Bridging the gaps with Iolite blockchain

Iolite Foundation

Dmitry Kuzminov dmitry@iolite.io

Alfred Shaffir alfred@iolite.io

Vladislav Makarian vladislav@iolite.io

Abstract

Blockchain[1] is an undeniably ingenious invention which clearly presents societys needs for decentralized solutions in the realm of digital assets and similar developments (see Figure 1).

Implications of the need for Blockchain are almost unlimited. (see Figure 2).

The most advanced distributed public blockchain network is of course **Ethereum**[2] and the key technology that is driving this decentralized platform is **smart contract**[3]. A smart contract is an innovated idea, however due to its complexity and relatively limited exposure to the development world, this technology is still struggling to gain a foothold in the real world. A smart contract with an **ICO**[4] crowd sale or token offering created interesting solutions for crowdfunding[5], however a substantial number of products were never released. We believe it causes a major obstacle to effective communications between mainstream developers/engineers and market and blockchain specialists.

Our goal is to create a convenient solution that will bridge the gaps between these individuals and let all engineers with programming skills write their own smart contracts and rapidly deliver applications to end users. We have created an **FAE** (Fast Adaptation Engine) that will adapt any known programming language to smart contract architecture in a very short time. We plan to create IoLite plugins in most popular **IDE**[6] solutions.

We have based our work on the advanced solutions developed at Stanford University[7], which allow users to naturalize the core languages. This approach is used to build a fast adaptation of formal languages so they can be used to build smart contracts.

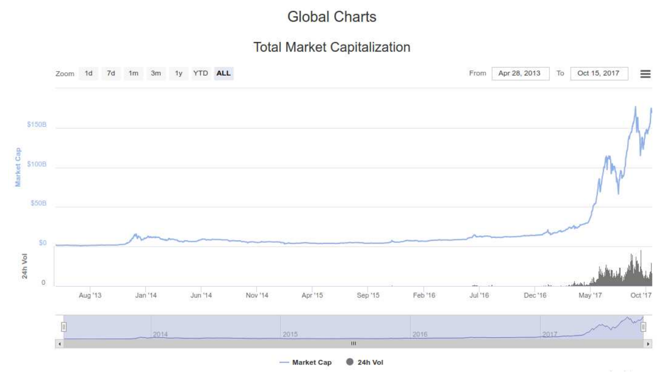


Figure 1. Global market growth from 2013 [8]

IoLite foundation

However, our technology involves much more than creating a faster and easier DApp adaptation . We are striving to create a new generation of blockchain that is built on top of knowledge and useful information that can be reused by creative scientists and programmers to develop new and exciting applications.

Our FAE engine and blockchain would serve many human to machine fields. Whenever formal language exists to control a machine such as hardware or software robots, (think about BOT), FAE will naturalize such language and create **knowledge** that would be available for reuse by any number of applications. Take, for example, robot-controlled parking lot management. FAE engines existing in a community would establish such a protocol to support a naturalized interface. Many applications with exciting UXs will utilize the IoLite blockchain to use the human to machine interface capability.

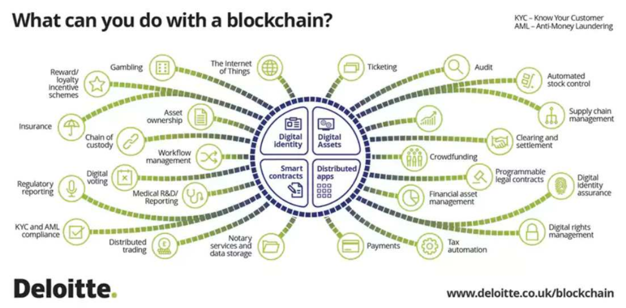
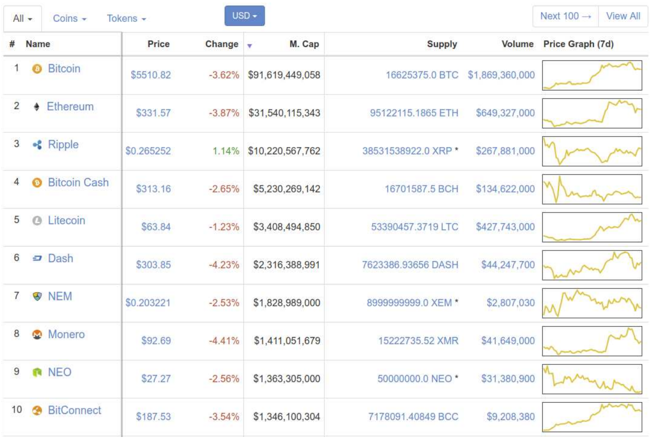


Figure 2. How blockchain technology could be used [9]

1. Introduction

Blockchain technology is a very important innovation engine in todays industry. It is driven by security and freedom. It is a decentralized form or records that eliminates borders and connects people around of world. The system of economics on which crpyto-currency is based, i.e. blockchain, has attracted many technology evangelists. Furthermore, Ethereum, the most popular blockchain for DApp (see Figure 3), is a special case that has motivated our team to bring new and advanced solutions. The heart of Ethereum technology is smart contract[3]).

> "Smart contracting 'technology could reduce banks infrastructure costs attributable to cross-border payments, securities trading and regulatory compliance by between $15 20 billion per annum by 2022'" Santander, The Fintech 2.0 Paper: Rebooting Financial Services, 2015



#	Name	Price	Change	M. Cap	Supply	Volume	Price Graph (7d)
1	Bitcoin	$5510.82	-3.62%	$91,619,449,058	16625375.0 BTC	$1,869,360,000	
2	Ethereum	$331.57	-3.87%	$31,540,115,343	95122115.1865 ETH	$649,327,000	
3	Ripple	$0.265252	1.14%	$10,220,567,762	38531538922.0 XRP *	$267,881,000	
4	Bitcoin Cash	$313.16	-2.65%	$5,230,269,142	16701587.5 BCH	$134,622,000	
5	Litecoin	$63.84	-1.23%	$3,408,494,850	53390457.3719 LTC	$427,743,000	
6	Dash	$303.85	-4.23%	$2,316,388,991	7623386.93656 DASH	$44,247,700	
7	NEM	$0.203221	-2.53%	$1,828,989,000	8999999999.0 XEM *	$2,807,030	
8	Monero	$92.69	-4.41%	$1,411,051,679	15222735.52 XMR	$41,649,000	
9	NEO	$27.27	-2.56%	$1,363,305,000	50000000.0 NEO *	$31,380,900	
10	BitConnect	$187.53	-3.54%	$1,346,100,304	7178091.40849 BCC	$9,208,380	

Figure 3. Top 10 blockchains, Ethereum is clearly leader as DApp blockchain [10]

To develop smart contracts on Ethereum, blockchain needs to use Solidity[11], a contract-oriented programming language. It is quite clear, that exposure to this language is not ubiquitous. The developer must study both the syntax and architecture of smart contract and this is no simple task. Of course, there are other solutions that employ smart contract technology. We will not review each of these in detail, but only mention that some of them introduce additional features and support, however, our development, IoLite is a totally different concept.

One of interesting mechanisms that the Ethereum smart contract created is the ICO crowd-funding methodology. A major part of smart contracts is written for ICO needs only. This is done by a small number of professionals and, most important, time is required to write applications, or migrate applications to blockchain technology, since this task requires capabilities of:

- Writing smart contracts (short term effort).
- Understanding application logic (long term application development).

As a result smart contract developers are overloaded since specialist must focus on short term efforts of writing smart contracts; which, by the way, is usually the most profitable part of a project (the most popular is ICO).

Of course, there other are smart contract-driven systems, that support other languages, like JS, C#. Many groups have attempted to bring a larger number of languages to support Virtual Machines for smart contracts. However, all these solutions are not complete and, of course, do not utilize the power of the community. This energy should move from enthusiasm over ICO to creating real products and engines!

Our goal is to make DApp development easier, reduce ICO costs and focus on application ecosystems, in cases where the application is required and has a real market. We believe that almost any application today that is available on mobile application markets and requires a server, has is reasonable place on a blockchain. Our goal is to develop the IoLite blockchain to create an inexpensive and effective ecosystem for DApps that will be easily integrated with FAE modules, where the same programmers/engineers who develop mainstream applications can write integration modules for blockchain.

Lets make blockchain easy and desirable for the application market!

We want to take the ICO idea of crowd-funding to create crowd-based efforts for developing/extending languages. We wish to motivate people to write new dictionary structures to better understand formal languages and thus machines.

Part of the IoLite vision is to make blockchain data valuable and reuse it, not only for block confirmation, but also as valuable knowledge. The FAE engine not only provides easy access from different languages to write smart contract, but actually creates a pool of vocabularies/solutions for **naturalized** interfaces.

Imagine a mobile developer attempting to create an application to manage a robot-controlled parking lot. She can go to the IoLite blockchain and check if such a robot naturalized interface exists (FAE), and if so, she can take the SDK and implement it. It is like a Google translator API. IoLite contributors will obviously want to extend such interfaces, based on market demand. Of course, academic centers will take advantage of such system as well.

IoLite is about bridging the gaps between human-beings and machines!

1.1. Smart contract technology gaps

There are several major blockers preventing the massive adaption of smart contract technology.

Use case 1: We want to add blockchain as additional market for our existing product

1. Our motivation:
   (a) Our business model is built on top of our data value
   (b) If keep data on blockchain and we dont need servers
   (c) We engage security and high availability from blockchain (done by community)
2. What are the issues:
   (a) We are developing our current product on JS, C/C++, PHP. We dont have Solidity developers
   (b) We can use other blockchains with C#, JS support, but still we don't know how to write smart contracts
   (c) We are using Microsoft Visual Studio. And we prefer to work with professional IDE
   (d) We want to use **unittest**[12] for writing smart contracts
   (e) Right now there are number of solutions, like: Embark[13], Truffle[14], Dapple[15], Populus[16]
   (f) But we have issues on how to use same language we used to develop an application to write unittest
3. Our conclusion:
   (a) We need to enable each programmer in a company to write smart contracts
   (b) We need to ensure that engineers will continue to use advanced IDE.
   (c) We need to certify that unittests will be written by programmers that developed code.

Use case 2: We want to use smart contracts for traditional contract law

1. Our motivation
   (a) Make our contract available to community, publishing
   (b) It is well secured by blockchain technology
   (c) Of course, there are security holes, but the entire community is demanding bug fixes and security solutions
2. What are the issues:
   (a) We need smart contract specialist to write the code; we do not have such in team
   (b) We need to test and verify requirements from legal experts and implement code. However, lawyers cannot read code
   (c) We need smart contract engineers who are both qualified programmers and lawyers. This is no easy task
3. Conclusion:
   (a) The best thing to do is get assistance from a lawyer and make it easier to define smart contracts
   (b) We want programmers to understand the written content
   (c) We want lawyers to be able to read final code in naturalized form so they can verify that requirements have been implemented

Of course the same requirements that we have in Use case 2 are relevant to other professional fields beyond law: science, marketing, medicine, etc. Thus, we require a smart translator to convert code back and forth between blockchain virtual machine code and various interfaces: legal documents, popular programming language, etc. In addition, we have the good fortune of having a full stack solution that could be used together with this technology, in such cases as

Android play market: publishing server, IDE, tests, Google tools, etc.

The Google team created a full ecosystem that has everything required by developers to create and manage applications on the Google market. Doing the same for blockchain technology is extremely difficult, and we focus on how our IoLite solution can be easily integrated in existing markets as well-defined and self-contained plugins in most popular IDE solutions.

1.2. Blockchain

IoLite is a blockchain technology and, as such, it faces all the challenges of such systems. IoLite is an innovative solution that is motivated by major achievements within the machine learning industry. The advanced FAE engine, keeps its data in blockchain and expands based upon collaboration algorithms. Our primary goal is to make our blockchain not only a financial platform, but to actually enrich it with valuable information about languages and user characteristics.

One of considerable efforts required during FAE development is to mine blocks to be used by AI algorithms, as in the work of Martn Abadi and David G. Anderse (Learning to protect communications with adversarial neural [17]). We want to maximize profit by aggregating knowledge in the IoLite blockchain. As a practical approach, the first blockchain code base would be derived from Ethereum EVM Solidity based code or EVM2 with eWASM code. Both of them have advantages and disadvantages, such as security, community support, technology popularity, exposure, etc.

We will enable EVM and EVM2 on our blockchain as testnets, and analyze the matureness of each solution. Ethereum, despite many issues raised by the community, is still the leader in DApp oriented solutions. So, focusing at the beginning on the FAE module, would create a community trust in blockchain and, of course, we would utilize Ethereum community efforts for security and platform effectiveness. As a win-win situation, the Ethereum team could use FAE modules to make smart contract easier for developers.

Of course, the main goal of IoLite to aggregate knowledge in blockchains and ecosystems would be changed to enforce AI algorithms. We are considering employing both **PoS**[18] and **PoW**[19] to produce valuable information about languages and users.

1.3. Economics

The IoLite team is determined to bridge the gaps that currently exist between the traditional industries and the crypto-world allowing traditional industries to benefit from using smart contract technology within their operational flows.

1.3.1. Insurance

The global insurance industry is valued at nearly $5 trillion, and insurance companies are at risk of losing a share of this valuable market to new entrants[20]. That is because these legacy players have been even slower to modernize than their counterparts in other financial services industries (see Figure 1.3.1).

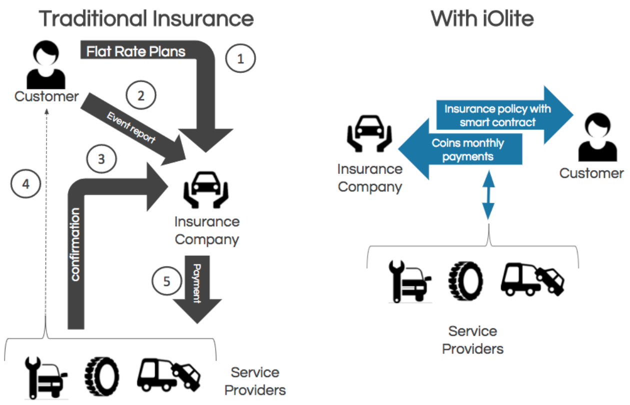


Figure 4.

1.3.2. Legal services

The Global Legal Services Market in 2016 was $584.4 billion[21] and there are more than 1.3 million lawyers in the United States.

The global online legal services market is $5 billion[22] with an annual Growth of 7.6%. There are about 9,000 online legal service businesses that are continuously searching to improve efficiency and reduce costs.

1.3.3. Trade Finance (Letters of Credit)

The average value of an L/C (Letter of Credit), the most common financial tool in trade finance, in 2016, was US$463,000. There were over 43 million L/Cs issued that year, which, as a percentage of monthly fees, represents a market of above $240 billion [23].

Operational overhead and various geo-related risks in its execution have caused banks to seek alternatives to the traditional L/C (see Figure 1.3.1).

1.3.4. Crowdfunding

The global Crowdfunding industry estimated fundraising volume in 2015 was $34 billion[24].

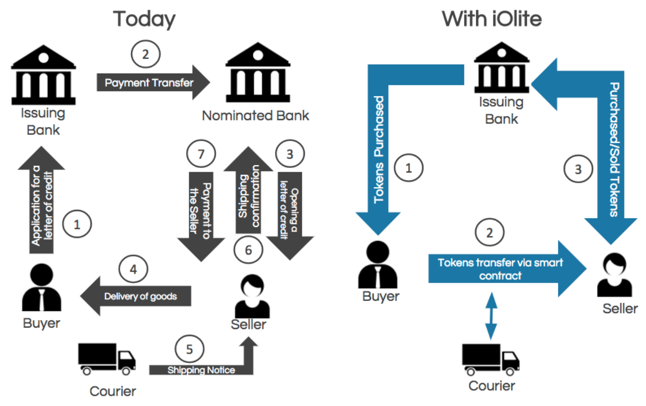


Figure 5.

The World Bank[25] predicts that crowdfunding investments will be a $96 billion-a-year-market in developing countries by 2025[26].

The biggest problem with crowdfunding is trust, an element which is not required when using smart contracts.

1.3.5. Supply Chain Management

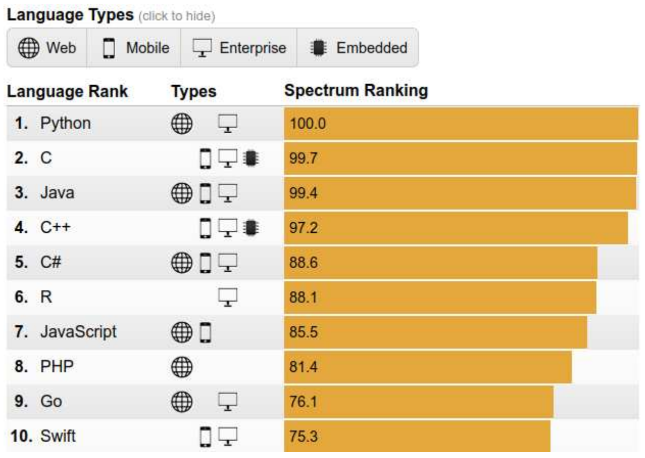
The global revenue generated through the supply chain management (SCM) software market in 2016 was worth US$11.2 billion worldwide, and is forecast to exceed $13 billion in 2017[27].

2. Fast Adaptation Engine (FAE)

The IoLite Fast Adaptation Engine (FAE) is based on the work done by Sida I. Wang, Samuel Ginn, Percy Liang, Christopher D. Manning as presented in their paper Naturalizing a Programming Language via Interactive Learning[28]. Existing natural language interfaces, as general solutions, are quite primitive and can be compared to programming languages. It is a very exciting field with many researches trying to find the most effective way to instruct engines using natural languages in fields such as operating mobile phones, manage robots, etc.

The question is, how is this related to our smart contract domain problem. As we discussed in Use Case 2, we want to be able to support non-programmer players, such as lawyers, to create smart contracts. This is a trivial use case for FAE as a natural language adjustment. But lets review Use Case 1 as well. As shown in Figure 6, there are a number of popular languages and the number of potential smart contract developers in these fields is sufficiently large to consider enabling this languages for writing smart contracts. How easy is it to adapt these languages for writing smart contracts? Consider EVM2 eWASM as a WASM model which was initially proposed to support compiled languages like C/C++. This appears to be a potential solution. However to solve this problem as a low-level bytecode format is no simple task.

Lets start with Python. Since Python is a dynamically typed language, this would only be possible if the assembly program would use a runtime environment. To make this happen we need to use modules such as PyPy, which has very restricted form. Also, languages that support lambda calculus[29] or automatic memory management cannot be compiled directly to assembly.



Language Types (click to hide)

🌐 Web | 📱 Mobile | 🖥 Enterprise | ▪ Embedded

Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	99.4
4. C++	📱 🖥 ▪	97.2
5. C#	🌐 📱 🖥	88.6
6. R	🖥	88.1
7. JavaScript	🌐 📱	85.5
8. PHP	🌐	81.4
9. Go	🌐 🖥	76.1
10. Swift	📱 🖥	75.3

Figure 6. Top 10 popular languages by spectrum.ieee [30]

So what about Java? Java is normally compiled to **bytecodes**[31] and the bytecodes are usually executed using a Java Virtual Machine. One way to run them on the EVM would be to write a Java Virtual Machine that ran on the EVM, or another target VM. Basically, it is very clear that making converters, if it possible at all, is very expensive and requires a major effort. The discussion about a business model approach is beyond the scope in this document, but it is definitely not simple, otherwise we would have implemented such solutions already.

In FAE our method is very different from attempts to create general solutions for converting languages. The idea of FAE is that we create, for each language, a core language that clearly defines how language syntax would be converted to a smart contract execution language such as Solidity or eWASM JS with restrictions for smart contract requirements. Then the community would expand this language incrementally by defining alternatives.

The work of Sida et al. [28] shows that user can naturalize the core language incrementally by defining alternative, more natural syntax and increasingly complex concepts in terms of compositions of simpler ones. In their work, they define Voxelurn (and we suggest adapting this code[32], [33]).

All users share one language starting from DAL (Dependency-based Action Language). The language composes actions using expressive control primitives such as if, for each, repeat, etc. Actions take sets as arguments, which are represented using lambda dependency-based compositional seman-tic (lambda DCS) expressions.

The goal of the user is to build a structure in Voxelurn. Voxelurn uses definitions, in addition to selecting candidates, to serve as the supervision signal. Each definition consists of a head utterance and a body, which is a sequence of utterances that the system understands. Allowing ambiguity and a flexible syntax is a key reason why natural language is easier to produce. This work demonstrates that interactive definitions is a strong and usable form of supervision.

The following is from the paper on Voxelurn:

> Unlike function definitions in programming languages, the user writes concrete values rather than explicitly declaring arguments. The system automatically extracts arguments and learns to produce the correct generalizations. For this, we propose a grammar induction algorithm tailored to the learning from definitions setting. Compared to standard machine learning, say from demonstrations, definitions provide a much more powerful learning signal

This work definitely provides a very strong motivation to strive to **naturalize** code bases for lawyers. As a first step our FAE will derive code from this work and we will define DAL for smart contracts.

For each field we will need to have a unique DAL: C, C++, Java, JS, legal documents, etc. Of course, part of the development process would be in the analysis of how much correlation there is between different DALs.

We know that, in the end, all of our languages need to produce code for smart contract EVM, and the rule base is the same for all of them. By design we are capable of translating from one language to another, so we can share the same code between different users using different languages (including spoken language), if such languages are supported by the system.

We are considering maintaining mapped data with original code in blockchain as well as being able to make better adjustment when a smart contract is translated back to one of the languages.

2.1. Dependency-based Action Language (DAL)

Each supported language is created with an understanding of the core language DAL (Dependency-based Action Language). It is part of the IoLite foundation teamwork. Each language subset will support:

- Converting to smart contract language schema
- Unittests blocks
- Core base code
- Staging code (new proposals from the community)

It is important to point out that plugins released from the community will have a strong supervising model. Only users **certified** by foundation blocks can make contributions to official releases. The code itself would be open source and any group will be able to create its own domain of FAE, however the IoLite foundation will not support it or take any responsibility for it.

2.2. Learning interactively

In IoLite we have two types of users: contributors and users. Contributors will build structures in FAE. For each structure that would be used, the contributor would be advanced by a system token. Used here, it implies that the system would publish smart contracts that employed the contributors structures. Each structure has an explicit contributors signature. The price to use code is always lower than the contract execution price, so the system is protected from contributors who desire to use the system to mine the blockchain. Furthermore, the IoLite foundation will run various algorithms to detect anomalies that could be created by bad contributors trying to hack the system.

To return to the learning process, contributors would be advanced if their code was used both by rank and tokens. Contributors who would like to be involved in developing foundation plugins, would have to be certified and would need to pass a **KYC**[34] process to contribute code to a staging tree. When a staging module has sufficient rating, the foundation will consider promoting both contributor and structures to base code. Thus, each contributor has the potential of being a part of the foundation team.

There is another issue that must be mentioned. Any user using the system in a malicious way would be identified, and if she passed KYC, she would not be permitted to participate in foundation plugins. All structures would be dropped from the codebase, and plugins would mark usage of this code as unsafe. From the system perspective, when a user writes code and base collection cannot solve the syntax, it will perform a look-up in the staging collection (code that is provided by contributors).

The part of code that was parsed well with the base collection would be marked correct (of course the application itself must pass unittests collection, but syntax would be defined as correct). If user code will not solve the text, it will perform a look-up in the staging code (code that is provided by contributors) and it will bring the solution from the top 10 contributors. Of course, this means that a new contributor should provide a solution for syntax that doesnt have proposals from staging. To establish a protection mechanism, each contribution would be marked with a blockchain timestamp, so more advanced contributors will not copy/paste the code and take advantage of their positions as more ranked contributors.

It is quite clear that only the staging part requires an algorithm for automatic processing. The approval of a structure and its addition to a base is done by an IoLite foundation engineer.

We use Model and learning and Grammar induction as they are employed in the Voxelurn paper. When we talk about learning programming languages, the situation, of course, is much easier since language itself is formal language and for programmers as contributors or foundation engineers, it is much easier to agree on new structures. For lawyers, it would be done as research with a lot of potential as we learned from the Voxelurn effort:

> As the system learns, users increasingly prefer to use the naturalized language over the core language: 85.9% of the last 10K accepted utterances are in the naturalized language.

2.3. Iolite users

IoLite users would be programmers who wish to write new smart contracts.

Our first goal is to provide plugins for Eclipse and Visual Studio IDE. The IoLite foundation will provide supported languages in the IoLite plugin. Each new language will be supported by both Visual Studio and Eclipse (of course specific versions would be selected).

For Visual studio, IoLite will use Visual Studio Extensions[35].

For Eclipse, eclipse platform plugin development would be engaged [36].

Both IDE products have strong parsing capabilities, however we will only need to read raw text and use IDE control features. Our FAE will use advanced Voxelurn parsing techniques.

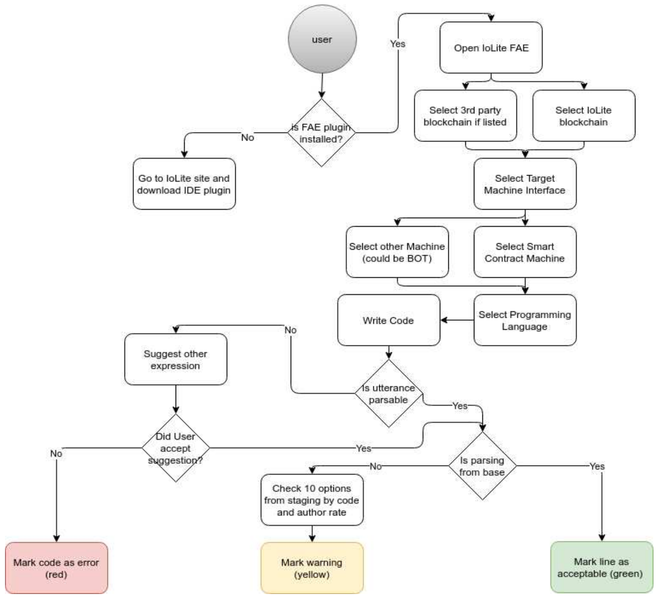

user
Is FAE plugin installed?
No
Yes
Go to IoLite site and download IDE plugin
Open IoLite FAE
Select 3rd party blockchain if listed
Select IoLite blockchain
Select Target Machine Interface
Select other Machine (could be BOT)
Select Smart Contract Machine
Select Programming Language
Write Code
Is utterance parsable
No
Yes
Suggest other expression
Did User accept suggestion?
No
Yes
Is parsing from base
No
Yes
Check 10 options from staging by code and author rate
Mark code as error (red)
Mark warning (yellow)
Mark line as acceptable (green)

Figure 7. User new code scenario diagram

2.3.1. User scenario:

1. User will need to install the IoLite plugin.
2. The IoLite plugin will have a help file that will explain smart contract principle and it will be unique for all languages provided by IoLite.
3. Users will have option to choose the IoLite system.
4. In IDE UI controls, a user will select the programming language she desires to use for writing smart contracts (we will provide an option for NLP languages which will be marked as NLP[37]).
5. After a user has selected a language, a controls panel and a debugging window will be provided.
6. Both windows for code writing and debugging will present correct syntax, but the debugging window will also present information about simulation of smart contracts and unittesing
7. Structures (parsing components) that are part of the basecode of the language will be marked in

green (the color here represents the level of trust), staging structures will be yellow and author information will be available on line. If a parser fails, it will instruct the user that this part of the code failed and no alternatives were found.

For more details check Figure 7 Algorithm 1

If user code cannot be parsed, the system will propose a solution based on a learning system. If user agrees, he must become a contributor (refer to Section 2.4). When a user completes a smart contract, she will be able to simulate publishing and identify:

1. Cost of the smart contract, including fees for code contributors (fees would be very low).
2. Simulate a smart contract triggers to check if code is correct. The user can create a list of events used by the smart contract and check results. Simulation cost is zero, only publishing will require fees.

Algorithm 1 Intelligence parsing thread

```
Function ParsingBase(data utterance)
  if utterance does not parse then
    return []
  else
    return [$utterance$]
  end if
Function ParsingStage(data utterance)
  if utterance does not parse or user rejected all proposed solutions then
    return []
  else
    return [$solutionSelectedByUser$]
  end if
Function ParsingThread()
  repeat
    if $textChainged$ is $true$ then
        repeat
            utterance = get utterance from changed text
            if $ParsingBase(uttearance)$ is not empty then
                mark utterance green color
            else if $ParsingStage(uttearance)$ is not empty then
                mark utterance yellow color
            else
                mark utterance red color
            end if
            update text with utterance
        until $textChanged$ is $true$
    end if
  until $running$ is $true$
```

2.4. Iolite contributors

The heart of the IoLite blockchain is its contributors. An Iolite contributor is a major force in our system. The main focus of the IoLite foundation is to provide contributors with a professional level of support and motivation to make the FAE learning curve effective. The goal of contributors is to create new structures and expand the languages supported in our system. For each successful structure, a contributor may receive a reward. In the IoLite system, a contributor will not have immediate benefits for writing code, but each successful effort will contribute to long term benefits, since use by user structures guarantees rewards. Of course, during participation in expanding languages, contributors must study how to find and identify real needs from customers and provide valuable solutions. We believe that experienced contributors will be a major source of NLP modules, as creating new structures is fairly advanced but also a very exciting process (see motivation from Sida et al. [28]).

2.4.1. contributor scenario

1. Contributor identified missing structures
   (a) Based on Iolite forums
   (b) By personal experience with release DAL for specific language
2. A contributor writes text that includes missing syntax.
3. The System identifies utterances that cannot be resolved. It will try to provide candidates, based on the current knowledge base, and a contributor can select the best option and potentially register for a very low fee since the system identifies alternatives.
   (a) If an utterance is unparsable and the contributor rejects all alternatives, she would be asked to define the body. Any utterance that is not yet understood would be defined recursively.
   (b) Interactivity is used to control the exploding ambiguity.
4. The moment system parsing accepts the solution, the contributor will be asked to write unittest code (a use case related to the smart contract).
5. The unittest and new structures would be signed by the user and the package would move to a staging tree.

6. Each time a new smart contract has been published in blockchain which has used contributor structures, contributors will be rewarded and their structure rank will increase.

7. If the structure rank passes the threshold, it will be submitted for review by a foundation engineer.

8. If it is approved in review, new structures would be integrated in the base tree.

2.4.2. Contributor account creation:

1. If the contributor is new, the system will generate tasks for her, the moment her account is created.

2. The user will be required to write a smart contract that will trigger events that enable the next step in registration.

3. After a user has passed the smart contract test (only engineers who understand smart contracts can be contributors), the user would be required to pass a KYC process.

4. After the contributor has passed the KYC process, she can write new code for a staging tree.

5. If a contributor was compromised, she would be black listed in blockchain.

6. A contributor who provides structures that passes review by foundation engineers, receives a proposal to join the foundation.

3. Blockchain

Blockchain is a critical records system for IoLite.

Our motivation is to change it from just keeping records of transactions to providing very valuable storage. In IoLite, the main goal is to use blockchain as a decentralized knowledge base of FAE structures.

FAE structures are blocks of information that help semantic parsers to understand **naturalized** language to operate machines. The first machine that the IoLite team is targeting to implement is a smart contract generator. Of course, any formal language for a machine is a target for the FAE engine.

Blockchain technology is critical to generate high quality databases, and algorithms of collaboration are major parts of our blockchain.

One of the first steps our group will implement is adding logic on top of Ethereum technology to minimize risks of security and add system capabilities to accept new smart contracts and transactions generally. During FAE development other options such as **DAG** [38] would be reviewed.

3.1. Blockchain collaboration method

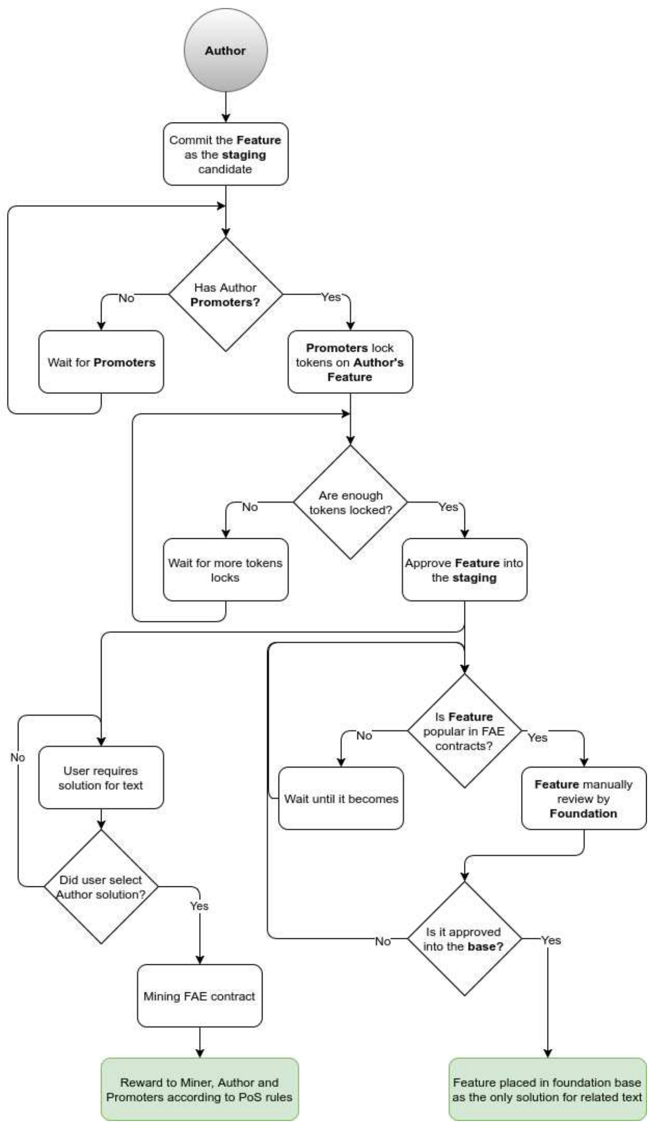

Author
Commit the Feature as the staging candidate
Has Author Promoters?
No
Yes
Wait for Promoters
Promoters lock tokens on Author's Feature
Are enough tokens locked?
No
Yes
Wait for more tokens locks
Approve Feature into the staging
Is Feature popular in FAE contracts?
No
Yes
User requires solution for text
Wait until it becomes
Feature manually review by Foundation
No
Did user select Author solution?
Yes
Is it approved into the base?
No
Yes
Mining FAE contract
Reward to Miner, Author and Promoters according to PoS rules
Feature placed in foundation base as the only solution for related text

Figure 8. Collaboration algorithm diagram

1. Someone puts forward a Proposal. Lets call him the Author. The Author informs the community about the desire to implement a Feature.

2. The Author implements the Feature in the staging segment.

3. Users of the network lock their tokens referring to the Authors profile and his Feature Proposal. Lets call them Promoters.

4. The Authors Rank depends on the total number of locked tokens on his profile.

5. A Promoter can, at any time, revoke the locking of her tokens or lock more.

6. The Foundation gives attention to the Author when a large number of tokens has been locked for a long period on her profile. Boundary conditions adopt to changes in the network. The Foundation gives attention to the Authors Proposal when enough tokens are locked on the Authors profile with reference to this Proposal. This Proposal becomes a candidate to enter the basis. The Foundation carries out a manual moderation.

7. When the contract is mined, part of the commission goes to all Authors whose Features were used in the source code of the contract. This condition is valid only for solutions in the basis.

8. The reward received by the Author is divided between him and the Promoters according to PoS rules. Problem: If the Author has many Promoters, each of them will receive a minor reward. Solution: In fact, this is good reason to look for new talented Authors and become their Promoters raising their rank.

3.2. Problems & Solutions

1. How to protect the system from self-voting? The attacker must keep a large number of tokens on different accounts to carry out this type of attack. Also, he may lose, even if the attack is successful, since his Feature is likely not to satisfy the users needs and will not become popular.

2. How to attract authors to work only on unimplemented features? The implementation of an already solved feature will not be accepted by the community. The Author risks disappointing his promoters and may lose some of their contributions.

3. How to prevent ideas developed by young authors from being stolen? Since this is a collaborative system, young authors can draw the communitys attention to the theft of an idea. It is not difficult to do this, since there is a timestamp providing the features creation in the blockchain. Thus, the thief risks losing trust and support.

4. What is the motivation for users to promote Authors? The Author divides the reward between all promoters according to PoS rules.

5. What is the motivation for the miners? Miners receives a reward for the block according to PoW rules (static reward + commission).

6. What happens if the Author loses the keys? In the context of blockchain technology it means the end. But in the IoLite system the Foundation can link the new profile with the old profile if no other transactions are made from its address. The solution is only valid for users whose features are accepted into the basis.

3.3. Blockchain economics

In our work, we derive ideas about ecosystems from the Ethereum foundation. And it is quite clear that Ethereums economic principals have strong bases.

The main goal of IoLite is to bring mainstream developers to blockchain technology and specifically to smart contract principals. The IoLite team wish to create strong machine learning methods to bridge the gaps, and to bring maximal value from any work done in our ecosystem.

Thus, our next candidate, after FAE, is AI as bases for proof of algorithms. The motivation is clear; we would like the system to study and generate information of value for humanity.

We will not presently discuss our plans for AI principles of PoW, except to mention that we are studying the work of Martin Abadi and David G. Andersen (Learning to protect communications with adversarial neural cryptography) But, to return to the economic model, As Vitalik explained in his article On Inflation, Transaction Fees and Cryptocurrency Monetary Policy [39]:

> ”The primary expense that must be paid by a blockchain is that of security. The blockchain must pay miners or validators to economically participate in its consensus protocol, whether proof of work or proof of stake, and this inevitably incurs some cost. There are two ways to pay for this cost: inflation and transaction fees. Currently, Bitcoin and Ethereum, the two leading proof-of-work blockchains, both use high levels of inflation to pay for security”

Simpler explanations are provided by Joseph Lubin, The Issuance Model in Ethereum[40] and Easy Ethereum[41].

Also, a major goal of blockchain is to find a model where only transaction fees would be the energy for the system. In IoLite we still aim to support miners and enhance using AI algorithms as PoW to solve different problems.

The current inflation model of Ethereum satisfies needs. Since, as the total number of Ether increases with the rate of issuance held constant, the inflation rate is expected to decrease over time. In 2017 the issuance rate of Ether is 14.75% [42].

About transaction fees, see Figure 9.

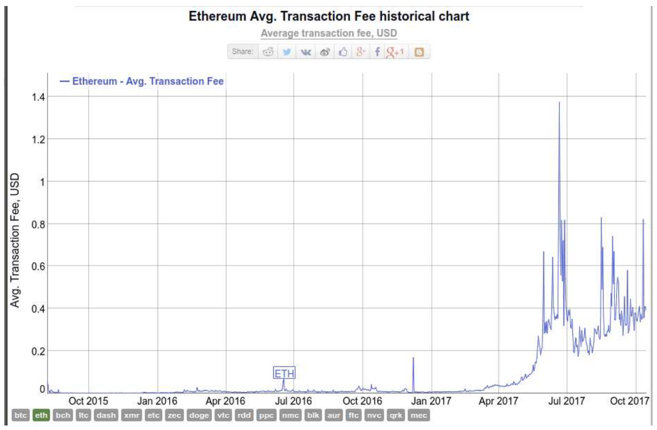


Figure 9. Ethereum Avg. Transaction Fee historical chart [43]

While gas is fixed per operation, the amount a user pays for gas (gas price) is dynamic and is dictated by market conditions. The price of gas is a value representing how much Ether the user is willing to pay per gas. When a user sends a transaction, they specify the gas price in

$$Gwei/Gas$$

$$1Gwei = 0.000000001ETH$$

and the total fee that they pay is equal to

$$gasprice * gasused$$

.

Miners are paid this fee and they prioritize transactions with a higher gas price. The higher the price of gas you are willing to pay, the faster your transaction will be processed.

In IoLite, we added additional fees that need to be paid to contributors. The calculation is about 5% of overall fees, and would be distributed between contributors based on the number of structures that were used in the contract.

One of IoLites targets is to make fees as low as possible. Of course, this can be done only if market competition would be strong enough. As part of the effort, PoW AI algorithms would receive fees based on usage of information that was generated by the miner(smart contract automation triggers).

A more in-depth explanation of AI and PoW will be available as an IoLite white paper (phase II, after FAE).

4. Related work and discussion

IoLites technological goal is to bridge the gaps. We are motivated by recent work in the realm of machine learning and specifically, interactive learning with advanced semantic parsing algorithms.

The work is derived from research and available code in Sida et al. [28]. This work extends previous studied research from Sida I. Wang Percy Liang Christopher D. Manning (Learning Language Games through Interaction [44]).

To understand better how Voxelurn works, one must study the SHRDLURN research project. SHRDLURN games present an interesting approach to interactive learning through language games. The computer knows nothing at the beginning but, through interaction with a player it learns the settings. This work provides interesting details about human strategies and how advanced players dramatically accelerate machine learning. One of the serious obstacles in this work is the interface itself, where the engine provides possible interpretations after the parsing utterance. The list of all possibilities makes selection very difficult and, in many cases, impossible.

However in Voxelurn there is an advanced option to define the body and recursively define all needed utterances and it is actually what pushed us to the idea of FAE. Of course, the amazing grammar induction method, with limited options to select possible answers after parsing, makes such a solution well defined for FAE needs. In FAE we learn the programmer language and advanced lambda DCS makes the process easier. We want to take full advantage of Voxelurn to support NLP, to naturalize language for lawyers in smart contracts or other human players managing machines.

Making smart contract writing a simple process and exposing it for mainstream market developers will increase demand and dramatically improve such technology.

Our blockchain can be defined as a knowledge base for naturalized languages that can operate machines. This is our vision and primarily goal!

References

[1] https://en.wikipedia.org/wiki/Blockchain.

[2] https://ethereum.org/.

[3] https://en.wikipedia.org/wiki/Smart_contract.

[4] https://en.wikipedia.org/wiki/Initial_coin_offering.

[5] https://en.wikipedia.org/wiki/Crowdfunding.

[6] https://en.wikipedia.org/wiki/Integrated_development_environment.

[7] https://www.stanford.edu/.

[8] https://coinmarketcap.com/charts/.

[9] https://www2.deloitte.com/uk/en/pages/innovation/solutions/deloitte-blockchain-practice.html.

[10] https://coinmarketcap.com/.

[11] https://en.wikipedia.org/wiki/Solidity.

[12] https://en.wikipedia.org/wiki/Unit_biblrefing.

[13] https://iurimatias.github.io/embark-framework/.

[14] https://github.com/ConsenSys/truffle.

[15] https://github.com/NexusDevelopment/dapple.

[16] http://populus.readthedocs.org/en/labiblref/.

[17] https://arxiv.org/pdf/1610.06918v1.pdf.

[18] https://en.wikipedia.org/wiki/Proof-of-stake.

[19] https://en.wikipedia.org/wiki/Proof-of-work_system.

[20] http://www.businessinsider.com/axa-turns-to-smart-contracts-for-flight-delay-insurance-2017-9.

[21] http://www.globallegalpost.com/big-stories/global-legal-services-market-slows-in-2016-17136055/.

[22] https://www.ibisworld.com/industry-trends/specialized-market-research-reports/advisory-financial-services/legal/online-legal-services.html.

[23] https://cdn.iccwbo.org/content/uploads/sites/3/2017/06/2017-rethinking-trade-finance.pdf.

[24] http://crowdexpert.com/crowdfunding-industry-statistics/.

[25] http://www.infodev.org/infodev-files/wb_crowdfundingreport-v12.pdf.

[26] https://www.forbes.com/sites/adigaskell/2016/03/15/the-rise-of-investment-crowdfunding.

[27] https://www.statista.com/statistics/271214/global-supply-chain-management-software-market-revenue/.

[28] https://nlp.stanford.edu/pubs/wang2017naturalizing.pdf.

[29] https://en.wikipedia.org/wiki/Lambda_calculus.

[30] https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages.

[31] https://en.wikipedia.org/wiki/Bytecode.

[32] http://www.voxelurn.com.

[33] https://github.com/sidaw/sempre-interactive.

[34] https://en.wikipedia.org/wiki/Know_your_customer.

[35] https://msdn.microsoft.com/en-us/library/bb166030.aspx.

[36] https://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Ffirstplugin.htm.

[37] https://en.wikipedia.org/wiki/Natural_language_processing.

[38] https://en.wikipedia.org/wiki/Directed_acyclic_graph.

[39] https://blog.ethereum.org/2016/07/27/inflation-transaction-fees-cryptocurrency-monetary-policy/.

[40] https://blog.ethereum.org/2014/04/10/the-issuance-model-in-ethereum.

[41] https://www.easyeth.com/economics-of-ethereum.html.

[42] https://docs.google.com/spreadsheets/d/150B9eytmjZ642tYD0jSdFZQHldmk7VG5Wm3KVctydpY/pubhtml.

[43] https://bitinfocharts.com/comparison/ethereum-transactionfees.html.

[44] https://nlp.stanford.edu/pubs/wang2016games.pdf.