system. There is no need to install a different cross-compiler for each target device. This approach also works well in practice, providing good performance and access to low-level APIs when needed.

BEGIN NOT FINISHED SEC

3. Results

We developed a proof of concept app using Elm and have provided the source code and app files at This app uses FRP to read the touch interface native to the mobile hardware and

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

control a game. The app also uses a plugin to show ads, mixing javascript code with Elm. It can run at ?? frames per second on an Nexus 5 running Android verison X.XX. pass in (fps 200) as the time signal to a display of "toString (1000/t)" to meausre actual FPS - capped at 200 on laptop.

The development of FRP and functional languages in general relies on adoption by industry. Simplifying the workflow to bring FRP code to market will encourage future work in FRP. END NOT FINISHED SEC

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

Acknowledgments, if needed.

References

[] P. Q. Smith, and X. Y. Jones. ...reference text...