Kiko Fernandez-Reyes
kiko.fernandez@it.uu.se
Uppsala University, Sweden
https://www.plresearcher.com

SUMMARY

My research focus is on programming languages, language abstractions, formal methods, compilers, and type systems. I am one of the main contributors of the concurrent and actor-based language Encore. My contribution to the Encore programming language ranges from implementation of lock-free data structures for the runtime (written in C) to the Encore compiler (written in Haskell). I have collaborated with researchers from different universities, wrote international publications (outside my main research area) and supervised bachelor thesis. I plan on defending my PhD thesis at the end of August 2020. Prior to my PhD studies I worked in a startup and later as a consultant for the Trifork Stockholm office.

My main interests are programming languages and abstractions, formal methods, type systems, and software language engineering. In the future, I would like to extend my knowledge in these topics, and use real-world case studies to motivate the need of formal methods in industrial settings.

AWARDS & SCHOLARSHIPS

- Distinguished Artifact Award at Software Language Engineering (SLE'19).
- Distinguished Artifact Award at European Conference on Object-Oriented Programming (ECOOP'19).
- Best Paper Award in Coordination Models and Languages (COORDINATION'18).
- Best Paper Award in Int. Federated Conference on Distributed Computing Techniques (DisCoTec'18).
- Scholarship from MINT [1] to present our gamification approach in an advance course at the Educational Symposium (EduSymp'18).
- Scholarship (*Beca de Proyecto de Colaboración, 2011*) from the Programming Language department at Universidad de Málaga, awarded with 3 000 Euros, to collaborate and expand the teaching platform of the NGO Doctors Without Borders (Médecins Sans Frontières).

EDUCATION

M.Sc. **in Artificial Intelligence**
Luxembourg University, Luxembourg

Erasmus scholarship I did the first year of the Artificial Intelligence programme with focus on different logics, knowledge representation, and data mining.

M.Sc. **Computer Science**
Universidad de Málaga, Spain

5-year Engineering degree in Computer Science where I took courses in parallel programming, statistics, neural networks, compilers, object-oriented, functional, and logic programming, among others.

[1] Centrum för Ämnesdidaktisk Forskning inom Matematik, Ingenjörsvetenskap, Naturvetenskap och Teknikvetenskap (MINT); (English) Center for Subject Didactics Research in Mathematics, Engineering, Natural Sciences and Engineering Science

ACADEMIC EXPERIENCE

Head Teacher for Advanced Software Design (ASD) Fall 2016 – onwards
Uppsala University, Uppsala

From Fall 2014 until Fall 2015, I was the teaching assistant for this course. From Fall 2016 onwards, I became the head teacher of this course, which is beyond the expected duties of a PhD student. In this course, with 80 students yearly, I teach advanced design patterns for object-oriented languages. During the lectures I connect design principles and design patterns with open source implementations, so that students get a better understanding by reading well-tested and well designed code. Under my taking, the overall satisfaction of students increased (based on course evaluations) taking the course from a mean of 2.6 (and median of 3) before I took on the course, to a mean of 3.9 (and median of 4.0) – scale from 1 to 5, the higher the better.

Teaching Assistant for Software Engineering Fall 2015 – Fall 2017
Uppsala University, Uppsala

I taught Scrum, and coached students to use this agile methodology. I guided teams, and became familiar with best software engineering practices. I also assisted the head teacher with other duties, such as drafting exam questions, and correction.

Bachelor thesis supervisor Fall 2015 – onwards
Uppsala University, Uppsala

- Micael Loberg, *Encoding of Algebraic Data Types in Encore*
- David Escher, *Parallel Performance Comparison between Encore and OpenMP using Pedestrian Simulation*
- Alexis Remmers, *Enhancing functionality with assistive error visualisations*
- Joy van den Eijkhof, *For-Comprehension: An Encore Compiler Story*

INDUSTRIAL EXPERIENCE

Software Engineer Spring 2013 - Fall 2014
Trifork, Stockholm

I worked as a consultant, involved in different projects that required communication with clients in Denmark and Switzerland. I developed an automatic inventory of wine cellars that required RFID, Raspberry PI, server with REST API (Python), and orchestration and deployment using Ansible. I also wrote a financial application that does a forecast of assets for fund managers, written in Clojure.

Software Engineer Spring 2012 – Spring 2013
Guidepal, Stockholm

My main duties were related to building a scalable travelling social network for which we relied on services provided by Amazon (AWS S3, EC2, ElasticBean, etc.); we used MongoDB and wrote the backend in Java. I also contributed with a hybrid mobile application written in JavaScript, using Sencha Touch 2 and PhoneGap.

PROJECTS

Type checker for an object-oriented language 2019

I developed (together with Elias Castegren) a type checker for an object-oriented language. We took inspiration from the implementation of the Encore type checker and wrote it Haskell. We started with a very basic type checker and added more advanced features with minimal changes to the compiler, using concepts such as monads, functors, monoids, monad transformers, typestate, and phantom types.

Automatic conversion of tests: parsing and code generation 2016

The Encore language transpiles Encore files into C files, which are linked to the runtime library to produce an executable. The idea of this project was to automatically update tests from the old Encore syntax to the new Encore syntax during a one day (8 h) hackathon. This task involed updating the code generation phase of the Encore compiler, to output Encore code using the new syntax instead of emitting C code, and creating a new parser for the new syntax. A flag was added to the compiler so that we can use the old parser, read test files, call the code generator that outputs new Encore syntax, parse the new Encore syntax, continue with the typechecking phase, call the C code generator and run the tests. It end up working quite nicely.

Prototype a parallel collection in Clojure 2016

Prototype implementation of the ParT parallel collection with influences of Orc combinators. The end result is a parallel collection that can create pipeline parallelism relying on futures and future chaining operation. The lack of types in plain Clojure made me take some pragmatic decisions against the Encore implementation, such as flattening the structure due to lack of type information.

VOLUNTEER SERVICE

Academic Service

- Artifact Evaluation Committee of ECOOP 2020
- Student Volunteer Co-Chair of SPLASH 2019
- Programme Committee of AGERE 2019
- Publicity Chair of DisCoTec 2019, 2020
- Subreviewer of international conferences and journals

Social Service

- **Promotion of Gender Equality**
  I have been the vice-chair of the ACM-W student chapter at Uppsala University for the last 3 years, promoting gender equality. As part of this activity, we successfully got a 6 000 $ grant from Microsoft, to continue promoting gender equality at Uppsala University. This led to the organisation of the Ada Lovelace celebration in 2017, where we brought international speakers and researchers to promote gender equality. We also successfully got a 30 000 SEK scholarship from ID24 to promote gender equality.
- **Promotion of Open Source technology**
  I write articles at `opensource.com` and DZone. These articles are always posted also in different student channels to promote open source technology among students at Uppsala University..
- **Advocating for Privacy**
  I am part of a student chapter in Uppsala that promotes privacy and security technologies, ranging from topics such as email encryption to how to surf the web safely.
- **Bridging the gap between industry and academia**
  As part of promoting the open source technologies, I was accepted to an industry conference, PartialConf'18, where I explained how we have built the open source compiler for the language Encore, developed at Uppsala University (Partial'18 in Sofia).

- **Helping NGOs**
  To guide my Master thesis project, I did a collaboration with the NGO, Doctors Without Borders (Médecins Sans Frontières), where I was in close communication with the e-Learning Project Leader Manuel Lorente, and where I built and designed a new Personal Learning Environment. This platform was integrated with their learning platform, Moodle. I also developed a REST API and a simple Android client application for ease of use.

PROGRAMMING LANGUAGES

- *Java:* I have been teaching advanced software design patterns in Java from 2014 – onwards, and I have used it professionally during my industry experience for writing a scalable backend.

- *C:* I have used it to extend the concurrent Encore runtime, designed and developed a lock-free data structure as well as minor runtime libraries.

- *Haskell:* I have used it from 2014 – onwards to build the Encore compiler. I have influenced the design of the compiler, and used advanced typing extensions. I fear no monads nor arrows.

- *Scala:* I used Scala to prototype an ECOOP artifact and test the limits of the language. This prototype implicitly inserts flattening rules to avoid the future proliferation problem, *i.e.,* future-based APIs expose their internal communication structure. In a functional setting, this can be seen as a monadic future where there is an implicit `join` operation on nested futures. I also toy with an implementation purely written in terms of the indexed state transformer monad (but it did not perform well performance-wise).

- *Python:* I used it when I worked in industry some years ago, to track the status of goods using RFID. The system interacted with a REST API, was running on a Raspberry Pi, and the information was displayed in a website as well as on an iPhone app.

- *Clojure:* I used it when I worked in industry as the backend of a large asset management company, where the application generated reports and predictions on securities on the stock market.

PUBLICATIONS

Developing a Monadic Type Checker for an Object-Oriented Language: An Experience Report

E. Castegren, **K. Fernandez-Reyes**
International Conference on Software Language Engineering (SLE), Oct'19

We report our experience using Haskell to develop a compiler for an object-oriented language, focusing on the implementation of the type checker. The paper starts from a simple type checker to which we add more complex features, with minimal changes to the overall initial design.

(Artifact) Developing a Monadic Type Checker for an Object-Oriented Language: An Experience Report

E. Castegren, **K. Fernandez-Reyes**
International Conference on Software Language Engineering (SLE), Oct'19

ACM Distinguished Artifact Award

Reproducible artifact from the research paper *Developing a Monadic Type Checker for an Object-Oriented Language: An Experience Report*. The artifact contains an incremental implementation of the monadic type checker. This design is based on the work performed on the Encore compiler. The code is well-documented, with an intermediate section to the type state design (Section 9 in the paper) approach with a less steep learning curve on type-level programming, contains tests, and plenty of examples.

Run, Actor, Run - Towards Cross-Actor Language Benchmarking

S. Blessing, **K. Fernandez-Reyes**, A. Mingkun Yang, S. Drossopoulou, T. Wrigstad
Workshop on Programming based on Actors, Agents, and Decentralized Control (AGERE), Oct'19

The actor paradigm supports the natural expression of concurrency. It has inspired the development of several actorbased languages. The adoption of these languages depends to a large extent on the runtime characteristics (i.e., the performance and scaling behaviour) of programs written in these languages, In this paper we investigate the relative runtime characteristics of Akka, CAF and Pony, based on the Savina benchmark suite.We observe that the scaling behaviour of many of the Savina benchmarks does not reflect their categorization (into essentially sequential, concurrent and parallel), that many programs have similar runtime characteristics, and/or that their runtime behaviour may drastically change nature (e.g.,go from essentially sequential to parallel) by tweaking some parameters. This observation leads to our proposal of a single benchmark program which we designed so that through tweaking of some knobs (we hope) we can simulate most of the programs of the Savina suite.

Godot: All the Benefits of Implicit and Explicit Futures

K. Fernandez-Reyes, D. Clarke, L. Henrio, E.B. Johnsen, T. Wrigstad
European Conference on Object-Oriented Programming (ECOOP), Jul'19

Concurrent programs often make use of future, which are handles to the results of asynchronous operations. Futures can be characterised as either implicit or explicit, depending on the typing discipline used to type them. Existing approaches to implementing futures suffer from "future proliferation", either at the type-level or at run-time. Many languages suffer both kinds. Previous work offer partial solutions to these problems of future proliferation; in this paper we show how these solutions can be integrated in an elegant and coherent way which is more expressive than either system in isolation. We describe our proposal formally, and state and prove its key properties, in two related calculi, based on the two possible families of future constructs (data-flow futures and control-flow futures). The first calculus relies on static type information to avoid unwanted future creation and the second uses an algebraic data type with dynamic checks. We also discuss how to implement our new system efficiently.

(Artifact) Godot: All the Benefits of Implicit and Explicit Futures

K. Fernandez-Reyes, D. Clarke, L. Henrio, E. B. Johnsen, T. Wrigstad
European Conference on Object-Oriented Programming (ECOOP), Jul'19
ACM Distinguished Artifact Award

Reproducible artifact from the research paper *Godot: All the Benefits of Implicit and Explicit Futures*. In this artifact we show how to prototype data-flow futures using common control-flow futures in the Scala language, the limitations and how to

overcome them.

Attached and Detached Closures in Actors

E. Castegren, D. Clarke, **K. Fernandez-Reyes**, T. Wrigstad, A. M. Yang
Programming based on Actors, Agents, and Decentralized Control (AGERE), Nov'18

This paper discusses the problem of who can safely evaluate a closure to avoid race conditions, and presents the current solution to the problem adopted by the Encore language. The solution integrates with Encore's capability type system, which influences whether a closure is attached and must be evaluated by the creating actor, or whether it can be detached and evaluated independently of its creator.

Forward to a Promising Future

K. Fernandez-Reyes, D. Clarke, E. Castegren, H. Vo
Coordination Models and Languages (COORDINATION), May'18
COORDINATION Best Paper Award
DisCoTec Best Paper Award

This paper presents a new construct, *forward*, that delegates the responsibility for fulfilling the current implicit future to another computation. Forward reduces synchronisation and gives futures promise-like capabilities. This paper presents a formalisation of the forward construct, defined in a high-level source language, and a compilation strategy from the high-level language to a low-level, promised-based target language. The translation is shown to preserve semantics. Based on this foundation, we describe the implementation of forward in the parallel, actor-based language Encore, which compiles to C.

The Impact of Opt-in Gamification on Students' Grades in a Software Design Course

K. Fernandez-Reyes, D. Clarke, J. Hornbach
Educators Symposium (EduSymp), Oct'18

An achievement-driven methodology strives to give students more control of their learning with enough flexibility to engage them in deeper learning. We observed in the course Advanced Software Design, which uses the achievement-driven methodology, that students fail to get high grades, which may hamper deeper learning. To motivate students to pursue and get higher grades we added gamification elements to the course. To measure the success of our gamification implementation, students filled out a questionnaire rating the enjoyment and motivation produced by the game. We built a statistical regression model where enjoyment and motivation explain 55% of the variation in grades, and report on the experience and design of the gamification approach.

A Survey of Active Object Languages

F. de Boer, V. Serbanescu, R. Hähnle, L. Henrio, J. Rochas, C. Chang, E. B. Johnsen, M. Sirjani, E. Khamespanah, **K. Fernandez-Reyes**, A. M. Yang
ACM Computing Surveys, Nov'17

This paper presents the design and implementation decisions regarding four actor (active object) parallel languages: Rebeca, ABS, Encore and ProActive. The paper analyses the design and implementation decisions based on the following key points: objective of the language, degree of synchronisation, transparency, data sharing, formal semantics and their implementation and tooling support.

Extended Abstract: Affine Killing

K. Fernandez-Reyes, D. Clarke
Type-Driven Development (TyDe), Sep'2017

This extended abstract introduces how to kill off speculative computations in speculative, parallel abstractions. It relies on an affine type system that leverages static information to safely kill ongoing speculative computations applying thread-local reasoning.

ParT: An Asynchronous Parallel Abstraction for Speculative Pipeline Computations

K. Fernandez-Reyes, D. Clarke, D. S. McCain
Coordination Models and Languages (COORDINATION), May'16

This paper introduces a calculi, the design and implementation of a new asynchronous, speculative, parallel abstraction to express complex coordination patterns. Values, futures and arrays can be lifted to the abstraction, which is controlled by non-blocking high-order combinators. The most complex combinator permits to safely kill speculative pipeline parallelism.

Parallel Objects for Multicores: A Glimpse at the Parallel Language Encore

S. Brandauer, E. Castegren, D. Clarke, **K. Fernandez-Reyes**, E. B. Johnsen, K. I Pun, S. L. Tapia Tarifa, T. Wrigstad, A. M. Yang
School on Formal Methods for the Design of Computer, Communication, and Software Systems (SEFM), May'15

This paper summarises the design decisions of a new language for many-core computers, Encore, presents its main parallel and concurrent constructs and a formalisation of its core features.

REFERENCES Upon request.