BitCAD

Encrypted Smart Platform
New Business Era

“Each our new fantasy is a construction from other fantasies, ideas and accumulated experience (both acquired through direct interactions, and received through third-party intermediaries).” - Vlad Mitrofanov, CEO BitCAD

Abstract

BitCAD – is a multipurpose smart platform based on the Ethereum blockchain. BitCAD users create reliable, trustworthy relationships between counterparties, thereby levelling out distrust, unreliability and unpredictability of business partners.

Platform’s mission lies in providing a venue for seamless integration of business and computer technology. BitCAD will help users to automate routine business processes, find business opportunities globally and locally, make instant value transactions and resolve disputes in a trustless environment.

Thanks to blockchain technology BitCAD eliminates financial and legal intermediaries and enables users to form more agile supply chains through smart contracts that automatically find, negotiate with and close deals with partners the world over.

This white paper explains the basics of using BitCAD, benefits and challenges of real world implementation and practical application advice, with the aim of increasing efficiency, flexibility and modernization of business processes.

1 BitCAD Blockchain - global business platform of the future

Blockchain is a distributed ledger, now primarily known as an integral part of bitcoin in the financial services world, that provides a distributed system of trusted assets and transactions without the need for a central trusted authority.

For manufacturers and their suppliers or logistic partners, an individual transaction in a block might contain bills of lading for raw materials or finished goods, proof of their origin, quality of operations performed or instructions for the place and time of a delivery. In each case, the information could be stored, verified, shared and changed by the partners without going to the cost and delay of negotiating formal contracts or requiring paperwork such as letters of credit from a bank or a bond for a transportation provider.

Unlike in a traditional supply chain, where these documents and contracts are maintained by each partner's purchasing, accounting or legal department, in a blockchain these elements are stored on many decentralized nodes. Their privacy and integrity is maintained by "miner-accountants" rather than by a counterparty or a third party such as a bank.

BitCAD provides a simple constructor of smart contracts and DAO, with terms and conditions both sides can specify and that assure trust in the enforceability of the contract and the identity of the counterparty. This system of distributed trust allows for lower transaction costs in the short term, but this is just the beginning. In the long run it will enable more agile value chains, "Live Price" technology, closer cooperation with business partners and faster integration with the Internet of Things (IoT), among other things.

1.1 Blockchain Application in BitCAD

Sluggishness of modern business processes, low level of trust between counterparties and the increase of friction in relations with governmental institutions have stimulated public demand for the development of simple, fast and accessible interactive environment without unnecessary intermediaries - BitCAD.

Business has to spend large amounts of time, money and effort on negotiation, communications and paperwork to overcome this absence of trust. This is where transformative power of BitCAD lies, delivered by three critical capabilities:

• Distributed integrity and reputation. BitCAD gives users immediate and low-cost trust in the identity and reputation of the counterparty in any financial or trading relationship. This not only reduces the cost and time of transactions with known partners, but reduces the time and cost required to establish new business relationships. It also expands the universe of suppliers and customers for everything from raw materials to shipping and repair services, delivering quantum leaps in efficiency and agility.

• Built-in monetary incentives to assure the security of every transaction and asset in the blockchain. This allows blockchain technology to be used not only for transactions, but as a registry and inventory system for recording, tracking and monitoring all assets across multiple value chain partners. This secure information can range from information about raw materials or work-in-progress to intellectual property such as product specifications, purchase orders, warranty recalls or any currency or contract.

• The ability to tap rules-based intelligence to perform business functions. Blockchains enable the creation of intelligent, embedded and trusted program code, letting participants build terms, conditions and other logic into contracts and other transactions. It allows business partners to automatically monitor prices, delivery times and other conditions, and automatically negotiate and complete transactions in real time. This reduces transaction costs, maximizes efficiency and allows manufacturers to use data in different ways.

How BitCAD can change modern business-processes:

• Smart contracts: It would take the form of a computer program that runs on the blockchain and is executed by the entire blockchain network. Its program code — the terms and conditions of the contract — cannot be changed, and thus provides the trust that used to require elaborate control and audit processes. Not only can blockchain contracts contain the same level of detail as a physical contract, they can do something no conventional contract can: perform tasks such as negotiating prices and monitoring inventory levels. This, again, replaces expensive, manual effort with automated, dynamic tracking of supply chains, inventory levels and prices to reduce costs and maximize profits. BitCAD can transform the vision of an “any-to-any” marketplace into reality.

• Smart equipment and products: Consider, for example, a smart vending machine that registers itself on a platform and tracks its own inventory and cash position. The machine will not only issue a replenishment order when it needs restocking, but can find the needed products at the best price, and order and pay for them without manual effort or the involvement of its owner.

The advantages of BitCAD:

• Low barriers to entry for users to conduct the transaction.

• The “reputation” of participants performance on past smart contracts will help the highest-performing users to demand premiums.

• Smart equipment can replace human contracting parties for certain transactions, as in our example of the vending machine.

• Devices on the IoT can communicate with smart contracts to keep track of the status and state of smart contracts for settlements. Smart shipping containers could, for example, automatically sell their surplus capacity.

• Faster settlements using cryptocurrencies.

Capabilities of BitCAD:

• Audit trails: reflection of the successive actions for each subject in BitCAD provides undeniable evidence for the movement of goods and opens up new opportunities for "Live Price".

• Real-time negotiation: smart contracts continuously query all other nodes on different platforms for the best pricing, delivery times, and other terms and conditions.

• Supply chain visibility and traceability: via production records, for example, one can trace whether defective goods were made.

• Tapping data from IoT: easily tracked and authenticated data from IoT gives business more and better data about their products, enabling them to improve quality.

• IP management in product development: makes it easier and less expensive to securely share intellectual property.

1.2 BitCAD’s governance: Multistakeholder model

BitCAD is a public-benefit organization. Its staff operates the smart contracts systems, coordinates allocation and assignment of unique identifiers, accredits delegates of industries, and helps facilitate the voices of volunteers worldwide who are dedicated to keeping the Smartnet secure, stable and interoperable. BitCAD promotes competition and helps develop Smartnet policy.

At the heart of BitCAD policy-making is what is called a “multistakeholder model”. This decentralized governance model places individuals, industry, non-commercial interests and government on an equal level. Unlike more traditional, top-down governance models, where governments make policy decisions, the multistakeholder approach used by BitCAD allows for community-based consensus-driven policy-making. The main idea is that governance should be borderless, multilateral and open to all.

The most important feature of BitCAD is open public discussions preceding every important decision, be it an operational (budget, development strategy), tech (security concerns) or administrative (amending major policies) issue.

Public discussions will be held according to the following routine:

- Every issue will be up for discussion for no less than 21 days.
- Following the commenting period the replying period begins also spanning at least 21 days.
- If during the commenting period no substantial comments were received the replying period will not commence.
- During the replying period only previously gathered comments will be addressed, new venues of discussion will no longer be available.

While the BitCAD Board of Directors has the ultimate authority to approve or reject policy recommendations, Business Organizations are responsible for developing and making policy recommendations to the Board.

Worldwide BitCAD community is the main self-educating force, where each participant can take a direct part in the development and promotion of the platform. Users across the World are encouraged to create local BitCAD organizations. The end goal is to have at least one such entity per country. BitCAD’s governing body comprises of the Board of Directors and inclusive and transparent advisory committees as shown in Fig 1.

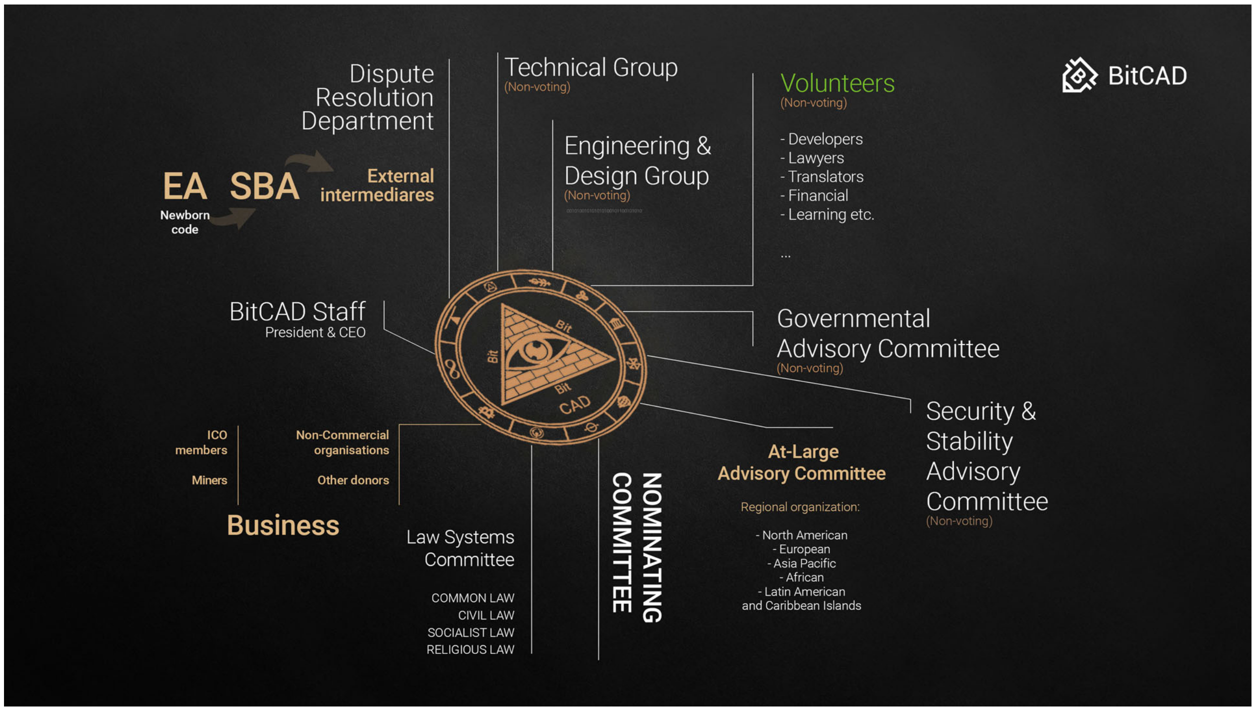


Figure 1: Multistakeholder model BitCAD

BitCAD's Board of Directors

BitCAD's Board of Directors is a multinational body in charge of policy making and day-to-day platform management. It is run by 16 voting members (including BitCAD's president) and five non-voting representatives.

Four voting members are appointed by Business Support Organizations (BSO). Law Systems Committee (LSC) and At-Large Advisory Committee (AAC) elect correspondingly two and one voting members. The remaining 8 voting members are voted in by the Nominating Committee.

Non-voting members are appointed by Governmental Advisory Committee (GAC), Security and Stability Advisory Committee (SSAC), Technical Group and Engineering Design Group. They provide technical advice during the decision making process.

Nominating Committee (NC)

NC includes leaders of Volunteer organizations. It elects a number of officials, including 8 voting board members, some members of At-Large Advisory Committee and representatives of Business Support Organizations and Law Systems Committee.

Business Support Organizations (BSO)

BSO is an inclusive community of professionals representing various fields of expertise. It is responsible for developing global policies and passing them to BitCAD's Board for approval. It also provides the community with a platform for exchanging vital information and discussing global issues.

BSO Council is a team of business support specialists tasked with overseeing all BSO activities, developing platform's general policies and advising BitCAD's Board of Directors. The Council officially approves four voting board members once they have been elected by all BSO members.

BSO Council includes 18 representatives. 15 representatives are voted in by BSO community (three per region) and 3 are appointed by the Nominating Committee.

At-Large Advisory Committee (AAC)

AAC is the core institution of BitCAD that represents interests of BitCAD's users and advises the Board of Directors on policies and activities that affect individual users.

AAC leadership comprises of five members appointed by the Nominating Committee and ten members elected by the community (two per region). The five members appointed by the NC have to represent the five regions of the world as well.

Governmental Advisory Committee (GAC)

GAC advices BitCAD on various public policies, especially when discrepancies between local regulations and BitCAD's policies and activities arise.
It is composed of appointed representatives of local governments and multinational organizations. Governments and other institutions will be able to petition BitCAD for GAC membership.

GAC is a consensus-driven organization. Not like the UN, GAC will make decisions upon reaching unanimous consensus. If unsolvable disagreements arise, the committee passes all the voiced opinions to the Board of Directors.

Law Systems Committee

LSC is an advisory committee tasked with resolving issues connected to international legal systems, including standardization of smart contracts and development of a universal digital legal language. LSC will also appoint two voting board members.

Twenty three committee members will be drawn from top experts in their corresponding legal fields and divided into two houses (akin to British Parliamentary model). One house will represent the industries and the other will speak for independent business and non-profits. This checks and balances system will prevent any major player from hijacking the committee.

Security and Stability Advisory Committee

SSAC is an advisory committee consisting of engineers, scientists and security specialists. This committee keeps the platform safe from external and internal attacks.

Members of SSAC are appointed by the Board of Directors. Any user can get onboard given he passes the rigorous selection process.

Multinational Volunteer Community

BitCAD volunteer community is the main driving force behind platform’s leadership. Every volunteer (or a group of volunteers) can participate in BitCAD’s activities developing and suggesting policies, procedures and business models. Active and productive volunteers will be able to eventually join the ranks of various committees and organizations that are a part of BitCAD’s multistake governance model.

BitCAD will welcome participation of volunteers with a wide range of skills: development, law, translations, accounting, marketing, etc. A Teaching & Learning online platform will be created within BitCAD which will allow for quick adaptation of new users who want to acquire new skills or fruitfully apply the existing ones.

BitCAD Staff

BitCAD staff is responsible for executing and implementing policies developed by the BitCAD community and adopted by the BitCAD Board. BitCAD Staff is a team of professionals in various fields from accounting and HR to tech support and engineering.

Tectum

Tectum is BitCAD's twin project that aims to facilitate BitCAD's global integration via APIs and works on creating a decentralized marketplace.
Tectum will grant smart contracts unhindered access to the market. It will help BitCAD's users purchase goods on all trading platforms, seek out and seal deals, resolve logistics issues and optimize business processes.

Currently Tectum operates in Russia and China. Tectum's ambition is to open its authentication centers in every country in full compliance with local laws. This will allow residents to improve their experience with the customs and tax office, and to interact with financial institutions and marketplaces more easily.

2 Smart Contracts - a better way to facilitate business processes

In 1996, Nick Szabo described a smart contract as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises." While the technology available to support smart contracts has evolved considerably since then, this definition continues to capture the essence of what a smart contract is and does.

Smart contracts are typically deployed on a blockchain (although it is possible for other platforms to host them too). Within a blockchain view of this, smart contract program logic sits within a "block." A block is a software generated container that bundles together the messages relating to a particular smart contract. Those messages may act as inputs or outputs of the smart contract programing logic and may themselves point to other computer code.

2.1 The semantics of contract

Part of our remit is to consider the semantic construction of a contract — i.e. what is the "meaning" of a contract? Does it have more than one meaning? How should a contract be interpreted? We start with a simple semantic framework and view a legal contract as having two interpretations:
1. The operational semantics: this is the operational interpretation of the contract, which derives from consideration of precise actions to be taken by the parties. Thus, it is concerned with the execution of the contract.
2. The denotational semantics: this is the non-operational legal interpretation (or "meaning") of the entire contract, including all of its obvious constituent parts and including any other legal

documents that it references. This is the meaning that would be given to a contract when a lawyer reads the contract.

These two semantics do not consider different parts of the contract — they are both interpretations of the whole contract, but with different aims.
A contract may comprise several documents, and the process by which these documents are agreed may be complex. The denotational semantics of even quite straightforward contracts can be very large and complex, yet by contrast the operational semantics might be simple and easily encoded for automatic execution.

The operational semantics dictate the successful execution of the contract to completion. If a dispute arises, then the denotational semantics of the contract typically dictate what happens next — i.e. in the context of the rights and obligations of the parties, the specification of what remedies shall be applied in the case of partial performance or non-performance by one party.

The greater part of a legal contract may often be devoted to defining the obligations and liabilities of the parties in the event of a problem with execution. Sometimes, the actions to be taken in case of a material breach of contract are expressed precisely; however, this is not always the case and dispute resolution may require a protracted process of negotiated settlement, arbitration or court proceedings.

Furthermore, it is important to realise the important role of law. It is not possible to take literally the doctrine that all one needs to know about a contract is contained within “the four corners of the document”. A lawyer would read and understand the contract in the context of the governing law — i.e. each legal document must be interpreted according to the relevant law (corporate law, consumer law, etc) of its stated or inferred jurisdiction, and therefore the semantics of that law must also be understood. It should be noted that the issue of law relates not only to the denotational semantics but also to the operational semantics — for example, trading with certain countries may be illegal due to government-imposed sanctions.

Given this semantic framework for the legal contracts that underpin financial instruments, we can derive a different perspective of smart contracts:
• smart contract code focuses exclusively on execution and therefore concerns itself only with the execution of those operational semantics that are expressed in the code, whereas
• smart legal contracts consider both the denotational and operational semantics of a legal contract, whose operational semantics must then be executed (possibly by smart contract code).

2.2 Smarter smart contracts

“Tamper-proof” execution is typically described in terms of distributed networks of computers that are unstoppable and in a technological sense cannot fail regardless of malicious acts, power cuts, network disruption, natural catastrophes or any other conceivable event. With such a system, it is assumed that a software agent, once started, could not be stopped. For truly “unstoppable” software agents, code must be embodied to take the appropriate action in response to various dynamic states that might occur (such as another party not having sufficient funds to execute a required payment). In a normal system, the software agent might abort and the wrong-performance or non-performance by a party would be enforced by traditional means; but in a truly unstoppable "tamper-proof" version of the system, all such possibilities would have to be anticipated and appropriate actions determined in advance, so they are no longer deemed wrong-performance or nonperformance but are instead anticipated states of the system with known resolution.

Although some groups are actively pursuing tamper-proof smart contract code, our preference is for smart legal contracts that are enforceable by traditional legal methods for reasons including:
• In a system with enforcement by tamper-proof network consensus, there would be no "execute override" provisions. Agreements, once launched as smart contract code, could not be varied. But it is quite common for provisions of an agreement to be varied dynamically — for example, to permit a favoured client to defer paying interest by a few days, or to permit a payment holiday, or to permit the rolling-up of interest over a period. Unless every possible variation is coded in advance, none of this would be possible in a tamper-proof system.
• Enforcement by network consensus can only apply to the execution of obligations, or the exercising of rights, that are under the control of the network. However, objects and actions in the physical world are unlikely to be under full (if any) control of the network.
• Smart contract code that involved payments would require posting collateral to be completely automated. This locking-up of collateral would lead to a serious reduction in leverage and pull liquidity out of markets. Markets might become more stable, but the significant reduction in leverage and consequent market decline would be strongly resisted by market participants.

2.3 Smart Contract Constructor & Templates

BitCAD's Smart Contract Constructor provides a framework to support complex legal agreements for financial instruments, based on standardised templates. It uses parameters to connect legal prose to the corresponding computer code, with the aim of providing a legally-enforceable foundation for smart legal contracts. It also facilitates automated execution of the contract and, in the event of dispute, provide a direct link to the relevant legal documentation.

Complex sets of legal documentation can be augmented with the identification of operational parameters that are key to directing the executable behaviour of the smart contract code (in this paper we call these "execution parameters") — the smart contract code is assumed to be standardised code whose behaviour can be controlled by the input of such parameters.

Here we explore the design landscape for the implementation of Smart Contract Templates. We observe that the landscape is broad and that there are many potentially viable sets of design decisions. We therefore propose that a new domain-specific language should be developed to support the design and implementation of Smart Contract Templates. Development of this language is already underway. We call this common electronic legal language LawTech or "CLACK" (Common Language for Augmented Contract Knowledge).

The intention is to interface with a wide range of execution platforms. Smart legal contracts could potentially be executed as software agents operating on distributed ledgers (such as Corda, Ethereum, Hyperledger, etc.).

2.2.1 Templates and Parameters

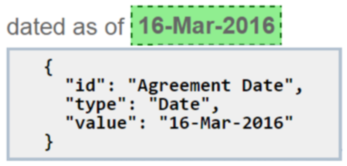
A template is an electronic representation of a legal document as issued by a standards body — for example, by the International Swaps and Derivatives Association (ISDA). A template contains both legal prose and parameters, where each parameter has an identity (a unique name), a type, and may (but need not) have a value. Agreements are derived from templates, and both the legal prose and parameters may be customised during negotiation. Values are mandatory for all parameters in a signed agreement.

An agreement is a fully-instantiated template (including any customised legal prose and parameters). The customisation of legal prose and parameters at this stage is commonplace and results from negotiation between the counterparties. We also observe that it is common for agreements to comprise multiple documents such as Framework Agreements (e.g. a Master Agreement) with various Annexes (e.g. a Schedule) and Credit Support Documentation (e.g. a Credit Support Annex). Thus, the legal prose of an agreement will be derived from that of the template, but need not be identical, and similarly the parameters of the agreement will be derived from the template but need not be identical.

Deriving the set of execution parameters may be complicated by three factors:
1. It is common for execution parameters to be embedded in the legal prose — identification of such parameters would initially be undertaken by visual inspection and be aided by a graphical user interface.
2. Some of the values identified as “parameters” in the agreement (and in the template) may not have an operational impact and therefore should not be included in the set of execution parameters.
3. It is possible for a parameter to be defined (given a name) in one document, given a value in a second document, and used (e.g. in business logic) in a third document. Although parameters need not have values in a template, they must have values in a signed agreement. All of an agreement’s parameter values are a critical part of the contract as they directly reflect the business relationship between parties and those that are execution parameters influence the operation of the contract.

2.2.2 The design landscape for Smart Contract Constructor

Most parameters in existing legal document templates have simple types, such as date, number, etc. These are “base” or “primitive” types and, as an example, Figure 2 illustrates the identification of a date in a master agreement; once highlighted and annotated, the name (“Agreement Date”), type (“Date”) and value (“16-Mar-2016”) of this parameter will be passed to the executable code.


dated as of 16-Mar-2016

```
{
  "id": "Agreement Date",
  "type": "Date",
  "value": "16-Mar-2016"
}
```

Figure 2

An editor permits a date in the legal prose to be highlighted, and then annotated to denote a simple parameter. The parameter has a name “Agreement Date”, type “Date” and value “16-Mar-2016”.

It is not necessary for parameters to be restricted to base types. It is very likely that values of more complex types, such as lists, will also need to be transferred to the executable code.

The passing of parameters to the executable code is necessary because of the desire to use standardised code. It would, for example, be theoretically possible to generate entirely new code for every trade and in this case there would be no need for parameters. The number of parameters, and the complexity of the types of those parameters, will typically increase as the code becomes more generic.

Beyond parameters with base types and more complex types such as lists, parameters can also be expressions containing references to other parameter names. Unless those other parameter names are defined within the expression, the expression is e ectively a function. Where a function is passed as a parameter, this is known as a “higher-order” parameter and the receiving code is known as a “higher-order” function.

The use of parameters may not only be used to support greater standardisation of code. In the far future, we may see an increasing use of a formally structured style of expression embedded in legal prose; if all business logic in legal prose could be replaced with arithmetical or logical expressions, such as the higher-order parameters discussed in the previous paragraph, this would lead to reduced ambiguity in legal prose and fewer errors in translating legal prose into execution parameters. Such adoption of formal logic into legal prose would require such formal constructs to gain acceptance in the courts and to be admissible as evidence of the intentions of the parties.

Figure 3 Illustrates our view on how the sophistication of parameters and their role in Smart Contract Templates may evolve in the future.

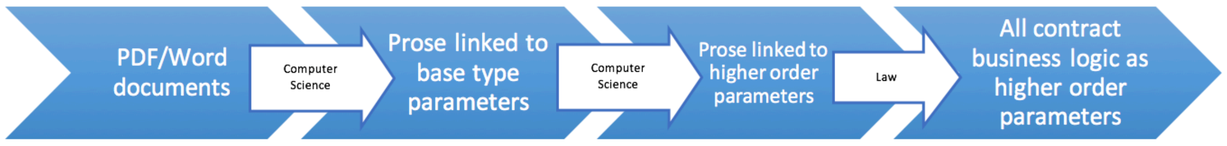

PDF/Word documents
Computer Science
Prose linked to base type parameters
Computer Science
Prose linked to higher order parameters
Law
All contract business logic as higher order parameters

Figure 3 Evolution of legal prose and parameters

Execution parameters may become more sophisticated in the future, evolving from just simple base type parameters to also include more complex higher-order parameters. In the far future, if the encoding of business logic used in the parameters becomes acceptable to lawyers and admissible in court, then it could potentially replace the corresponding legal prose.

In the previous subsection, we observed that the passing of parameters to executable code is necessary because of the desire to use standardised code. This is important for efficiency reasons as different smart contract would otherwise have to be built, tested, certified and deployed for every different trade. The effort is reduced if such code can be standardised with parameters being passed to each invocation of that code.

This therefore drives a desire for greater genericity of code, which can be enabled by passing more parameters, and/or more sophisticated parameters (with more complex types). Yet despite the gains of standardised and more generic code, there remains the problem that each bank currently manages its own distinct codebases. If smart contract code could be common (i.e. shared) then it could be built, tested and certified once — and then utilised by every counterparty.

One possible evolutionary route could build upon the use of common utility functions — programs that are already very nearly identical in all counterparties.

2.3 Further Work: universal language for legal smart contracts

A good example is the potential for greater straight-through-processing from contract to execution. Currently, lawyers draft legal contracts, which are then negotiated and changed by possibly other teams of lawyers, and then operations staff inspect the contract documents and/or other materials to identify the execution parameters that are then passed to code that may have been written some time ago.

This raises several issues:
• Can we be absolutely certain of the meaning of the contract? Are all parties truly agreed on the meaning of the contract, or do they instead each have a subtly different understanding of what the contract means?
• Can we be certain that all execution parameters have been identified by the operations staff ? Can we be certain that those parameters that have been identified are indeed operationally relevant? And can we be certain that their names, types and values been faithfully transcribed?
• After the parameters have been passed to the code, and the code runs, can we be certain that the code will faithfully execute the operational semantics of the contract? and will it do so under all conditions?

Solution would be to develop a formal language in which to write legal documents — i.e. contract documents — such that the semantics would be clear and the execution parameters could automatically be identified and passed to standardised code (alternatively, new code could be generated). This formal language would:

1. derive a number of important qualities from well-designed computer programming languages, such as a lack of ambiguity, and a compositional approach where the meaning of any clause can be clearly deduced without reading the rest of the document; and
2. be simple and natural to use, to such an extent that a lawyer could draft contracts using this formalism instead of using traditional legal language.

The former aspect has already received considerable attention in academia and beyond. In contrast, the latter aspect is likely to be by far the greater challenge.

Another challenge is whether such a contract, written in a computer-like language, would be admissible in court as a true and faithful representation of the intentions of the parties. Issues of signature and tamper-evident documents are easily solved, yet whether a court would accept the definitions of the meanings of phrases in such a contract is not immediately clear. This problem could be solved in two ways:
1. As a first step, the language could generate a document version of the contract in a more “natural” legal style, with the expectation that this document would be admissible in court.
2. Eventually, further research in domain-specific languages and law could result in a new formalism itself being admissible in court.

As a result of this complexity, we were motivated to define a common language to support the specification of different solutions across the design space of Smart Contract Templates.

Initially the language will help in the specification of different design choices and in the building of prototypes. In general, the language should be as flexible as possible to support a wide range of requirements. An initial set of requirements has been sketched as follows:
• It should provide support for both legal prose and parameters.
• It should support different internal structured formats, such as XML.
• It should support the output of execution parameters in a variety of formats, such as FpML.
• It should support contracts that comprise multiple documents.
• It should manage multiple agreements being instantiated from a single template (and hierarchies of templates).
• It should permit parameters to be defined in one document, given a value in a second document, and used in a third document.
• It should support a wide range of parameter types, including higher-order parameters.
• It should support increasing standardisation and sharing of common code.
• It should support multiple execution platforms.
• It should support full interaction with, and increasing automation of, legal prose.
• It should support digital signing of contracts, the construction of a cryptographic hash of the contract, and the use of that hash as an identifier for reference and recovery of the smart contract.

The CLACK language is being specified and prototyped to support Smart Contract Templates. Next steps include fully specifying intra-document and inter-document referencing including ambiguity and conflict resolution strategies, syntax and semantics of expressions within higher-order parameters, etc.

There are many open questions for the future. We have explored some of these questions in this paper, but we will end with one more: is it possible to provide straight-through-processing of financial contracts, with full confidence in the fidelity of the automated execution to the operational semantics of the contract? This, of course, will require substantial work from academia working with lawyers, standards bodies and the financial services industry.

3 Smart Oracles - connecting realities

Smart oracles provide a simple, flexible way to implement "smart contracts", which encode business logic, laws, and other agreed-upon rules. Smart oracles build on the idea of oracles, or entities that provide smart contracts with information about the state of the outside world, and combine information gathering with contract code execution. In such a system, rules can be written in any programming language and contracts can interact with any service that accepts cryptographically signed commands. This includes, but is not limited to, cryptocurrency networks.

Some smart contracts systems, including the one built into Bitcoin, are strictly deterministic. In order to interact with the real world, these systems rely on cryptographic signatures submitted by outside systems called "oracles."

Oracles are trusted entities which sign claims about the state of the world. Since the verification of signatures can be done deterministically, it allows deterministic smart contracts to react to the (non-deterministic) outside world.

3.1 From Oracles to Smart Oracles

The concepts of smart contracts and oracles have existed for some time. Several earlier designs (including Bitcoin) have relied on executing the contracts within consensus networks, leading to the requirement that their execution be deterministic. In this paper we aim to show that placing contract execution in the hands of smart oracles generalizes and simplifies the system significantly.

Recently, the advent and explosion of interest in cryptocurrencies has spurred a resurgence of interest in smart contracts. Math-based currency networks provide an important building block for smart contracts: valued digital assets that can be transferred with a cryptographic signature. Assets in protocols are owned by accounts identified by public/private key pairs. Payments are executed when the transaction carries a cryptographic signature that could only have been produced by the holder of the account’s private key. Smart contracts can trivially create such cryptographic signatures and, thus, be designated the partial or sole owner of any type of digital asset.

Unfortunately, cryptocurrency developers have found it challenging to design a system that encompasses both a powerful smart contracts language and a robust consensus system. Bitcoin scripts allow for simple logic to be encoded and executed on the Bitcoin network. However, encoding advanced logic and executing untrusted code have proven more complicated to integrate.

We argue that it is possible to implement powerful smart contracts in a secure and trustworthy manner without increasing the complexity of existing consensus networks.

The execution of untrusted code should be decoupled from the consensus databases and other services that track and transfer asset ownership. The separate contract system can handle untrusted code execution and interact with the consensus databases through cryptographic signatures. These signatures are already native to consensus protocols so no modifications are necessary. Decoupling contracts from consensus networks gives the added benefit that contracts can interact with multiple networks at once as well as virtually any type of online service. This means that a single smart contract could interact with Bitcoin and Ripple, web-based services like PayPal, Google, Ebay, etc. or even other Internet protocols, such as SSH, LDAP, SMTP and XMPP.

If the contract execution is decoupled from existing systems, where should the code be run? This is where smart oracles come in.

Most proposals for smart contracts, even those that are internal to consensus networks like Bitcoin, depend on independent entities to inform contracts about the state of the outside world. Bitcoin contracts rely on "oracles" to attest to facts from the outside world by introducing signatures into the network if and only if specific conditions are met. Smart oracles takes this concept a step further to place the untrusted code execution in the oracles' hands. The smart oracles are trusted or semi-trusted entities that can both provide information about the outside world and execute the code to which the contracting parties agreed.

3.2 Implementing Smart Oracles

Smart oracle implementations could take many different forms. In the following sections we outline some of the elements we see as essential for most, if not all, smart oracles. Namely, the key components are: securely identifying code, sandboxing code, oracle APIs, contract hosting and billing models, and contract clients.

Once contracting parties have agreed on the terms of their arrangement they must translate the rules into code. It is crucial that the parties inspect the proposed code and ensure that it represents the business logic to which they agreed to be bound. It is equally important that they can easily verify that the code uploaded to the smart oracle(s) is exactly that which they already inspected. This is where deterministic code compilation, hashing, and code reuse with modules come in.

All parties to a contract have a large stake in ensuring that the final, machine-executable code represents the logic they agreed to. For compiled languages, this means that the source code must be shared along with a reproducible process to compile it to machine code, such as with Gitian. For interpreted languages it is sufficient to share the source code. Either way it is critical that participants agree upon the final instructions that will be executed by the smart oracle.

Cryptographically secure hashes are a convenient way to identify agreed-upon binaries or source code files. Hashing functions take arbitrary amounts of data as inputs and produce a short, fixed-length string of characters. For practical purposes, this "hash" can be used to uniquely identify any text or data.

Although it might not be strictly necessary, we recommend using collision-resistant hash functions. This means that it would be impractical to attempt to find two inputs with the same output hash. It would be exceptionally difficult to produce two pieces of working code with the same hash, even using a hash function that is only second preimage resistant. However, it would cause serious problems if someone could create two distinct contracts with the same hash. Therefore we recommend hash functions that are second preimage and collision resistant.

Traditional contracts often share common "boilerplate" elements and smart contracts are no different. Any smart oracle system is likely to offer some form of code reuse. This adds convenience as well as security.

Many contracts will have relatively simple and easy to understand logic built on top of well-known and widely used modules. Modules could encompass basic functionality, such as mechanisms to connect to Bitcoin or another systems. They could also include more advanced features such as a standard auction, escrow, or bond implementation. The logic would likely be widely used and verified by many independent parties.

The heart of the smart oracles concept is the ability for users to agree on the code of a contract and then to upload it to a trusted third party or parties for them to execute it. Smart oracles must be able to safely execute the user code, which is untrusted and may actually be malicious. Oracles must protect their own systems and the integrity of the other contracts they are running.

One of the most flexible pieces of smart oracles is the billing system that allows contracting parties to pay the smart oracles for the contract execution. The billing system is entirely decoupled from the core system design so smart oracles can accept any payment methods they choose, from credit cards to Bitcoin. The decision to require costs to be prepaid or billed after the fact is also left entirely to the oracles' operators.

Smart contracts are an exciting new frontier for technology, business, and law. Smart oracles combine the idea of an oracle, which provides information about the real world, with a sandboxed code execution environment. It is independent of existing distributed networks such as Bitcoin and Ripple and can interact with any Internet-based service, including all distributed consensus databases. Separating the untrusted code execution from distributed networks reduces the complexity and thus increases the security of both systems.

Smart oracles in general open up new possibilities for developers, entrepreneurs, and enterprising legal and financial professionals. Agreements that previously required lengthy legal contracts can be translated into code and run automatically by smart oracles. Smart contracts hold the potential to empower people to build a fairer, more affordable and more efficient legal system and smart oracles are one of the simplest ways to realize that dream.

4 Dispute Resolution

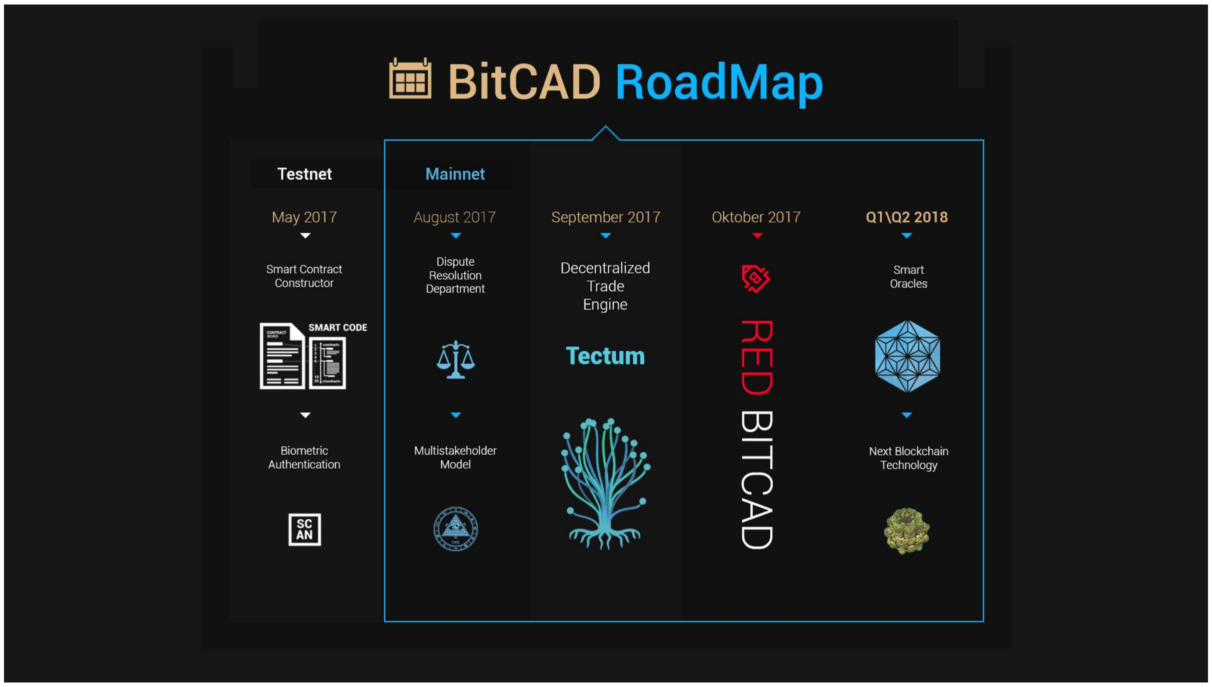
The Dispute Resolution Department is a three-stage automated process area consisting of electronic and story based arbitration, or independent and impartial hired person with competence to resolve problems and complaints regarding the decisions, actions or omissions of BitCAD and the organization's management, as well as unfair treatment of the participant Communities from employees, board or representatives.

Electronic arbitration is a fully automated method of dispute resolution using Internet technologies. Electronic arbitration can be applied to resolve a wide range of problems within the BitCAD platform - from interpersonal disputes, including consumer disputes, to interstate conflicts. Electronic arbitration has a great potential for resolving conflicts in the field of e-commerce. The newborn code model is embedded in the basis, when this code evaluates the parties according to their behavior and the history of interaction during the whole transaction. When the parties don't agree with the computer decision, they can solve the outcome of the case online with a voluntarily chosen randomness in the game additions form of certain sets.

Story based arbitration is a model of a decentralized court with a multi-level scheme of randomly elected judges sitting in a decentralized judicial system. Access to the history of events and all documentation is opened by each party separately by providing a private key with a ghost of the necessary arguments. In some cases, access to encrypted video recordings of the parties negotiations can be provided only if there is private key from each parties, depending on the originally selected conditions for the commencement of negotiations.

Ombudsman is an independent, impartial and neutral person contracted to BitCAD, with jurisdiction over problems and complaints about decisions, actions or delay by BitCAD and the supporting bodies, as described in this Guide. He is an advocate for fairness. The Ombudsman investigates complaints and attempts to resolve them. Emphasizing mediation as a means for resolving issues, he will often convene a mediation between parties. The Ombudsman does not have the power to make, change or set aside a policy, administrative or Board decision, act, or omission. Ombudsman does have the power to investigate these events, and to use dispute resolution techniques like mediation to resolve them. Arbitration courts, private attorneys and international advisors are all eligible to be contracted as Ombudsmen.

Roadmap



MAY 2017 - Testnet Launch

- Biometric Authentication
- Constructor of Smart-Contracts & DAO

AUGUST 2017 - Mainnet Launch

- Dispute Resolution Department
- Multistakeholder Model (self governance system for BitCAD)

SEPTEMBER 2017

- Decentralized Trade Engine **Tectum** (application & API's launch)

OCTOBER 2017

- **Red BitCAD** (The platform is ready to launch. Any person or company can start and develop business on the platform, accessing announced features, including reputation tracker.)
- ***Dark BitCAD*** *(Platform's integration within Darknet communities)*

Q1/Q2 2018

- Smart Oracles
- Next Blockchain Technology (experimental technology based on fractal math, machine learning and quantum cryptography. Employs supercomputers for business-logic and virtual business ecosystems modelling.)

Glossary

Blockchain

Blockchain is a distributed ledger that continually checks the security and integrity of each transaction or data entry. Blocks chained by hash values and incentivized by proof of work (or alternative proof algorithms) provide a foundation for distributed trust in blockchain.

While it's best known in the financial services world as an infrastructure that enables trusted financial transactions between parties without the need for a third party such as a bank, it can be used in any industry to enable faster, less expensive transactions and to support more agile supply chains that would be impossible otherwise.

Business processes

Business processes are complex, multi-tiered combinations of various types of tasks aimed at creating a certain product (or service) and its market integration. Producing even a single component of a single product may involve a myriad of transactions, ranging from requests for quotes to the transmission of purchase orders and engineering change notices. Each transaction type may require different financial and regulatory intermediaries, as well as its own contract and trust relationship among the parties. With The immediate and low-cost assurance of trust, BitCAD can unleash disruptive innovation by allowing any users to instantly find one another and begin a relationship.

Smart Contract

In 1996, Nick Szabo described a smart contract as “a set of promises, specified in digital form, including protocols within which the parties perform on these promises.”

In other words, a smart contract is an agreement whose execution is both automatable and enforceable. Automatable by computer, although some parts may require human input and control. Enforceable by either legal enforcement of rights and obligations or tamper-proof execution.

DAO

DAO (Decentralized autonomous organization) is a new paradigm of economic cooperation. Unlike traditional companies, DAO is decentralized (it has no owner and prefers horizontal governance structure) and autonomous (financial transaction record and program rules are maintained on a blockchain, maintained by the entire community). Most business processes in DAOs are automated and executed through rules encoded as computer programs called smart contracts.

Live Price

BitCAD's signature feature which allows for monitoring the price of any product in online or brick-and-mortar shops in real time. This future tech relies on ubiquity of blockchain in order to gather reliable market data globally.

Live Price data is a godsend for market analysts and also a boon for customers enabling them to monitor realistic prices and not fall for speculatory schemes.

Smartnet
Internet 2.0. The next stage of the Internet evolution marked by decentralized networks and worldwide smart contracts integration.