ActorForth* – Architectural Drivers

by Benjamin Scherrey
January 13th, 2019

Aligning problem domains, mental models, and language implementation.

Introduction

The application of imperative, shared-state, deterministic, single thread concurrency model programming languages to address distributed, decentralized, non-deterministic problem spaces creates such an impedance mismatch systems implemented in such languages are fundamentally incapable of fully modeling the problem space nor capable of delivering a correct solution. Classic examples of this phenomenon can be seen in the emergence of distributed "block chain" crypto-currencies, unfortunately named "smart contracts", and the ever prognosticated "DAO" or distributed autonomous organization. All of these technologies show great promise but their implementations suffer from existential defects that undermine their credibility and make them unfit for their stated purposes.

As these technologies conduct their way through the third, "Trough of Disillusionment", phase of the "New Hype Cycle"[1], this paper argues that the impedance mismatch must be fixed once and for all if progress is to continue and their ultimate potential realized. We propose a new programming environment that addresses these challenges architecturally. A programming language that provides for clear and enforceable abstractions inherent in the problem space, aligns the programmer's mental model with the actual implementation of the solution, and yet is simple enough in its implementation to be fully understood by the developer.

Broken Abstractions

"all models are wrong, but some are useful"
– George E. P. Box

Since the 1950s, imperative procedural programming languages implemented on top of von Neumann architected machines have been the basis for the vast majority of computing systems. An impact of Moore's law[2] has been that rapid advances in hardware and its inversely proportional cost relationship have surpassed or certainly kept pace with the ability of programming languages and software models to implement solutions necessary to forge the new information age economy.

Problems with these types of architectural styles were encountered and identified fairly early on. Two of the most significant realizations and suggestions on how to improve things come from Edsger Dijkstra's[3] championing of structured programming, and John Backus'[4] introduction

1 Hype Cycle (https://en.wikipedia.org/wiki/Hype_cycle and https://en.wikipedia.org/wiki/Hype_cycle#/media/File:Hype-Cycle-General.png) <try to find a better reference>

2 <moore's law reference>

3 In Commun. ACM 11 (1968), 3: 147-148 https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF

4 Turing Award Lecture, 1977, In Commun. ACM 21 (1978), 8: 613 – 641 http://worrydream.com/refs/Backus-CanProgrammingBeLiberated.pdf

* ActorForth is only a working title for now. It may somewhat accurately describe what the language is (probably should be "Strongly Typed Ontological Actor Context Interactive Forth") but that might just distract from what the language does and confuse potential adopters. Perhaps that's why all the popular languages have names that don't mean anything?

<Second Header Title>

<Second Optional Header Subtitle>

of the functional programming style and identification of the "von Neumann bottleneck".

Slow functional systems & injecting another level of indirection…

Presumption of a "black box" von Neuman machines physically isolated/protected from <unmanaged> access breaks down in "white box" decentralized systems where anyone can access and change anything at any time. <consensus model><does this limit the types of consensus models that can be considered? what are our other possibilities we can enable?>

Current Practices

The domain of concurrent and often distributed systems not addressed by current common programming languages. CSP inadequate. Actor model best fit.

misaligned mental state caused by this impedance mismatch forces the developer to simultaneously maintain at least two conflicting mental models and the additional effort of reconciling their conflicts effectively tripling the effort necessary to delivered the desired functionality. While the cost complexity increases linearly, the product quality deficit and associated risks increase exponentially.[5]

programming languages should put the developer into a correct mental state to properly reason about the problem space being addressed.

A correctly designed programming language enforces an architectural style that makes as many of these common defects actually impossible to occur in the first place.

Cardano (ADA)[6] is attempting this through their use of Haskell as a programming language. <limitations and impracticalities of lambda calculus>

All the research languages look like variants of lisp because it's an easy language to implement (no syntax to speak of – super easy to parse) and is the programming *lingua franca* for academia. Due to the above noted limitations of lambda calculus, we suggest that they're starting off on the wrong foot.

"Provable programs" not possible past a certain point of complexity. What's more important is simplicity c.f. "Social Processes and Proofs of Theorems and Programs" by Richard A. De Millo (https://www.cs.columbia.edu/~angelos/Misc/p271-de_millo.pdf). It is more important that software be made reliable than verifiable – primarily through social processes and experience.

Actor model is based on physics rather than math.

"All models are wrong, some models are useful" – George P. Box

5 <reference for code size/complexity → exponential defect/risk likelihood>

6 Cardano https://iohk.io/projects/cardano

<Second Header Title>

<Second Optional Header Subtitle>

Pushing the boundaries of existing computational models into contexts that only overlap rather than fully contain exposes their limitations and starts moving them away from the "useful" boundary towards becoming misleading and wrong.

Moore's law & von Neuman machines coming at the same time as real decentralized distributed apps

Distributed, asynchronous, computing platforms have recently made their impact felt through the popular adoption of crypto-ledgers. A cryptoledger is simply a non-refutable persistent store of immutable facts, typically time series based, managed in a completely decentralized manner.

Declarative rather than imperative flow control.

Indicators of a Well Designed Programming Language

Blah blah blah blah blah

Cryptoledger Ontologies

Extended type system supporting declarative constraint-based resolution?

Every good consumer cryptoledger application is based on a distributed open call bid system.

Integrity over Reputation

As simple as possible but no simpler...