Spring 5-20-2019

CONTRACT BUILDER ETHEREUM APPLICATION

Colin M. Fowler
San Jose State University

CONTRACT BUILDER ETHEREUM APPLICATION

A Project Presented to Dr. Robert Chun and

The Faculty of the Department of Computer Science

At San Jose State University

In partial fulfillment of the

requirements for the class

CS298

By

Colin M. Fowler

Spring 2019

The Designated Project Committee Approves the Project Titled

CONTRACT BUILDER ETHEREUM APPLICATION

By

Colin M. Fowler

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Spring 2019

Dr. Robert Chun, Department of Computer Science

Dr. Philip Heller, Department of Computer Science

Brian Coleman, Perkins Coie LLP

TABLE OF CONTENTS

ABSTRACT

Developments in Blockchain, smart contract, and decentralized application ("dApps") technology have enabled new types of software that can improve efficiency within law firms by increasing speed at which attorneys may draft and execute contracts. Smart contracts and dApps are self-executing software that reside on a blockchain. Custom smart contracts can be built in a modular manner in order to emulate contracts that are commonly generated and executed in law firms. Such contracts include those for the transfer of services, goods, and title. This article explores exactly how implementations of smart contracts for law firms may look.

TABLE OF FIGURES

I. INTRODUCTION

Lawyers are expensive; I know, I am one.[1] Lawyers typically bill every six minutes. Reducing the time a given task requires in a firm makes the firm's services more attractive to clients and enables attorneys to increase the number of clients they may serve. There is a growing desire in the legal field to reduce the time attorneys spend performing repetitive tasks and focus time instead on advising and counseling. One area that remains time intensive, though still repetitive is the drafting, revising and consideration of contract language. Care needs to be taken in the choice of wording (in human languages) because variable interpretation of contracts leads to a significant amount of court litigation every year.

Unfortunately, contracts have been difficult to automate or streamline because of the variability included in their generation and the trust required to convert them to program code. While program code is not prone to interpretation like English, or other human languages, program code must be run on someone's computer, and thus private execution of program code strains trust required for a contract negotiated at arm's length.

Cryptocurrencies have opened up new space for software innovation. Starting with Bitcoin in 2011, numerous cryptocurrencies have iterated on the general concept of a decentralized immutable public ledger, also known as a "blockchain." Each cryptocurrency operates on its own respective blockchain. The Bitcoin blockchain is largely intended as a means to track movement of digital currency. Within a short period, additional uses became apparent as users realized that documents could be inserted into transactions of Bitcoin. Storage of documents on the blockchain enabled users to prove that they had a given set of

[1] https://www.perkinscoie.com/en/professionals/colin-m-fowler.html

information (whatever was stored on the document) at a particular date and time. Other cryptographic currencies have iterated and expanded on the utility.

Recently, the Ethereum cryptocurrency has enabled on-blockchain processing using virtual machines as a type of public computer. Nodes that support the Ethereum blockchain provide unused processing power to execute open source software that is stored on the blockchain. This on-blockchain software is referred to as a smart contract or a decentralized application ("dApp"). Smart contracts are programmed to execute upon the satisfaction of one or more predetermined conditions. Smart contracts executing on a blockchain can be trustless and are secured by the immutable nature of the blockchain. As a result that Smart contracts are executed publicly, the requirement of additional trust for arm's length parties of a contract created in program code is removed.

The legal industry can make direct use of smart contracts to improve efficiency and reduce litigation.[2] Litigation is often a result of ambiguity in contracts and implementing such contracts in program code that follows strict rules for execution reduces ambiguity and may in turn reduce litigation. Streamlining administrative issues enables lawyers to direct greater focus to legal analysis of problems. Lawyers will transition from drafting traditional contracts to building standardized smart contract templates. The templates would use code modules that would plug in as necessary/selected. In appearance, the user interface would look similar to the standardized traditional contracts that one might find on LegalZoom (a do-it-yourself legal help service). In some cases, the old contracts will evolve into a hybrid of paper and digital content where contracts are verified via blockchain and substantiated by physical copy.

[2] Rouse, Margaret, "Smart Contract", last updated on April 2018, (available at http://searchcompliance.techtarget.com/definition/smart-contract and accessed 3/29/2019).

This research explores exactly how implementations of smart contracts for law firms may look. As noted, the researcher is, himself, a patent attorney with significant experience working in a large law firm environment (commonly referred to as “BigLaw”) and has billed hundreds of hours drafting and reviewing contracts.

II. BACKGROUND

A. THE BLOCKCHAIN

Cryptocurrency networks operate on a distributed network architecture. Key to understanding cryptocurrency is the data structure upon which the entire network operates. The Bitcoin and Ethereum networks use what is referred to as a Blockchain.

The Blockchain includes a history of all transactions that have ever occurred on the network. Each full node in the distributed network holds a full copy of the Blockchain. To participate in the network at all, the Blockchain history must be consistent with the history of at least a majority of other nodes. This consistency rule has an important effect of causing the Blockchain to be immutable. In order to effectively attack a Blockchain, one must control 51%+ of the processing power of the entire network. Where the network is comprised of thousands of nodes, assembling the requisite 51% is exceedingly difficult.[3]

When a given node intends to generate a transaction, the transaction is propagated throughout the nodes until it reaches a node or group of nodes that can assemble that transaction and other transactions generated during a contemporaneous period of time into a block. The nodes that assemble blocks based on a collection of transactions are referred to as "miners." When blocks are generated by miners, the miners receive a predetermined number of coins (in whichever currency's blockchain the miner is operating on). The term "miner" originates from the idea that these users/machines are performing work to obtain

[3] While it is true that many nodes often group together in pools that together work together to solve for nounces to propagate the Blockchain, the grouped nodes of the pool do not necessarily share common control. While they have agreed to pay any mined coins to a central pot that is shared amongst the pool, this is far and away from agreeing to make changes to the Blockchain.

valuable resources. Until a transaction appears in a block it is not published or made public. Often a transaction isn't considered confirmed until 6 additional blocks have been added.[4]

At the time of writing this article, Bitcoin blocks are limited to the size of 1 MB and are generated approximately every ten to fifteen minutes.[5] This illustrates an important limitation of the Bitcoin network: that it only processes approximately 7 transactions per second. Conversely, Ethereum limits block size based on the amount of processing the contracts in the given block call for and blocks (including processed operations) are appended every five to fifteen seconds.[6] While cryptocurrency networks technically begin processing transactions in real-time, and the existence of a block including a given transaction verifies that transaction's authenticity, until that block is published to the Blockchain, the transaction is not verified.

This introduces the issue within the Bitcoin network at a given moment of "who has the money." During the ten to fifteen-minute span between block generation transactions that have been submitted may not actually process. This would occur when a user spends money they didn't have, or double spends. This is not to say the network has no verification mechanism between blocks. For example, when a given user attempts to pay another user, the system may easily query older blocks to inspect the given user's balance as of at least the most recently published block. If the given user has sufficient funds, it is moderately safe to trust the transaction.

However, if the given user is attempting to double spend all of their money, only one of those transactions will publish in the next block. The other will be rejected (which is

[4] Bitcoin Wiki "Confirmation" last updated March 16, 2018, available at https://en.bitcoin.it/wiki/Confirmation and accessed on 4/14/2018).
[5] Block Explorer available at https://blockchain.info/ and accessed 3/25/2018.
[6] Etherscan available at https://etherscan.io/ accessed 3/24/2019.

rejected and which processes is subject of a race condition and not necessarily dependent on time of generation). When discussing trivial amounts of money (e.g., paying for coffee), this is not really a big concern, but when handling larger purchases that occur quickly (e.g. stock in a company), the amounts can become significantly greater, and a clearance time of ten to fifteen minutes is not ideal.

Thus far, Bitcoin has been discussed as a network for trading Bitcoins. However, Bitcoin transactions have additional utility in that they can embed additional data. As contemplated above, Bitcoin can be used to purchase and record stock purchases. This is performed by including hashed data within an output field of a given transaction. In this manner, the proof of existence for any document or recorded data may be embedded into the immutable history of the Blockchain. In this manner, nearly anything may be traded via the use of an immutable cryptocurrency public ledger.

Systems that utilize the Bitcoin blockchain to transfer the ownership of non-coin assets require software that is separate from, and merely relies upon the immutability of, the Blockchain. The separate software is not necessarily secure or immutable itself. This extra-blockchain software is thus an inherent weak point in a system that relies upon the immutability of the blockchain to ensure security. Ethereum takes the ability to buy and sell non-coin assets a step further.

Ethereum smart contracts are in effect software that runs on the Blockchain. That software is open source and subject to inputs that are related to the Blockchain itself. Of course, one can still write code including vulnerabilities, but the platform enables greater security and fewer weak links in the chain.

B. SMART CONTRACTS AND DAPPS

Smart contracts and dApps execute on an Ethereum virtual machine ("EVM"). The EVM is instantiated on available network nodes. Smart contracts and dApps are applications that execute; thus, the processing power to do so must come from hardware somewhere. Nodes must volunteer their processors to execute these operations based on the premise of being paid for the work in Etheruem coins, referred to as Ether, measured in "gas". Gas is the name for a unit of work in the EVM. The price of gas can vary, often because the price of Ether varies,[7] and is specified within the smart contract/dApp.

Every operation that can be performed by a transaction or contract on the Ethereum platform costs a certain number of gas, with operations that require more computational resources costing more gas than operations that require few computational resources. For example, a multiplication instruction requires 5 gas, whereas an addition instruction requires 3 gas. Conversely, more complex instructions, such as a Keccak256 cryptographic hash requires 30 initial gas and 6 additional gas for every 256 bits of data hashed.

The purpose of gas is pay for the processing power of the network on execution of smart contracts at a reasonably steady rate. That there is a cost at all ensures that the work/processing being performed is useful and valuable to someone. Thus, the Ethereum strategy differs from the Bitcoin transaction fee, which is only dependent on the size in kilobytes of a transaction. As a result that Ethereum's gas costs are rooted in computations, even a short segment of code can result in a significant amount of processing performed. The use of gas further incentivizes coders to generate efficient smart contracts/algorithms.

[7] At the time of writing this, the price of Ether has varied by a factor of greater than 100 in the previous calendar year.

Otherwise the cost of execution may spiral out of control. Unrestricted, an exponential function may bankrupt a given user.

While operations in the Ethereum virtual machine (EVM) have a gas cost, gas has a "gas price" measured in Ether. Transactions specify a given gas price in ether for each unit of gas. The fixing of price by transaction enables the market to decide the relationship between the price of ether and the cost of computing operations (as measured in gas). The total fee paid by a transaction is the gas used multiplied by gas price.

If a given transaction offers very little in terms of a gas price, that transaction will have low priority on the network. In some cases, the network miners may place a threshold on the gas price each is willing to execute/process for. If a given transaction is below that threshold for all miners, the process will never execute. Where a transaction does not include enough ether attached (e.g., because the transaction results in so much computational work that the gas costs exceed the attached ether) the used gas is still provided to the miners. When the gas runs out, the miner will stop processing the transaction, revert changes made, and append to the blockchain with a "failed transaction." Failed transactions may occur because the miners do not directly evaluate smart contracts for efficiency. Miners will merely execute code with an appropriate gas price attached. Whether the code executes to completion or stalls out due to excessive computational complexity is of no matter to the miner.

Where a high gas price is attached to a transaction, the transaction will be given priority. Miners will process transactions in order of economic value. Priority on the Ethereum blockchain works similarly as with the Bitcoin blockchain. Where a user attaches more ether to a given transaction than necessary, the excess amount is refunded back to that user after the transaction is executed/processed. Miners only charge for the work that is

performed. A useful analogy regarding gas costs and price is that the gas price is similar to an hourly wage for the miner, whereas the gas cost is like a timesheet of work performed.[8]

[8] Coleman, Jeff, question answer on Stack Exchange, "What is meant by the term 'gas'?" last edited on May 31, 2017 (available at https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas and accessed on 3/24/2018).

III. MOTIVATION

There are a lot of different subjects that people and corporate entities use contracts to resolve. The relative cost of these sorts of contracts vary widely. Some are routine transfers that are generated and processed entirely by paralegals whereas others require a delicate selection of terms that delineate rights and processes of a transfer. Fundamentally, every contract is a transfer. Two or more sides bring some value (referred to as "consideration") to the table, and the consideration[9] is exchanged.

The aim of this article is to delineate a means of streamlining the drafting, review, interpretation, and execution of contract. To do so, I discuss a number of considerations that need to be made.

A. COMMON CONTRACTS

In a law firm most contracts generated are form based or "boilerplate". Generating contracts is often an exercise in selecting groups of boilerplate language to include. Disputes are often based on one lawyer figuring out potential corner cases in another lawyer's boilerplate provisions.

The circumstances of these disputes are a result of the inherent imprecision of the English language (or any spoken language for that matter). "Synonyms" in English will vary in breadth. As an illustrative example, the words "large", "mammoth", "gargantuan", and "humongous" all mean "big" though each has some varied degree of "how big" each refers. It is further posited that one cannot, with complete certainty, rank these words in the order

[9] Consideration is a legal term in contract law that refers to what one party is offering in the contract. For any contract to be enforceable, it must be supported on both (or all) sides by consideration. As a result of the way consideration is viewed under the law, it is best that no single module handles the consideration of both parties. Each party should to a contract should employ at least one code module for their respective consideration. This is because the consideration may vary wildly from contract to contract. implementations would require fewer modules in this construction.

of magnitude to which they refer. Which is bigger, mammoth or humongous? The answer is open to interpretation. While varied synonyms of "big" are not necessarily common in legal contracts, they provide an able illustrative example of the failings of the English language.

Take for example a contract that includes an addendum to be executed after the contract ends, and another provision refers to exist for the duration of the whole contract. Does the "whole contract" include the execution of addendum? The addendum is technically written within the contract, but expressly says it takes function after the contract ends? The language does not solve this apparent contradiction on its face.

Enter program code -- program code is far more precise than English, or any other spoken language. A program executes according to its code and by no other rules. The code must inherently be precise to execute. This is a well-known characteristic of program code as compared to spoken language. This being the case, one may ask, "Why do legal contracts not substantially exist in software?" Aside from the fact that the legal profession notoriously slow to adopt technology, there are two primary issues: first, security, and second, the ability to function in a trustless environment.

In order to generate a self-executing contract, one must rely on secure variable inputs. A contract may only properly execute where there is trust in its respective triggering events (e.g., a signature of a party to the contract). If the execution of a contract can be triggered based on an imperfect premise, then the contract itself isn't something that can be trusted.

Secondly, contracts are often generated in what is legally referred to as "at arm's length." This merely means that parties to the contract have competing interests and may not trust one another. A first attempt at solving this issue is to cause the software contract to use open source code. However, there are then two follow up issues with the use of open source

code: reasonable assurance that the open source code is what is actually being executed and the anonymity of the parties.

To explain the first issue, another illustrative example is necessary. “Open source” code is a broad term that refers to many things. Commonly, code is posted to a repository such as GitHub; however, when a first user claims to execute that exact code on a computer a second user does not have direct access to, the second user has no ability to prove that the posted code they have reviewed and accepted is what is being executed. While somewhere there exists contract terms (e.g., code in the form of a contract) that the second user has agreed to, the second user does not know that this is the contract that is executed on the first user’s computer.

The issue of anonymity is related to that many legal contracts are executed in secret. If the code of a given contract is posted as open source, it becomes more difficult to remain anonymous.

What’s changed? Ethereum has enabled a system whereby the users can operate in a secure, trustless, and anonymous environment. A result of executing on a public EVM enables users to know exactly what code is being executed. Thus, the code is both open source, and users are assured that the open source code is being used. Users can further generate brand new digital wallets/Ethereum accounts for each requisite contract and thus remain anonymous from any users who aren’t expressly aware of the content of the contract (presuming of course that the inputs of the software contract do not betray the identity of a user). Finally, the inputs are often based off the status of cryptographic accounts that are verified by the immutability of the Blockchain and are thus secure. Therefore, Ethereum has generated an environment ripe to build self-executing software contracts.

With a suitable platform established, the next issue is to determine the sorts of contracts one wants to build, and how exactly those are to be implemented. The most notorious aspect of a contract is that they are signed. In order to do this, each party to the contract will have a public/private key pair. The signatory key pair may or may not be connected to any sort of monetary value, the key pair serves as a party's signature. This is one of the weakest points of security with regards to any crypto system. Access to the key pair is external to the system. The party must ensure the security of their own respective private key. If the private key is compromised, the security of that party's access to the system collapses.

With a system for signatures established, contracts are often for exchanging one of three things: services, goods, and/or title. These contracts are counterbalanced with monetary value (either immediately, or at a later date) or similarly with goods, services, or title. The inputs for each vary and merit analysis.

i. Services

A contract for services is the most common contract offered at a law firm. Lawyers sell their time. Thus, an engagement letter is often the most common contract generated by a law firm. Every new client requires one. The features of such a contract include non-executing language as well as executing language. An example of non-executing language is an advisory notice. Such a notice merely advises a client of their rights in a given circumstance. While this sort of language is an important element to an engagement letter, it is not the relevant portion to the function of this article.

Executing language concerns items such as a billable rate, services covered, and a means to trigger payment. A services contract can execute based on one, or both of two principles: acceptance of the buyer (client) and accounting of the provider (lawyer). In each of these circumstances, at minimum, a contract builder will require two digital wallets, one for the client and one for the lawyer (each having its own respective public private keypairs).

Acceptance of the buyer can be triggered after services are rendered. The client indicates the services have been performed and accepts the associated fee. Acceptance is determined based on client signature on a cryptographic transfer. Once a specific predetermined transaction has occurred using the buyer's cryptographic wallet, services are flagged as accepted. While such a method can be used in multiple circumstances, the most easily illustrated is one based on a fixed or contingency fee basis, though each would require separate modules and inputs.

a. Fixed Fee

Fixed fee arrangements are based on the premise that the lawyer does a thing, and they are paid a fixed amount for that thing. The use of legal services here is merely illustrative. Legal services could just as easily be replaced with plumbing repair, accounting/financial services, painting, or any other service profession. In order to establish a fixed fee system a software module is established that includes the wallets of the client and the lawyer, though in addition, may include wallets that serve to merely to hold a condition (e.g., a Boolean). In order to alter the condition, one must have access to the necessary private key. Control over that private key is provided to the client. Thus, when the lawyer demonstrates the completion of a task, the client can use their private key to alter the

condition to “accepted”. Money is then transferred between the client and lawyer’s wallets based on an amount established in the cryptocurrency contract.

b. Contingency Fee

Contingency fee arrangements are where the payment to the lawyer is a matter of their success (e.g., 40% of a judgment award or settlement). If there is no judgment award, then the lawyer does not get paid. In order to enable this arrangement, there is an additional contingent wallet. The key pair for the contingent wallet is held by the associated dApp. When monetary value is put into the contingent wallet, the wallet automatically divides the value according to the contingency fee and delivers to the client and lawyer wallets respectively.

Where the client’s acceptance comes in first with the signature accepting the agreement, and then subsequently with escalating fee arrangements based on phase or stage of the case. The exact percentage of a contingency fee is often a function of how far along a case is (e.g., complaint/answer, start of discovery, end of discovery, beginning of trail, etc.). At each of these, the attorney may query the client whether the client wishes to proceed. In doing so, the contingency fee percentage escalates. In order to enable this, the lawyer wallet may send query transactions to the client wallet using the respective public(client)/ private (lawyer) keys.

* * *

Conversely to acceptance of the buyer, accounting of the provider is often time based. The most common fee arrangement is one based on the billable hour (often measured in tenths of an hour or 6-minute intervals). Where the accounting is time based, the provider

executes a specific predetermined transaction using the provider's cryptocurrency wallet to start a clock, and a second specific predetermined transaction to end the timekeeping. Once timekeeping is ended, the client wallet is charged based on the hourly rate. The amount in the client wallet works similarly to a retainer.

Alternatively, the associated dApp charges the client wallet while the clock is active at each relevant interval. In practice, it is more likely that a law firm would employ a system wherein the lawyer was able to adjust their time down after the clock was turned off. An attorney's time is spent in various levels of efficiency. Where the cost for a given task becomes too great, the attorney may wish to adjust their time down to account for periods of inefficiency.

Billing at an hourly rate is a relatively simplistic implementation -- the contract building dApp needs to be able to generate more flexible contracts. There is a preexisting system described in a U.S. Patent Application entitled "Cryptographically Managing Telecommunications Settlement" invented by George Melika and Akbar Thobhani (hereafter, "Melika"), that handles a service for service economy.[10] Melika describes a system for settlement of communications channels between two telecommunications companies (a circumstance where lawyers would otherwise establish a contractual basis of operation).

When a user makes a phone call to another user, the respective companies of those users (e.g., Verizon and ATT) open up a communication channel therebetween. This channel may include multiple users. Often it is the case that one telecommunications company is

[10] Cryptographically Managing Telecommunications Settlement, U.S. Pat. Pub No. 2017/0078493 invented by George Melika and Akbar Thobhani and filed on September 14, 2015 (Drafted by Colin Fowler, and available at https://patents.google.com/patent/US20170078493A1)

being charged for the service by the other. However, as more callers enter the system, calling from the opposite direction, the service charges go the other way. The system of Melika uses what is referred to as a channel monitor to keep track of the relative service fees. The fees can either offset based on individualized rates held by each of the companies and relative time of service, or each company can change an amount held in escrow.

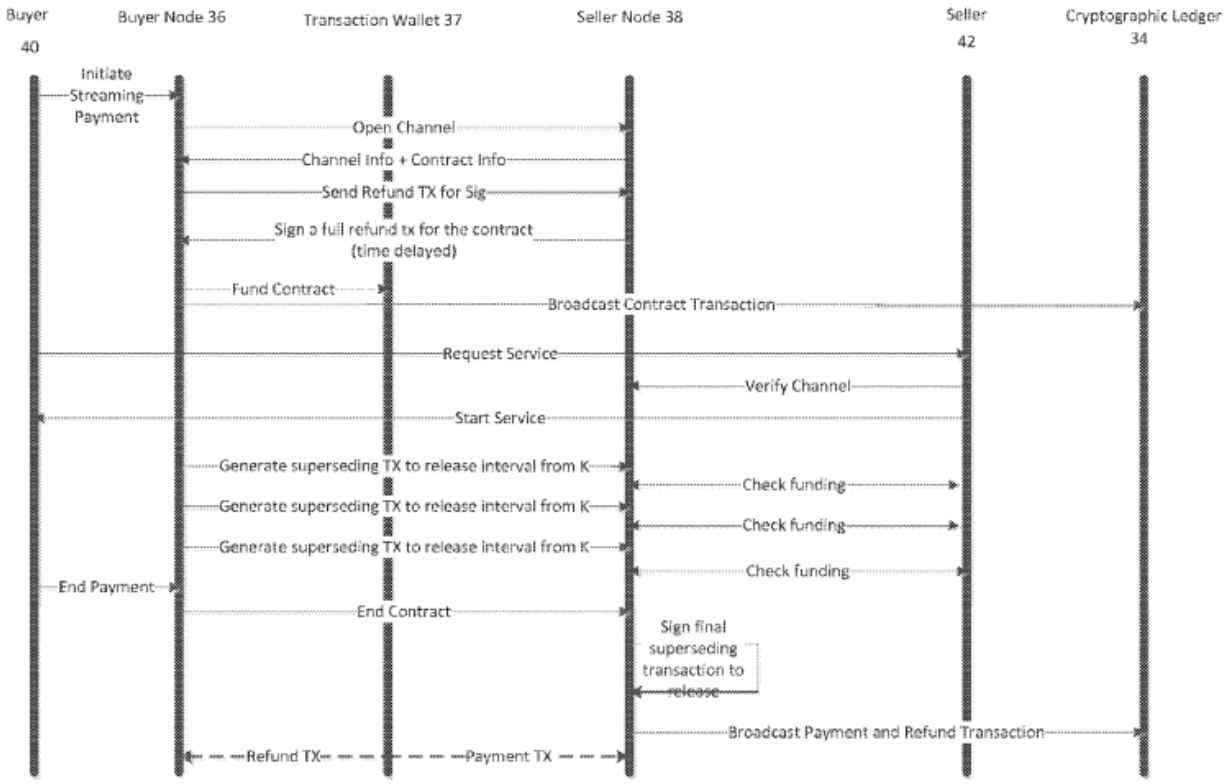

FIG. 1 is a flowchart illustrating transactions directed by a channel monitor to track fees.[11]

The system of Melika was designed and disclosed prior to the release of Ethereum and thus was based on the premise that it would function with Bitcoin. The channel monitor

[11] Cryptographically Managing Telecommunications Settlement, U.S. Pat. Pub No. 2017/0078493 invented by George Melika and Akbar Thobhani and filed on September 14, 2015 (Drafted by Colin Fowler, and available at https://patents.google.com/patent/US20170078493A1)

is disclosed in an implementation-unspecific manner but was intended originally to function based on software maintained by one of the two telecommunications companies, or as software managed by a 3rd party service. However, the channel monitor is better implemented as a dApp on the Ethereum blockchain.

In order to adapt the channel monitor to the subject dApp of this article, the contract builder, the channel monitor would need to be broken down into element modules. Effectively plug and play pieces of boilerplate that when assembled in the right combination resulted in the channel monitor. To self-execute this complex contract, one needs to build the following modules:

- a clock module controlled by a single party to the contract, the input is a Boolean (on or off), and the output is an amount of time;
- a magnitude multiplier configured to multiply an input based on a second input (A, B);
- a rate module configured to set a rate for a given input, X, and providing an output of an amount, Y; and
- a price reconciliation module configured to reconcile charges from each party to the contract (Party1, Party2).

To assemble these, there are two of each of the clock module, the magnitude multiplier and the rate module, and a single reconciliation module. The selected modules then have to be linked such that the user on each side uses their respective private keys to affect the Boolean of the clock module; the output of the clock module is used as the input, X, of the rate module; the output of the rate module, Y, is connected to the magnitude module input, A, while the second input of the magnitude module is controlled by the opposite party,

wherein the number of users at a given period of time is indicated; and the output of each respective magnitude module is provided to the price reconciliation module where the comparative amounts are compared to one another, and the remainder is charged to the party that has received the least service.

While these modules are expressly used to build the example of the telecommunications channel monitor, these same modules may be used in the construction of multiple other contracts. The modules are selected based on the specific needs of the clients.

ii. Goods

While a contract for goods is not necessarily one that a law firm itself is a party to, these contracts are still built by law firms. Contracts of this nature are often for supplies, or materials used to manufacture larger goods. For example, a cell phone includes many parts that are not manufactured by the cell phone developer. The cell phone manufacturer, will, for example, purchase processors from another company.

There are many concerns involved in this sort of contract beyond an amount of goods and the cost thereto. For example, such a contract may be a requirements contract, where the purchaser determines the number of units they wish to purchase based on their own business interests. Or alternatively, an output contract, where the purchaser agrees to purchase 100% of the output the manufacturer generates (and is thus based on manufacturing effectiveness).

To generate the two contracts described above, one would make use of a quantity module and a received goods module. In the case of a requirements contract, the buyer has control over the quantity module (e.g., the buyer's private key enables them to input a

quantity of a given good desired). The buyer additionally has control over the received goods module. When the goods are received, the buyer provides necessary input into the received goods module, and the smart contract will cause payment to transfer between the buyer and manufacturer wallets.

In the case of an output contract, the quantity and the received goods modules are in control of different parties (e.g., the manufacturer has control of the quantity module, and the buyer has control over goods received module). The manufacturer indicates the amount of the given good they have generated, and subsequently the buyer indicates they have received the goods.

Goods contracts do not always execute ideally. Thus far, only a binary possibility of contract resolution has been described. Unlike services contracts, goods contracts may be partially fulfilled. Such a circumstance occurs when the manufacturer only sends some of the required/output amount of a good. The buyer has a number of responses to receiving only some of the expected quantity. The buyer may accept partial delivery and pay in kind, or the buyer may send the entire shipment back as a rejection. In each of these circumstances, the received goods module is configured to handle multiple inputs. To do this the received goods module is set up such that it receives input from the quantity module and can alter that quantity.

iii. Title

A contract for title may sometimes be similar to that for goods but includes an additional regulatory component. Traditionally title is something transferred between parties with respect to assets that cannot physically be moved (e.g., intellectual property, company

ownership, real estate, etc.). However, title can also be applied to goods that can physically change hands (e.g., cars, boats, etc).

An important distinction between transfers of title and goods are that title is handed by a regulatory agency. Receipt of title is based on what the government agency is aware of, rather than what the owner is aware of (with some exceptions). In the future, it is theorized that most if not all title transactions will exist entirely on a blockchain type system. At the time of writing, these systems are largely localized databases managed by the relevant regulatory agency.

Accordingly, there is a module to identify the property in question and the relevant regulatory agency. For example, if the property is a car, a particular state's DMV is identified. If the property is a patent, the USPTO is identified. If the property is a trademark, a different part of the USPTO is identified.

The module is programmed to interact with the relevant regulatory agency based on the structure of that agency's website. This will require consistent updates because the relevant websites of the regulatory agencies are also updated frequently. Further, some agencies may not be accessible based on the use of CAPTCHAs. In such a circumstance, the dApp may instead be programed to generate paper documents that may be mailed to the relevant agency.

To execute a transfer in title one needs three primary inputs: an identification of the property (which in turn identifies the relevant regulatory agency), an identification of the type of conveyance (e.g., ownership or a security interest), and a signature of the current owner. Each of these three inputs is built into a title module. As with previous types of

contracts, handling the consideration used in exchange for title should exist separate from the title module.

A similar module may be used to complete a settlement agreement. A settlement agreement may include the transfer of title, though the most common action in a settlement agreement is the dismissal of a pending lawsuit. Lawsuit dismissal is not precisely a title transaction, but deals with a regulatory agency (e.g., a court) none the less. The same 3 inputs are relevant: type of property (e.g., a lawsuit in a particular state/district), type of conveyance (e.g., dismissal of suit), and signature of the interested party.

B. COMPLEX CONTRACT COMPONENTS

Thus far, this article has referenced contracts for services, goods and title. These are not the only types of consideration a contract may be based on. Some contracts are based on more abstract concepts. Most other types of consideration would be viewed as non-executing. That is, the consideration offered in exchange is based on the premise of non-action (e.g., silence, agreement not to sue, agreement not to enforce particular rights, agreement not to compete, etc.). In such a case, the module is merely a print function with inputs that merely extract non-variable terms from other modules (e.g., subject matter covered).

In some circumstances, one may want to build modules that verify that the party has taken no action. There is an issue in verifying abstract conditions that the input sources may not be trusted. For example, if a contract includes a non-compete clause, there is not an objective way to verify whether the clause has been violated. Despite this, the dApp may include advisory inputs. Advisory inputs merely cause messages to propagate through the system. An example of such an input is analysis of a given user's LinkedIn page. This

particular example is somewhat problematic. The problem is that the dApp is all open source the party is aware that their LinkedIn page is under observation; thus, they need merely not edit the page to avoid triggering the advisory input.

Another important module that may exist separate of consideration terms is that of anonymity. In order to enable the anonymity of the parties to a contract, the dApp creates new Ethereum accounts for the entire smart contract. Additionally, the smart contract must be funded (and provided gas) in an anonymous manner. To some extent, each Ethereum wallet is already anonymous in the sense that the public key of the account is not inherently publicly tied to anyone. That said, the after repeated patterns of use, one can associate a particular entity to a particular Ethereum wallet. While those seeking anonymity should thus divest their holdings into multiple wallets in order to obscure patterns, the anonymity module can establish a form of open source cryptography whereby ether is moved between a random number of wallets and using random amounts and intervals of transaction. To further obscure the source of money, a third-party exchange may be utilized in order to convert the ether to a form of fiat currency and then back to ether in another wallet.

Similarly to the anonymity module, the parties may wish to not hold value in cryptocurrency. Cryptocurrencies are notoriously volatile in value. Accordingly, a conversion module may be slotted into smart contracts that uses a third-party exchange for all amounts that have finished transacting to convert from ether to a fiat currency. The module may be optimized for speed or to reduce exchange fees. At the time of writing this, the largest exchange in the US, Coinbase (via the GDAX platform) does not charge fees for posting trades but does charge fees for completing posted trades. In order to post a trade, the

dApp can post a trade for the minimum required variance and wait until the trade completes, Alternatively, the conversion module can immediately execute a market order.

A sample complex contract that brings a number of the discussed concepts together is a contractor agreement[12] for a software engineer. Fundamentally, a contractor agreement is an exchange of services, often for money. However, contractor agreements usually include a "work for hire" provision that states that all intellectual property generated by the employee while working belongs to the employer. A work for hire provision is a title transfer contract.

In order to build the contractor agreement, the employer first includes an acceptance module and a monetary payment module triggered by the output (Boolean) of the acceptance module. The title module is accessed by both parties. The contractor provides their signature, while the employer has delayed access to the type of property (e.g., trademark, copyright, or patent). The type of conveyance is entered immediately as an assignment. The contract may further include a clock module to enforce a lifespan of the work.

[12] Contractor agreements are distinctly less complicated than employee agreements and thus are easier to generate in a self-executing manner.

IV. RESULTS

While running a proof of concept application, I was able to generate and execute a sample Series A equity financing agreement contract in approximately five minutes. Granted, by the point the code was functional, I had long since memorized the process, and the speed at which I could run through the user interface was significantly higher than one who was taking careful consideration of their choices. That said, even on a first pass through, it does not take more than fifteen minutes to operate the interface.

Steps in operating include, generating a contract through adding blocks

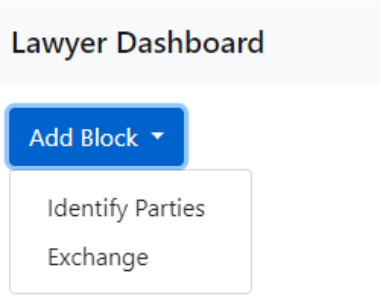

FIG. 2 is a screenshot of the user interface to create blocks in a contract.

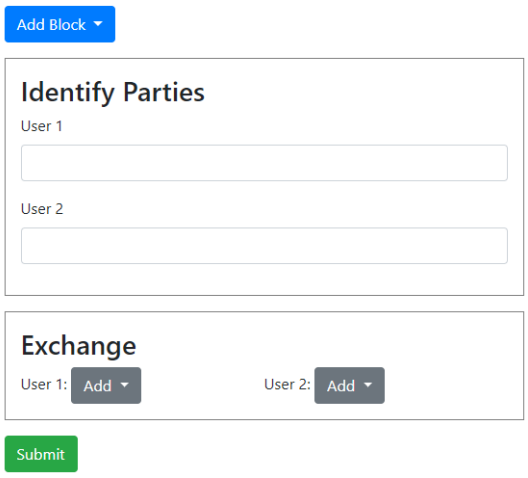

FIG.3 is a screenshot of the user interface with two blocks.

The user adds blocks by clicking the add block buttons (see FIG. 2, an illustration of an initialization interface). Here two blocks are implemented as required for the relevant contract (see FIG. 3, an illustration of two selected blocks). Once added, the blocks are filled out to include the exchange for capital (here, ETH coins) for both stock and stock options (represented by ERC20 tokens). Relevant parties to the contract are identified by their respective Ethereum wallet public addresses (see FIG. 4). The sides of the contract are

FIG. 4 is a screenshot including the parties block with the relevant wallet addresses.

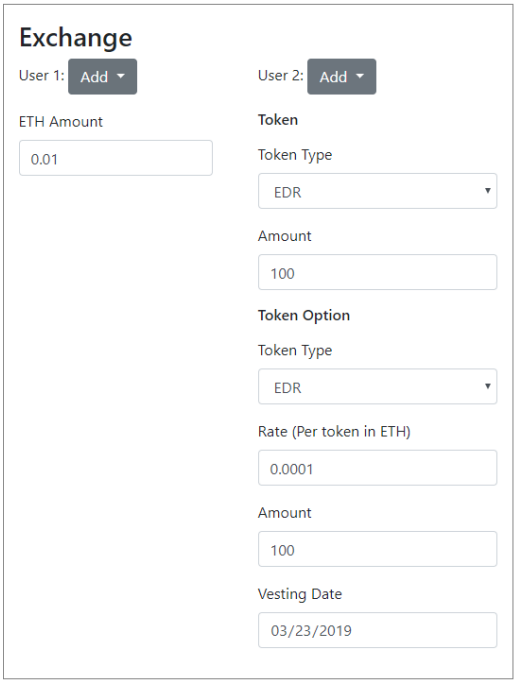

FIG. 5 is a screenshot including the exchange block filled out.

balanced on either side of the Exchange block. One user provides capital in exchange for "stock" and "stock options" (see FIG. 5). When complete, the attorney clicks the submit button, and the contract is submitted to the Ethereum network and is provided a contract address (see FIG. 6) that may be inspected on etherscan.io[13].

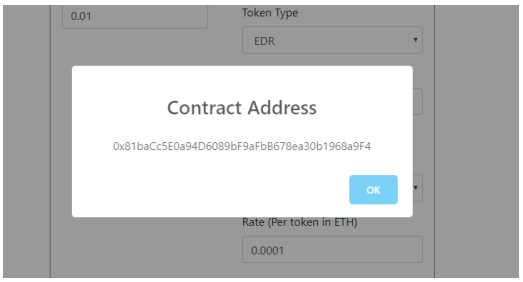

FIG. 6 is a screenshot of a contract address.

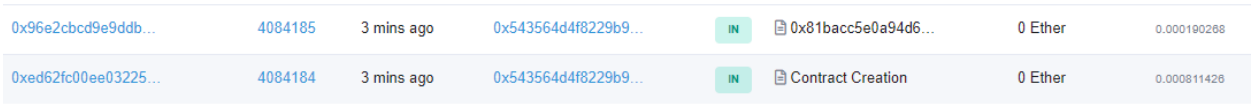

FIG. 7 is a screenshot of transaction records on Etherscan.

Shown in FIG. 7 a first transaction is generated on the blockchain that creates the contract (hash prefix: 0xed'), and a second transaction is performed by the "lawyer account" (hash prefix: 0x96'). Also available on Etherscan is the code for the contract (see FIG. 8). From this point onwards, the lawyers job is done. All that remains are for the relevant parties to act on the contract.

[13] Notably, texts are run on the Rinkeby test blockchain in order to make reduce cost of operation. https://rinkeby.etherscan.io

Contract Source Code

Find Similar Contracts

```
pragma solidity >=0.4.25 <0.6.0;

contract Futures {

    address payable public lawyer;
    address public tokenAddress;
    uint public tokenAmount;
    uint public futurePrice;
    address payable public shortPlayer;
    address payable public longPlayer;
    uint public vestingDate;
    bool public contractStarted = false;
    uint public ethAmount;
    uint public futureAmount;

    event TermsAdded(address lawyer, uint timestamp);
    event Signed(address signer, uint amount, uint timestamp);
    event Executed(address executor, uint timestamp);

    constructor() public {
        lawyer = msg.sender;
    }

    function addTerms(address _tokenAddress, uint _tokenAmount, uint _futurePrice, address payable _shortPlayer, address payable _longPlayer, uint
ethAmount, uint  vestingDate, uint  futureAmount) public {
```

FIG. 8 is an Etherscan screenshot publicly displaying the code of the contract.

Using a web interface, both parties identify the contract address and sign using their relevant interfaces.

FIG. 9 is a contract signing interface.

When the last signature is processed on the blockchain, the stock is exchanged for the capital (see FIG. 10). Though the smart contract continues to exist because a stock option

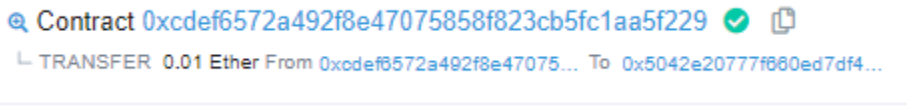

FIG. 10 illustrates successful transfer.

still exists. The stock option is executed in a separate interface that enables the VC to execute as long as the vesting date has been met, the amount of stock available has not been

exhausted, and the price is set correctly. Once all of the contract has been executed, the record of all actions is present on Etherscan (see. FIG. 11).

FIG. 11 is an Etherscan screenshot including annotations labeling transactions.

One may notice that the timestamps between transactions is approximately eleven minutes. This time is not reflective of the lawyer's involvement. All of the time a lawyer spends occurs prior to the first transaction (contract creation).

A sampling of my associate colleagues who have drafted Series A round equity financing agreements averaged approximately 3 hours to draft and revise the relevant contract. The billing rates of my colleagues interviewed ranged from $435 an hour to 590$ (averaging to $530 an hour). With each of these contracts, a partner reviewed for approximately fifteen to twenty minutes. Partners charge between $780 and $1115 per hour. However, the partner involved in each of the contracts reviewed charges $980 an hour. Therefore, the average series A equity contract costs approximately $1884**.**

A fifteen-minute operation time counts for 0.3 billed hours comparatively costs $159 and additional partner review remains the same at $294. *Use of the application saved $1431 in billed time.* It's theoretically possible that the associate wouldn't be involved in the generation of the contract at all and only the partner's time would be necessary, but this would require further investigation after implementation.

V. DISCUSSION OF METHODOLOGY

A. BUILDING THE LAW FIRM DAPP

The best way to build the relevant contracts is a modular approach. Due to the potential complexity of contracts, the best way to efficiently construct them is to do so in a piecemeal style. I've also noticed through my own legal practice that I usually draft contracts from an original shell that is "close" to the final product, copy-paste paragraphs from other contracts I've worked on, and then modify to fit the particular client's circumstances. This approach is wildly inefficient in that it requires that the drafter remember where the paragraphs they want to draw from are located and doesn't outline a set of variables that can be substituted for the current client. Making the changes in a text document such that the sections/clauses flow together and being certain that the "values" from the prior client are cleared out entirely is a time-consuming process.

However, the concept of picking and choosing from something one knew worked before is not bad. This is, after all, how a significant amount of software is written (e.g., drawing licensable libraries). Thus, an interface that enables users to choose from a selection of modules that are all in one place and includes easily edited values (that do not store the values of prior clients is useful.

Importantly, because a module-based contract builder is not actually constructing an English sentence it may represent the functions of the contract to be built abstractly. Aside from issues of precision in English, human languages often suffer from the ability to convey abstract concepts clearly. Conversely, mathematical expressions and program code have no issues in conveying abstract concepts with high levels of specificity. For purposes of being a data structure, human language is exceedingly inefficient.

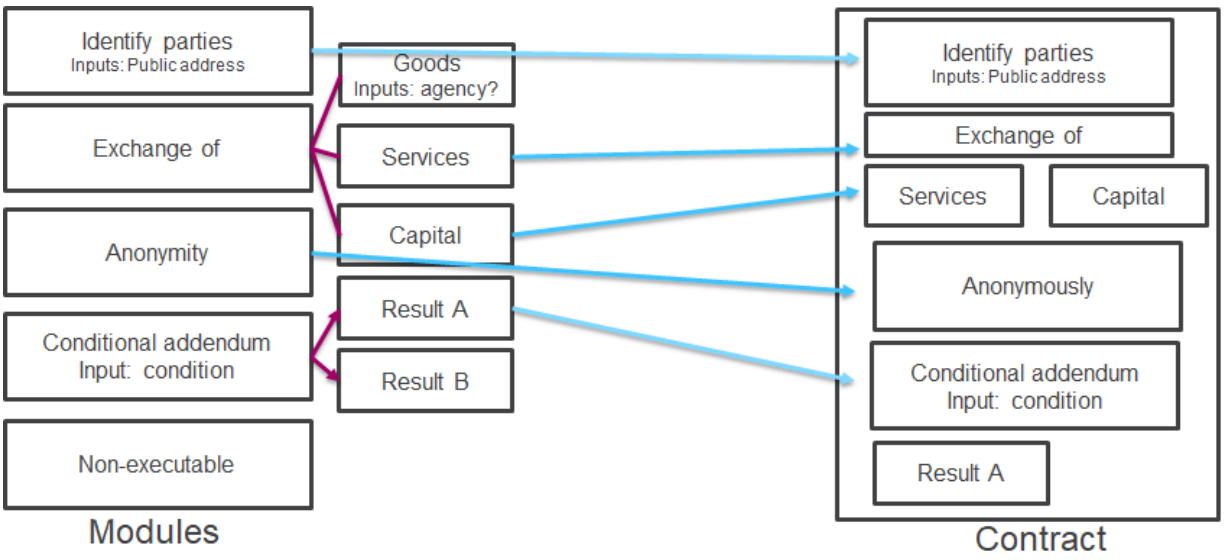

FIG. 12 is an illustration of module selection.

Users select the modules (see FIG. 12) that are relevant to their contract and build that contract. The selection interface is accessed by the attorney, while review and signature interfaces are accessed by the relevant parties. The sample contract built in Figure 12 doesn't exactly include any complete sentence, though from casually parsing from top to bottom, a read is given a general idea of what this contract would do. This is a contract that exchanges money for services anonymously and renders some additional result upon the satisfaction of a given condition.

To improve the user interface the dApp would include smart contract modules. This article has disclosed a number of contract styles, some are more common than others and thus having pre-built contract modules improves a user's experience. Modules also improve the ease of use.

In addition to the executing modules, some modules are used merely to hold data (e.g, a POJO). These modules are referenced as input by other executing modules in conformance

to standard object-oriented design practice. Such holding objects include public private key pairs for wallets and amounts that the executing modules transfer.

Another type of module must send messaging between the various parties to the contract. Upon initial generation, the parties designate a contact address for each of them and upon the receipt of every input, the messaging system notifies all parties to the contract that the input had been received.

The proof of concept includes merely two modules (identify and exchange), and 3 sub-modules (exchange of: capital, "stock" and "stock options") that are necessary for the purposes of generating a Series A equity seed funding contract. Further development to generate more modules enables a greater breadth of contracts to be built on the Ethereum blockchain.

i. Proof of Concept

The variety and scope of potential contracts is too great a scope for one person to solve in less than a year. However, in order to prove that a given concept is effective, some selections needed to be made. An example of a useful contract that this system could produce is an equity financing agreement, such as those that are often obtained by start-up companies. Equity financing is an agreement where a company sells a part or all of itself (often via stock/shares) in exchange for money. The money is used to finance operation of the company, often before a long-term revenue model is implemented.

In determining a type of contract to implement in modules there are a few factors that are important to consider. First, the contact must be common enough that executing it would not be an uncommon usage -- users would actually need to want this contract. Second, the

contract must be uncommon enough that the it cannot simply by executed with boilerplate. Third, the contract needs to be complex enough to not merely be a “one-and-done” contract. A contract that executes completely upon completion has little need for public execution. If the whole contract completes right away, there is no ongoing concern that the contract is doing something nefarious in the background.

With respect to the first factor, avoiding rare or unique use cases, the equity financing agreement is exceedingly common. While this sort of contract tends to have greater regional relevancy to the Bay Area and other geographic regions that have a culture of start-up companies, it’s still an important contract and commonly used. Many firms are catered toward addressing start-up culture, and indeed cater start-ups. The equity financing agreement is one of the first contracts a start-up needs in order to obtain funding to operate. Any given start-up may execute a number of these sorts of contracts in their lifetime based on how many seed rounds that start-up undergoes.

With respect to the second factor, avoiding boilerplate, the equity financing agreement can take several different formats. The type of stock provided (preferred/non-preferred), which seed round the present contract pertains to (e.g., the first round or a later round), the amount of the investment involved, how many investors are involved, what the prior investors have rights in, and the expectations of investor rights may cause unanticipated complications.

With respect to the third factor, avoiding one-and-done contracts, equity agreements often include stock options which are executed at a later date. One of the primary advantages of using a blockchain based system is that there is a record of what code was executed or will be executed. If a given contract executes once and is forever complete, there is little

reason for requiring public execution. The results of the contract are apparent from the completed execution. Stock options are a contract item that live on. One has to track a future vesting date, an option price, and a set of stock that is available at that time. The ongoing executing code would be of interest to the parties relevant to the contract and thus public execution would be relevant.

B. CHALLENGES

Now that an equity financing agreement has been identified as a proof of concept contract, I will examine issues that arose while developing the proof of concept. Some of the issues are purely technical while others are legal and lead to technical solutions. Those issues are first, how does one represent movement of stock on a blockchain? Second, how should the blockchain be structured (e.g., public or private)? Third, how does one access the program?

i. How does one represent stock exchanges?

Early in this article, it was discussed that the Bitcoin blockchain could store documents that in turn represent the movement of stock. This is an inadequate solution as it doesn't actually move the stock, merely represents an intention to do so. Representing an intension to trade doesn't actually reduce the amount of time an attorney needs to spend involved in a given case. It is a contract to act in the future, or a contract to contract further. Both circumstances require further attention. Therefore, the system should make use of cryptotokens that, themselves, represent the stock in question.

While established companies cannot merely shift all their stock out of public exchanges, start-ups are generally in the unique position that they can dictate the rules on

their stock. The Ethereum blockchain commonly includes utility tokens operating under the ERC-20 standard[14] (stands for "Ethereum request for comment-20"). ERC-20 tokens are those that are tied to a specific smart contract that operates on the Ethereum blockchain, these are separate from Ether, which is the base currency of the Ethereum blockchain.

While the ERC-20 token acts as a functional example of stock is has a few legal deficiencies regarding limitations of downstream transactions. ERC-20 tokens can be traded freely amongst users unless a centralized authority provides limits. Fortunately, there is another type of token, following the ST-20 standard (stands for "security token-20"). Security tokens enable the type of limitations on transacting that actual shares of stock include.

ST-20 was developed by Polymath Labs[15] within the last year. As a result, platforms using ST-20 tokens do not quite have the sufficient interfaces to integrate with external services (such as a contract builder). For the purposes of a proof of concept, an ERC-20 token serves as a suitable substitute. As the laws regarding treatment of security tokens are not uniform yet, and in many cases unwritten, this proof of concept adopts the simpler of the proposed legal options. In the simpler versions of potential legal regimes, there is no functional difference to the modular contract builder whether a ST-20 token is used or an ERC-20 token is used.

[14] Buterin, Vitalik and Vogelsteller, Fabian, ERC-20 token standard commit on GitHub, last updated on March 8, 2019 (available at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md accessed on 3/24/2019).
[15] Polymath Repository on GitHub, last updated March 24, 2019 (available at https://github.com/polymathnetwork accessed on 3/24/2019).

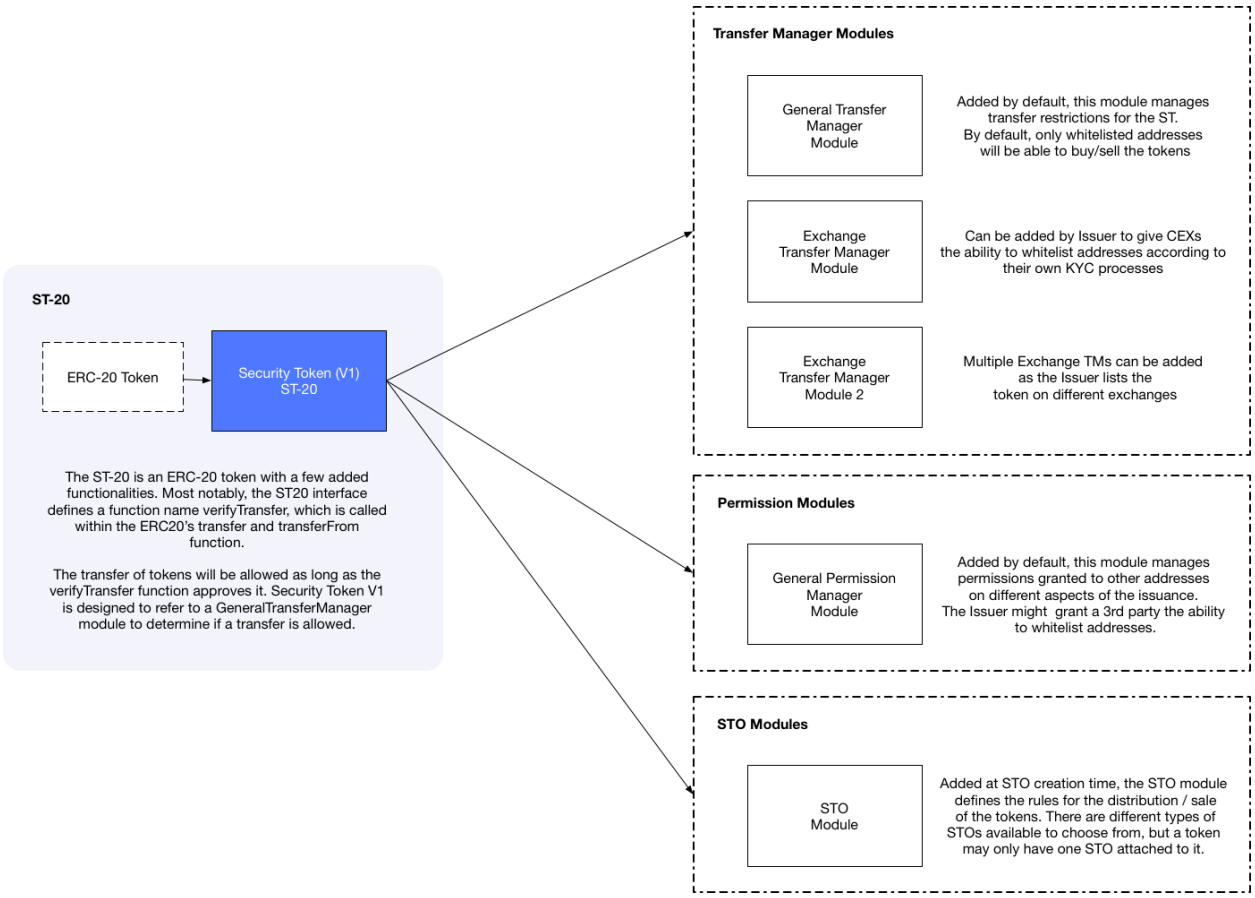

FIG. 13 is block diagram of ST-20 token features[16]

Notably, future developments in legal treatment of security tokens can affect the relative interchangeability (e.g., between ST-20 and ERC-20). At the time of writing this article, states are split on how to treat security tokens between two regimes.[17] Some states want to treat security tokens as if they are money where a right to tokens requires that those tokens be on hand at all times, whereas other states are comfortable with a contract right to have the tokens at a future date.

[16] Ruiz, Pablo, "Overview of the ST-20 Interface and polymath Core", Polymath Blog, April 13, 2018 (available at https://blog.polymath.network/overview-of-the-st-20-interface-and-polymath-core-86bf64c8929 accessed on 3/3/2019).
[17] https://www.forbes.com/sites/andreatinianow/2019/03/07/a-split-emerges-in-blockchain-law-wyomings-approach-versus-the-supplemental-act/#4730b2ab719a

The legal distinction leads to a technical position regarding a stock options. Where the tokens must be on hand, a module identifying a stock option must highlight the specific tokens that would be traded were that option executed. Where a stock option merely requires tokens upon request those tokens may either be identified at the time of initial agreement or created/minted at upon execution of the option. Either of these options may be selected when creating the stock option contract.

ii. Public or Private Chain

A blockchain is ultimately just a data structure supported by multiple nodes. The character of those nodes is not inherently specified. The term “public chain” refers to blockchains that are maintained by an enormous number of nodes. A private chain is a blockchain supported only by nodes of known parties. Management of a private chain is the easiest option because it requires action of the fewest parties. However, on a private chain, control of appending actions to the chain is subject of the control of the small number of nodes on the chain. The “miners” or EVM are in fact those nodes. Use of a private chain effectively ruins the concept of public execution of code. The machine that acts as the EVM could effective run whatever program code it wanted. Private chains are thus not effective in the sense of trustless execution. Private chains are additionally prone to 51% attacks, especially if a single node going inactive reduces the overall processing power of the network by any significant degree.

For at least the reasons listed above, the contract builder cannot operate on a private network blockchain. Despite any downsides of a public blockchain they will not outweigh the feature breaking aspects of a private chain.

Connection to public chains has one very large disadvantage: the disk space requirement. Communicating directly with a public chain requires that the user support a node of the public chain. A node is required to have a full copy of the blockchain. The blockchain of most cryptocurrencies is exceedingly large. As of this writing, the Ethereum blockchain is approximately 200GB for a full node[18] (and over 2TB for an archival node[19]).

For the practical purposes of use on the machine I was doing the relevant work on, an addition of 200GB of storage was not an option I could elect. External to a proof of concept, the issue of disk space is not particularly daunting. Any professional application would likely be run on a cloud service (AWS, Azure, etc…). However, inability to support a full-node of the Ethereum blockchain is an issue that others have anticipated. There are a number of APIs that provide access to full-nodes. One such service is BlockCluster.io[20]

Use of a node service enables those with limited disk space to conduct experiments. Accordingly, the proof of concept functions with a public blockchain.

iii. How does one access the program?

The contract building dApp is a web application. At some level, some of the "backend" must necessarily execute on a EVM that could be anywhere in the world. The dApp therefore has access to the Internet. The interface thus uses HTML. The HTML code was developed using an online WYSIWYG (what you see is what you get) editor as well as some direct coding to pass in the correct variables.

[18] Etherscan statistics for a full node (available at https://etherscan.io/chartsync/chaindefault and accessed on 3/24/2019).
[19] Etherscan statistics for a full archival node (available at https://etherscan.io/chartsync/chainarchive and accessed on 3/24/2019).
[20] BlockCluster website (available at https://www.blockcluster.io/ and accessed on 3/24/2019).

From a user standpoint, it makes sense to have different interfaces for the different parties. Often parties to a contract are never in the same room with one another. Many documents that include multiple signatures are all signed individually and then stored with multiple signature pages that include only the single signature. As a result, there should be individualized ways to access the Application. The proof of concept includes four. A primary lawyer dashboard where the contract is constructed. Signature pages individually for the first and second parties to the contract, and finally an interface that executes the future options.

While the application does execute publicly, most people aren't going to be able to understand the code. Thus, having graphic interfaces that state very clearly what is going on is important. While code is not inherently readable by most of the populace, it is unambiguous. Where a problem occurs, it would be exceedingly easy to figure out that the problem had occurred, and who did the wrongdoing.

For the sake of testing, the proof of concept makes use of node.js rather than a cloud service to communicate with the web browser-based interface. The hosted node passes values to the main application code. The application code must necessarily be resident on some backend server.

Accordingly, the application includes both a centralized and a decentralized backend. The centralized backend is the server running the web interface, and the application that includes pre-built contract module constructs that are selected through the browser interface. The decentralized backend is the EVM that executes the contracts that were built via the centralized backend.

C. PROGRAM STRUCTURE

The proof of concept includes a HTML user interface that is being listened to by the node. The node reports to the application. The Application makes calls to methods from the BlockCluster.io API in order to coordinate with the Rinkeby Ethereum test network.

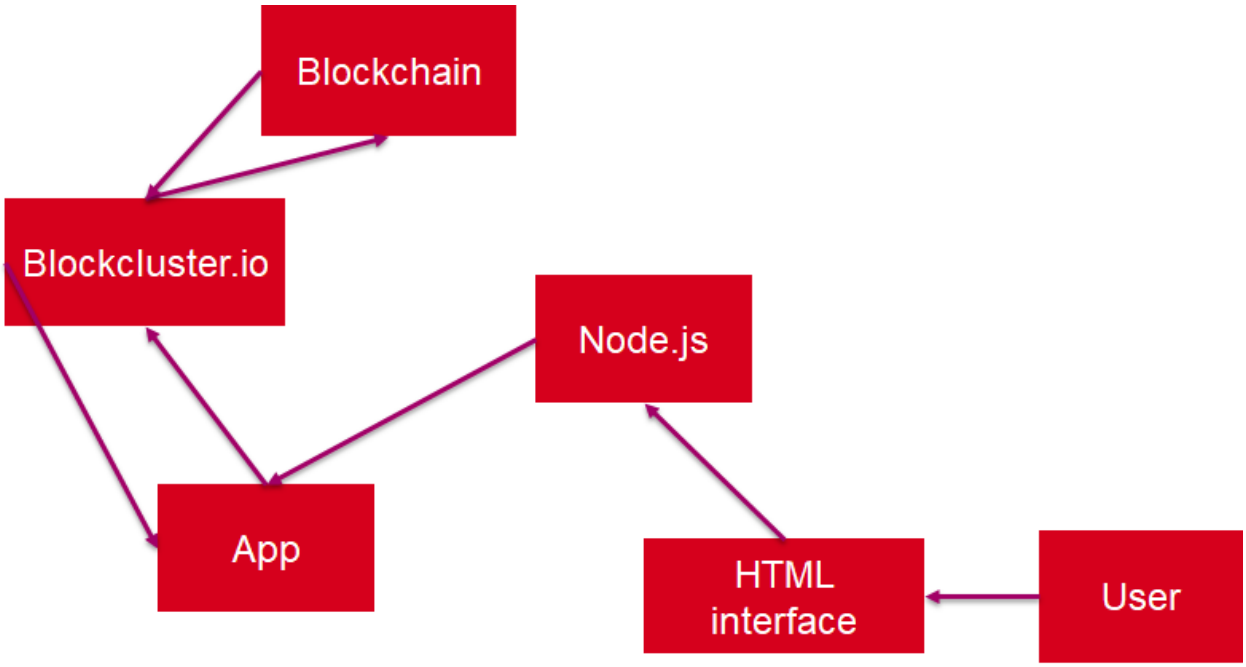

FIG. 14 is a block diagram of the proof of concept application.

BlockCluster.io provides more than a conduit to the blockchain. The BlockCluster API includes prebuilt libraries with respect to instructing Ethereum wallets and creating the smart contracts on the Ethereum blockchain. The API includes traditional getters/setters as well as transactional methods. Most of the work generating the contracts is performed by calls to this API.

VI. CONCLUSION AND FUTURE WORK

Application of blockchain structures and public virtual machines to legal contracts is an effective practice because the process is transparent and secure. Use of the inherent security in the blockchain enables contract execution in program code that was not previously available. Program code is more precise than human languages, which can reduce overall potential of costly court disputes.

There are some disadvantages to legal contracts that are implemented in Ethereum smart contracts. Not every contract can be effectuated using Ethereum smart contracts. The best sort of contracts to generate on a blockchain are those that involve satisfaction of publicly observable conditions. Contracts that require one party to exercise their judgment are difficult to reflect in program code. Conditions such as non-action with regard to taking part in a given activity that is not typically monitored by computers (e.g., working at a competing company) cannot be effectively implemented in a smart contract.

Despite the gaps in programmable contracts, many important contract types are better when implemented as Ethereum smart contracts. Specific examples of contracts that can be effectively implemented in Ethereum smart contracts include exchanges of assets or money that occur over extended periods of time (e.g., service contracts, financing agreements, payment based on results).

The proof of concept is a small piece of a significantly larger greater concept. While it has been proven that generation of a single contract has been vastly improved upon, that contract was constructed in hindsight (e.g., before being constructed by the proof of concept, I already knew the substance of the contract).

In order to be truly effective, one would have to be able to create a contract, on the fly, without a previous example to work from. Once the general terms are negotiated, those terms need to be put into an executable form quickly through a modular interface. To do so requires a significant time investment building contract modules. Some of these modules were discussed specifically -- many were not. Further, the modules need to interact with one another in variable fashion based on the existence or non-existence of one another. Doing so requires a significant number of underlying heuristics in the system that cause the modules to function intuitively.

VII. REFERENCES

[1] Rouse, Margaret, "Smart Contract", last updated on April 2018, (available at http://searchcompliance.techtarget.com/definition/smart-contract and accessed 3/29/2019).

[2] Bitcoin Wiki "Confirmation" last updated March 16, 2018, available at https://en.bitcoin.it/wiki/Confirmation and accessed on 4/14/2018).

[3] Block Explorer available at https://blockchain.info/ and accessed 3/25/2018.

[4] Etherscan available at https://etherscan.io/ accessed 3/24/2019.

[5] Coleman, Jeff, question answer on Stack Exchange, "What is meant by the term 'gas'?" last edited on May 31, 2017 (available at https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas and accessed on 3/24/2018).

[6] Cryptographically Managing Telecommunications Settlement, U.S. Pat. Pub No. 2017/0078493 invented by George Melika and Akbar Thobhani and filed on September 14, 2015 (Drafted by Colin Fowler, and available at https://patents.google.com/patent/US20170078493A1)

[7] Buterin, Vitalik and Vogelsteller, Fabian, ERC-20 token standard commit on GitHub, last updated on March 8, 2019 (available at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md accessed on 3/24/2019).

[8] Polymath Repository on GitHub, last updated March 24, 2019 (available at https://github.com/polymathnetwork accessed on 3/24/2019).

[9] Ruiz, Pablo, "Overview of the ST-20 Interface and polymath Core", Polymath Blog, April 13, 2018 (available at https://blog.polymath.network/overview-of-the-st-20-interface-and-polymath-core-86bf64c8929 accessed on 3/3/2019).

[10] Etherscan statistics for a full node (available at https://etherscan.io/chartsync/chaindefault and accessed on 3/24/2019).

[11] Etherscan statistics for a full archival node (available at https://etherscan.io/chartsync/chainarchive and accessed on 3/24/2019).

[12] BlockCluster website (available at https://www.blockcluster.io/ and accessed on 3/24/2019).