CO600 Project: Jarlang
Personal Report

Chris Bailey

cb661@kent.ac.uk

School of Computing

University of Kent

United Kingdom

Word Count: 6,335 (via texcount)

March 31, 2018

1 Project Contribution

Throughout the project, it was primarily Andrew working on the Erlang codebase; implementing and refining our abstract-syntax tree translation module which quite literally is the main functionality of Jarlang as a compiler. On the other hand, Nick's contribution to the project was spent almost entirely in implementing the Jarlang runtime environment (our JavaScript codebase) so that we could actually execute the result of Andrew's work.

My contributions to the Jarlang project as a whole however, have been more or less equally distributed between both codebases, in both actually implementing features or bugfixes, as well as ideating and helping both Nick and Andrew acheive what they have. I will outline my main contributions below:

1.1 Project Architecture

One of the most notable contributions I've made to this project as a whole involve the several architectural decisions I've made in order to make this project as clean, extensible, testible and simple as possible.

I was the one who originally instigated the project, as well as the one who first started writing some code outlining the project and as such the initial direction the project went was because of said outlining. In addition to this, the majority of the research done for this project (i.e. finding other similar projects such as LuvvieScript, as well as finding out about useful tools which aided in our AST generation / design) was done by me which laid a strong foundation for future development.

As the one in our project which multitasked most, working on both the Erlang and JavaScript codebases, whenever I noticed something get very messy, or unneccessarily complex, I would call everyone to band together and decide about a good refactoring we could perform.

On many occasions, I called for refactorings which whilst they took valuable time away from functionality implementation, they made future functionality implementation much easier to perform. This happened many times on both codebases, such as when I called for a refactoring to Andrew's AST translation module which originally heavily used pattern matching to determine what functions to run to translate a given AST node. Whilst there was nothing wrong with this in principle, refactoring to make use of switching on a given AST node's type field was much more readable and extensible to someone uninitiated in our project.

Another example in our JavaScript codebase is the fact that we were originally creating classes for our JavaScript representations of Erlang datatypes with the ES5 closure method, which works very nicely but gives us some drawbacks. The main drawback this faced was discovered when we wanted all of our datatypes to be comparable to eachother with common functions found in the runtime system, and as such the ideal and most elegant solution to this was to refactor our codebase to use ES6 Class constructors which allow us to use the 'extends' keyword — allowing all datatypes to inherit from a base class and override values and functions as needed.

Thus at a high level, one could say that I was responsible for our project following good software engineering practices, as well as consulting, commenting on and improving on our codebase iteratively throughout the project duration.

1.2 The Jarlang Compiler

As well as architectural / software engineering based contributions, I also worked on several features in the Jarlang compiler which follow.

Being the one who started our Erlang codebase, I was the one who researched methods of hooking into the Erlang compiler and extracting from it our CoreErlang AST which our project depends on. In addition to this, I also wrote an open source tool, Erlpkg, which not only builds binaries for Erlang projects, but also generates help text for good UX as well as parses command line options for users allowing us to support plenty of quality-of-life features such as outputting our intermediate compilation steps or toggling multithreaded compilation for better Erlang errors.

I was also the one who implemented module compilation in the Jarlang compiler, so whilst Andrew did the much more difficult task of compiling functions into JavaScript, I implemented code which took functions and a module prototype and created a correct JavaScript module out of these. To aide with this I implemented an interface for generating JSON nodes which represent JavaScript AST nodes which is used extensively by the core of the compiler.

Lastly, we all originally had wildly different programming styles and as such, over time, I enforced a common style guide for our Erlang modules such that it was easy and regular to read no matter what module you were reading. I integrated Dialyzer (a static analyzer for Erlang), as well as Elvis (a linter for Erlang) to aide with this.

1.3 The Jarlang Runtime

During the course of the project, I not only architected how we should implement the runtime system, but I also actively took a role in implementing parts of it.

The segments of the Jarlang runtime I implemented primarily centered around implemented actor-model primitives in JavaScript including writing the actor agents themselves, implementing message passing functions to allow actors to communicate, as well as implementing the PID and Process datatypes such that PIDs would be unique identifiers for Processes, which would run asynchronously and encapsulate Erlang style concurrent behaviour.

In addition to this, whilst Nick primarily worked on designing the other datatypes Jarlang uses, as well as implementing much of the base modules current integrated into the runtime (IO and Erlang modules), I implemented the core of the runtime including emulation of the process queue, atom table, ETS table and functions in any given module which interacts with processes.

I also implemented the layers responsible for interopability between the

JavaScript runtime and our Jarlang runtime.

2 Project Reflection and Evaluation

3 Conclusion