CodeDepot: A Proposal for a Modern Software Ecosystem

Working Draft - Not final

Contents

Overview **1**

Introduction **2**

The CodeDepot Cooperative **3**

Overview 3

CodeCoin, The currency of the ecosystem **3**

Overview 3

CodeDepot, The decentralized software marketplace and code repository **4**

CodeChain, The decentralized processing engine **4**

Overview 5

FBP: The Architectural Inspiration 5

Limitations of FBP 6

Component Based Programming 6

Utilizing Software Translation 6

General Purpose Graphical Processing Units (GPGPU) 7

Everything is a Component 7

Decentralized Network 8

Processor Nodes 8

Secure Computing 8

Micropayment Calculation 8

User Interfaces 8

WorkSource **8**

Overview 9

CodeSweet **9**

Rapid Application Development 9

Agile Development Manifesto 9

Overview

Utilizing state-of-the-art crypto-currency technologies and an unconventional programming model, a complete paradigm shift in technology production is possible. Time-to-market for all software developers could decrease by a hundredfold. It is strongly believed that the innovations presented below, will change the way all computing is done in the future. ProDataLab would like to emphasize,

> that this truly amazing efficiency would not be possible without the ability to stand on the shoulders of giants.. the open source software community! ~Peter Alexander, ProDataLab

The artist Bob Dylan, once titled a song "Everything is Broken". We take that notion and consider it in reference to the conventions in today's software usage, sales, production and execution. A participant can find inefficiencies and unwarranted complexities practically everywhere in today's environment. We ask the reader to take a moment and think of their own experiences with software. Have you ever found yourself frustrated? Have you ever considered some aspect overly complicated, unintuitive or inflexible and thought to yourself "it shouldn't have to be this way"?

An end-user of software might have thought of how hard it is to get familiar with a software product or that the documentation and help system are simply not much help. They might have wished for a feature they needed to be available or that the there were too many features packed into the product. Small business owners may find navigating the obstacle course of software system selection and integration impossible for mere mortals. Software engineers may have thought of many areas of foolishness that surround their profession. The lists of areas ripe for innovation surrounding the software ecosystem are quite abundant.

Other aspects of concern and inefficiency can be seen in the marketplace itself. In some markets, Goliath sized companies hold monopolistic power over software marketplace silos. These concentrated marketplaces take large percentage commissions from a lowly software producer's product sales. They dictate rules that are at times overly stringent or simply used to control the domination they hold in the market.

This project's goal is to collaboratively make the experience of creating and using software as efficient and as intuitive as possible. The readers of this paper are asked to consider very thoughtfully what they themselves, specifically, would have improved, because it is that specific item or list of items that will be fixed by what is proposed in this paper and its addendums.

This paper is currently in its first draft and is itself an open source project for inspired persons to contribute their ideas to. Eventually it will reach a 1.0 status; at which point it will have the appearance more of a specification than of merely a white paper. At this point I encourage you to envision the possibilities of a collaborative effort to change the current paradigm surrounding the production, consumption and usage of software. You are encouraged to join this effort to modernize the entire software ecosystem via open source software and its principles. Become a contributing member of the CodeDepot Community.

Introduction

Software is ubiquitous in our lives whether we are users or producers. Yet, even a less than astute observer can see that the entire ecosystem is ripe for innovation. In this paper we will begin a conversation of the direction of the needed improvements and we ask the community to participate. This community that is mentioned, are the people that engage with software whether it be in its production, sales, tutelage or usage.. whether they are casual or power users. Having discussion and contribution globally and easily, will ensure that the best solutions of problems we all identify, will be addressed most thoroughly and efficiently.

Below are six foundational pillars of a proposed software ecosystem that may forever change the entire paradigm of our engagement with software. What is presented at this time are merely proposals that will be refined by any willing individual participant and all the participants in whole. The paper and its addendums will also introduce other worthy aspects and notations pertinent to the execution of this proposal.

The CodeDepot Cooperative

Together, an engaged society will undoubtedly find the best solutions to issues that face them. Over the past 40 years, the open source community has shown that open participation most often provides better results than that of a closed system. We present here a decentralized organization where every member of the community has the opportunity provide direction through proposals, discussion and ultimately to that of solutions. We suggest that Decentralized Autonomous Organizations [1], a new concept now available through cutting edge technology, will provide many benefits over conventional and centralized organizational structures where they are applicable.

Overview

The CodeDepot Cooperative is the working name of a proposed open governance model for the CodeDepot ecosystem; where directives and decisions are decentralized and democratized amongst the entire community [2]. This notion of decentralized governance and transparency is aimed at stemming the problems of any centralized governing system that naturally creates hierarchical levels of control.

> Bitcoin's underlying technology of the blockchain [3], brings a radical departure from the governance model of the old world. It follows trends of decentralization that have emerged through the internet in recent years including Bittorrent, Free Software and Open Source Movements and collaborative production platforms like Linux and Wikipedia. The essence of this game changing invention is distributed trust (no need for third party reconciliation). Bitcoin solves the scaling issue of trust. With its decentralized security, we can now create a more open and inclusive society at a global scale.
>
> Bitcoin's decentralized system opens a door to a new paradigm where people can choose to abide by a protocol of consensus which is a different from the logic of domination and control of a centralized paradigm. Technology can't solve everything. Technology is just a tool. It always needs to be accounted for by democratic consensus of people. Technology should never be used to replace human interaction and connection.. it should be used only to enhance it. [4]

CodeCoin, The currency of the ecosystem

The invention of blockchain technology in 2008, has provided for the world a whole paradigm shift in financial and contractual mechanisms. It is truly a marvel that will disrupt many incumbent institutions. Systems that once required intermediaries and centralized solutions can now be peer to peer and decentralized. Trust of persons and entities unknown can know be established without the need of 3rd parties for verification or validation. Moneys can now be sent digitally without any concern for fraud or corruption. Ability to make payments in fractions of a penny are now facilitated. These are truly amazing times and we as a community of software engagers can benefit from this remarkability.

Overview

CodeCoin is the proposed programmable money, or cryptocurrency [5], that will be used in this software ecosystem. It is essentially a token that will be used to monetize all the goods and services within the ecosystem. This cryptocurrency will be at the center of a well considered autonomous economics model of incentives and disincentives formulated by the community.

As with other aspects of this proposal, an emphasis for community thought leaders to be engaged in the formulation of these models is sought after and strongly encouraged.

CodeDepot, The decentralized software marketplace and code repository

The core component of the ecosystem is a marketplace where users and producers engage. This is composed of two parts where developers will have an interface to deposit software code into the system and users will have an interface to utilize it. Other participants will include documentation contributers, article writers, bloggers, audio and video producers. Actually, anything surrounding software and related technology.

As described in the section for CodeChain below, developers will be rights holders for what essentially are snippets of code called components in this paper's nomenclature. They will use CodeDepot as a repository and market interface for their components of code and other contributions.

Users will have many advantages over conventional software produced today including but not limited to:

- Flexibility
- Scalability
- Security
- Ease of use
- Accessibility
- Privacy
- Complete control and ownership of their personal data.

Mechanisms will be built-in that provide the end-user and easy ability to:

- Give seamless, instantaneous feedback.
- Directly request alterations or needed features.
- Intuit the product through complete documentation and autonomous tutelage.
- **Hire affordable freelancers directly from their user interface.**
    - For assistance or instruction.
    - For software developers to provide any possibly needed customization.
- **Possibly choose "payment methods"**
    - Advertising
    - Onetime fee for apps
    - Per execution cycle (micropayments)
    - Contractual, eg. monthly/yearly
    - Synergies via CodeDepot's economic partners
    - **Selling value of their usage characteristics**
        - e.g., Facebook business model, where the user is the product

CodeChain, The decentralized processing engine

The problems with conventional programming paradigms are numerous. In fact, at the time of the initial draft of this paper, querying Google's search engine for "problem AND programming AND language" produced 50.8 million results [6]. With many articles titled similar to "The Problem with Programming" [7], It seems fruitless to itemize these problems as a comparison to what is proposed here, let alone the problems of the entire software centric paradigm. Instead, as an introduction, we ask the following questions:

1. What if snippets of software were in essence Lego like reusable components that just snapped together, even autonomously into a desired structure?
2. What if once a component or a structure of components was created, it would never have to be created again by anyone? Yes, ever again as in the literal sense of forever.
3. What if once a component was created it would be shared by every piece of running software in the world that required it?
4. What if an executing software system, even a mission critical system, could be altered or replaced without even a nanosecond of downtime.
5. What if it did not matter which programming language was used to create an individual component and that any component could seamlessly communicate with others?
6. What if a software producer's time-to-market was reduced by a hundredfold?
7. What if a software's execution was most reliable and the most possibly secure from intrusion?
8. What if a software user's privacy was held in the highest regard.

CodeChain, it will be shown, is a system that could and will provide these desirable properties, as well as others.

Overview

CodeChain is a decentralized processing system [8] which at its core reflects the philosophy of component based software engineering (CBSE) [9]. It is entirely inspired by J.Paul Morrison's [10] Flow-Based Programming (FBP) [11]. The concepts of FBP are improved with secure decentralized computing, communication and database models from various sources including BitCoin [12], BitTorrent [13] and BOINC [14]. CBSE exhibits the very desirable property of loose coupling [15]. FBP, a particular form of dataflow programming [16] extends loose coupling to define bounded buffers, information packets with defined lifetimes, named ports, and most importantly a separate (lazy) definition of communication connections [17].

- FBP is essentially a 3 dimensional hierarchical graph.

FBP: The Architectural Inspiration

The following is taken from J.Paul Morrison's Website. [18]

> Flow-Based Programming is a programming paradigm that uses a "data factory" metaphor for designing and building software applications. Applications are defined as networks of "black box" processes, which exchange data across predefined connections by message passing, where the connections are specified external to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component orientated.
>
> It views an application not as a single, sequential process, which starts at a point in time, and then does one thing at a time until it is finished, but as a network of asynchronous processes communicating by means of streams of structured data chunks, called "information packets" (IPs). In this view, the focus is on the application data and the transformations applied to it to produce the desired outputs. The network is defined externally to the processes, as a list of connections which is interpreted by a piece of software, usually called the "scheduler".
>
> The processes communicate by means of fixed capacity connections. A connection is attached to a process by means of a port, which has a name agreed upon between the process code and the network definition. More than one process can execute the same piece of code. At any point in time, a given IP can only be "owned" by a single process, or be in transit between two processes. Ports may either be simple, or arraytype. It is the combination of ports with asynchronous processes that allows many long running primitive functions of data processing, such as Sort, Merge, Summarize, Collate, etc., to be supported in the form of software black boxes.
>
> **Because FBP processes can continue executing as long they have data to work on and**

somewhere to put their output, FBP applications generally run in less elapsed time than conventional programs, and make optimal use of all the processors on a machine, with no special programming required to achieve this.

The network definition is usually diagrammatic (see: 'Component Based Programming' below), and is converted into a connection list in some lower-level language or notation. FBP is thus a visual programming language at this level. More complex network definitions have a hierarchical structure, being built up from subnets with "sticky" connections .

FBP has much in common with the Linda language in that it is, in Gelernter and Carriero's terminology, a "coordination language": it is essentially

language independent. Indeed, given a scheduler written in a sufficiently low-level

language, components written in different languages can be linked together in a single

network. FBP thus lends itself to the concept of domain specific languages or

"mini-languages".

FBP exhibits "data coupling", described in the article on coupling[*] as the loosest type of coupling between components. The concept of loose coupling is in turn related to that of service-oriented architectures, and FBP fits a number of the criteria for such an architecture, albeit at a more fine-grained level than most examples of this architecture.

FBP promotes high-level, functional style of specifications that simplify reasoning about system behavior. An example of this is the distributed data flow model for constructively specifying and analyzing the semantics of distributed multi-party protocols.

Limitations of FBP

At a superficial level, FBP is an ideal programming paradigm that offers quite a few benefits over conventional paradigms. At scale though, there is a limiting condition of context switching, especially so on conventional general purpose CPUs [19]. For an FBP paradigm at scale, a point will be reached where the number of context switches on a single machine, multi-core CPU, overwhelms the system and causes notable latency. On average, context switching costs approximately 30 microseconds of overhead per occurrence. One benchmark of the theoretical limitations of context switching has an upper bound of 18.75% of CPU cycles wasted due to context switching. Generally, optimal CPU use is to have the same number of worker threads as there are hardware threads when a process is CPU bound, whereas I/O bound permit more [20]. These considerations puts the FBP paradigm at very much a disadvantaged ideal of maximal efficiency.

Component Based Programming

In order to overcome the conditional limitations of context switching per processing node, in a strictly FBP paradigm, we provide here an area of consideration to help maximize the efficacy of the CodeChain system. The term Component-based Programming (CBP) is coined here for the purpose of a enlisting a stronger emphasis on components over that of data flow as it is for FBP.

Utilizing Software Translation

The concepts fundamental to FBP (autonomous blackbox components loosely coupled via lazy linkage) can be easily considered at the various phases of the compilation stack prior to execution. Essentially what this means is that we can remove the constraints from that of each component, needing to be its own execution process or thread, yet still be most loosely coupled. We can redefine networked inter-process components to that of a virtual model, that can then be implemented, by encompassing one or all of the compilation's translation stages prior to execution.

1. Source code
2. Semantic analysis
3. Intermediate Representation (IR) code and its linkage

4. Machine code and its linkage
5. Just-In-Time compilation or interpreter engine

General Purpose Graphical Processing Units (GPGPU)

Another most exciting and promising consideration, is to apply the notion of CBP to include that of specialty hardware processors like that of GPGPUs [21]. GPGPUs provide a processing model of thousands concurrently executing threads. Utilizing these high-scale concurrent processors, one can imagine the promise of the original FBP concept of inter-communicating processes/threads and lazy linkage, without the burdens of scalability that are imposed when merely targeting that of a CPU architecture.

- Task-based and Data-based parallelism (MIMD)
- Scheduling and Communication need to be implemented
- **Task Parallel**
    - Irregular workloads with dependencies
- Both CUDA and OpenCL have open source LLVM backends
- **Whippletree for task-parallel model**
    - "Dynamic Irregular Workloads
- **Although designed for SIMD, MIMD is feasible.**
    - **A GPU Task-parallel model with Dependency Resolution**
        - http://www.idav.ucdavis.edu/func/return_pdf?pub_id=1091
    - **Whippletree**
        - http://data.icg.tugraz.at/~dieter/publications/Schmalstieg_286.pdf
        - https://github.com/apc-llc/whippletree

Everything is a Component

Todo

- Components are snippets of code
- Components can be atomic or composites, made up of other components.
- Chains (component networks) are defined either statically or dynamically.
- Components are virtually snapped together like Legos.
- High scale concurrency.
- Processing efficiency
- **Once a component is created:**
    - It can be reused and repurposed to anywhere it is possibly needed.
    - It will never need to be created ever again.. as in the literal sense of forever.
    - It can be virtually shared by every piece of running software, in the world, that depends on it.
- Can be comprised of code from any programming language**.
- All apps and "libraries" are comprised of networks of components.

- **The networked components model, lends itself overwhelmingly to visual programming interfaces (VPI).**
    - We propose vast "smart" improvements over conventional visual programming environments.
- **In the future, developers will have ready-made "base" component networks, in that they only need to minimally append and/or tweak what has been done before them.**
    - Extremely smart and efficient component "search" mechanisms.

Decentralized Network

Todo

Processor Nodes

Todo

Secure Computing

Todo

- zkSNARKS for C

Micropayment Calculation

Todo

User Interfaces

Todo

- Are Ubiquitous
- Are also component based.
- Very flexible and powerful thin/thick Clients.
- Graphical or headless
- Context menus have always been broken.
- Extremely customizable and "skinable".

WorkSource

WorkSource is a proposal for a open governance, decentralized, peer to peer marketplace for end-users to hire freelancers. It will employ modern cutting edge technology for monetization, accounting, reputation, contractual obligation and in the case needed, arbitration. The most prominent aim is to incorporate very simple access and functionality directly into the CodeDepot userinterfaces. Unlike current freelance market places, freelancers will be made to feel as equals and not of a second class, as compared to employers.

Non-technical users of software often find themselves in need of instruction or in need of customization. By incorporating direct and easy contact with software professionals, the users needs can be addressed painlessly, immediately and reliably.

Technical users find themselves paying exorbitant fees to hire developers at conventional centralized services. Often the employer will find these services confusing, frustrating and simply inadequate.

Freelancers will often find that scanning and applying for jobs, is simply too time consuming.

Overview

Current freelance and other work-sourcing like exchanges, are usually run by a centralized corporate entity that enjoy a significant percentage of the cost of the transaction; together with collecting monthly fees. In a decentralized; self-governed; peer-to-peer (P2P) marketplace there isn't any centralized entity, just a community of colleagues and clients. Freelancers enjoy the near entirety of the proceeds of their transactions without some third party dipping their greedy hands in.

New technologies, most significantly bitcoin's blockchain, have now enabled P2P marketplaces to thrive unencumbered by any need of a centralized entity or 3rd party. The need for trust is virtually eliminated.. providing free, flat, P2P markets.

WorkSource will be decentralized, community effort, that will provide reliable sources of service providers to those that need such services. Current cryptocurrency, and other new technology, make it possible to enable most efficient market ecosystems where trust and incentive/disincentive mechanisms are automated; built right in to the platform. This, together with ideas generated and implemented by the community, will make the platform most desirable and efficient to participate in.

CodeSweet

A programmers toolbox is most often burdensome and time consuming to be productive with. It could be argued that every tool in use by engineers is in some way problematic or simply incomplete. If every aspect of every tool and its interface were a component, then the programmer could fashion their tool and hence their toolbox to be just the way they liked it. That bears repeating.. If everything is a component, then the programmer could fashion their tool and hence their toolbox to be just the way they like it!

CodeSweet will be a component based toolkit where engineers have the ability to add features that they deem worthy.. leaving any others behind. Features like automation, intuitive instruction, reimagined user interfaces, and ease-of-use will be of strong focus.

Other aspects will include:

> Todo

- The best documentation tools and interfaces
- Employing intelligence and automation as much as possible
- Search and reference to be topped by no other
- Intuition
- Cutting edge compilation and translation chains
- "Code once for everywhere" user interface compiler

Rapid Application Development

> Todo

Agile Development Manifesto

> Todo

---

1 Decentralized Autonomous Organization http://en.wikipedia.org/wiki/Decentralized_Autonomous_Organization
2 Open-source Goverence Model http://en.wikipedia.org/wiki/Open-source_governance
3 Bitcoin's Blockchain http://en.wikipedia.org/wiki/Bitcoin#Block_chain
4 Bitcoin, The Beginning of Open-source Goverence http://falkvinge.net/2014/11/10/bitcoin-the-beginning-of-open-source-governance/
5 Loomio http://loomio.org
6 DemocracyOS http://democracyos.org
7 Cryptocurrency http://en.wikipedia.org/wiki/Cryptocurrency
8 Decentralized Computing http://en.wikipedia.org/wiki/Decentralized_computing
Problem with Programming
_q=problem+program+language&as_epq=&as_oq=&as_eq=&as_nlo=&as_nhi=&lr=&cr=&as_qdr=all&as_sitesearch=&as_occt=any&sa
10 The Problem with Programming http://www.technologyreview.com/news/406923/the-problem-with-programming/
11 Component-Based Software Engineering http://en.wikipedia.org/wiki/Component-based_software_engineering
12 J. Paul Morrison http://en.wikipedia.org/wiki/John_Paul_Morrison
13 Flow-Based Programming http://en.wikipedia.org/wiki/Flow-based_programming
14 Bitcoin http://en.wikipedia.org/wiki/Bitcoin
15 BitTorrent http://en.wikipedia.org/wiki/BitTorrent
16 BOINC https://boinc.berkeley.edu/
17 Loose Coupling http://en.wikipedia.org/wiki/Loose_coupling
18 Dataflow Programming http://en.wikipedia.org/wiki/Dataflow_programming
19 Flow-Based Programming http://ersaconf.org/ersa-adn/papers/adn003.pdf
20 J.Paul Morrison's Website http://www.jpaulmorrison.com/fbp
21 Context Switch http://en.wikipedia.org/wiki/Context_switch
22 How Long Does It Take To Make Context http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html
23 General-purpose computing on graphics processing units http://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units
24 LLVM compilation toolkit http://llvm.org