Architecture

Version 2
September 2016
[DRAFT]

Joe Roets
- j03 -
joe@dragonchain.org

Dragonchain Architecture

The purpose of this document is to outline and communicate the architecture and design of a blockchain platform which will allow ease of integration for real business applications. In the author's opinion, there is a growing need for simplified blockchain integration. The decentralized and singular approach to blockchain implementation is sometimes at odds with the real business need to protect information and control business processes. This document seeks to shed light and provide examples for the successful implementation of enumerated blockchain architectural elements.

Architectural Goals

1. Ease of integration of existing systems
2. Ease of development for traditional engineers and coders unfamiliar with blockchain, distributed systems, and cryptography
3. Client server style and simple RESTful integration points for business integration
4. Simple architecture (flexible and usable for unforeseen applications)
5. Provide protection of business data by default
6. Allow business focused control of processes
7. Fixed length period blocks
8. Short/fast blocks
9. Currency agnostic blockchain (multi-currency support)
10. No base currency
11. Interoperability with other blockchains public and private
12. Adoption of standards as they become available (see W3C Blockchain Community Group blockchain standardization)

Architectural Elements

Abstraction of Proof

In Bitcoin and most other cryptocurrencies, we witness the use of "Proof of Work" (PoW) algorithms as a basis for consensus in a "trustless" system. In this architecture, "proof" will be abstracted and may be implemented in one or more ways for a given blockchain. For some uses, one may desire to use a trust based system, for instance, in a fully private blockchain system. One may also find value in a hybrid proof configuration that would see trust applied alongside limited Proof of Work to add additional security against attack (i.e. a potential attacker would not only need to compromise or attain a set of private keys, but would also need to perform computation to accomplish the configured Proof to reassemble a given blockchain.

Implementations:

1. Trust (Default)
2. Proof of Work (PoW)
3. Proof of Stake (PoS)
4. Other as yet determined algorithms

Given such an abstraction, a user would be able to configure one or more simultaneous proofs to suit business need, whilst a system developer may build new proof implementations as blockchain technology progresses.

It is possible to conduct PoW even in the case of fixed time length block constructions (see block construction discussion elsewhere in this paper) by spanning one or more proof implementations across blocks. As an example, let's say that we have a particular use case that requires higher than normal security. If we assume that Trust is implemented by default, but we desire to configure some amount of trustless verification, we may wish to configure some level of Proof of Work on our blockchain. Depending upon the difficulty level configured, and given the nature of PoW algorithms, we may not see a PoW solution for every block. Some blocks would have no PoW, and the PoW answer may only appear occasionally. In such a case, it may be reasonable to configure two or more levels of PoW. A higher difficulty proof may be tuned to appear approximately every 20 minutes, and a lower difficulty proof may be tuned to appear approximately every 2 seconds. In the same manner other proofs such as PoS may be applied simultaneously within a single chain. An interesting philosophical point is that such proofs may be used in competition against a future attacker rather than as competition with other miners for a block reward.

Checkpointing and Proof of Existence

Another element in the abstraction of proof is the further ability to hybridize by checkpointing into other (public) blockchains. This can be seen as a first level or simple interoperability between blockchains, public or private. Of particular potential value, is the ability to ascertain risk by measuring a public blockchain's attributes. That is, if tied to a public blockchain which uses PoW such as Bitcoin, the system can estimate the amount of hashpower that has been applied since the checkpoint and even extrapolate that compute power to dollars spent. With this, a risk unit may be developed that shows how much compute power would be needed (and how much that would cost) to calculate the percentage likelihood of success in an attempt to counterfeit a given artifact (e.g. a transaction of high value). In the same manner, tying a checkpoint to a public blockchain based on PoS, the system could measure the amount of assets that must be held (and likely sacrificed) in order to counterfeit the transaction in question. See discussion of Level 5 verification below for more information.

Transaction Definition

A transaction is the basis with which all events or data transfers are recorded within the blockchain platform. The system should define a flexible and extensible standardized

transaction structure.

Implementations options:

- JSON with standardized structure
- JWT (JSON Web Token)
- Other encoded structure (with supporting libraries for multiple languages)

Header

Contains system or network defined standard standard metadata fields for the transaction.

Example fields:

- Transaction ID
- Transaction type
- Transaction class
- Create timestamp
- Transaction timestamp
- Origin ID
- Business organization and/or organization taxonomy
- Actor
- Entity

Payload

Arbitrary structure and content, defined at the business level and implemented or controlled within the level 1 approval code.

Some level of structure within the payload (e.g. fields and structures) may be implemented as network-wide templates to be utilized and noted based upon the optional `transaction class` header field. This will allow nodes to implement some needed behavior defined at the Enterprise or network level, as well as simplification of capabilities such as currency. See below for examples of transaction class.

Signature

Portion of the transaction holding a cryptographic signature to allow parties to prove the source and/or that the contents of the transaction are unaltered since the signing (i.e. tamper evident). The signature within the transaction should not be required from the transaction source (e.g. client or 3rd party system) as some clients may not be cryptographically enabled or aware of the blockchain platform. Alternatively, a client system may for example invoke a process of multi-party signing prior to transaction submission. Either way, the blockchain platform node's Transaction Service component should cryptographically sign any inbound transaction that it accepts for processing (with its configured key pair).

Perceived requirements:

- The structure should allow for multi-party and nested signings
- The signature should hash all fields within the signature structure itself minus hash and signature to make the signature itself tamper evident
- The structure should see re-use in the verification record (block verification) signing process

Fields:

- Signatory
- Hash
- Stripped Hash (for transaction signatures only - a hash of the full transaction with the payload stripped - to allow level 2+ nodes to validate a transaction even when the payload is not available)
- Public Key
- Signing Timestamp
- Signature
- Child Signature (optional)

Implementation options:

- JWS (JSON Web Signature)
- Custom JSON

Classes

Some examples of possible transaction classes are:

- Default (custom Level 1 business payload structure)
- Currency
- Provisioning
- Network communications
- Marketplace transaction
- Information interoperability (foreign blockchain currency or information payload)

Block Definition

Block definition may see multiple implementations, although there are more or less common elements involved such as:

- Block ID
- Timestamp
- Transactions
- Hash of prior block
- Proof (e.g. PoW, PoS) artifact
- Signature
- Block period

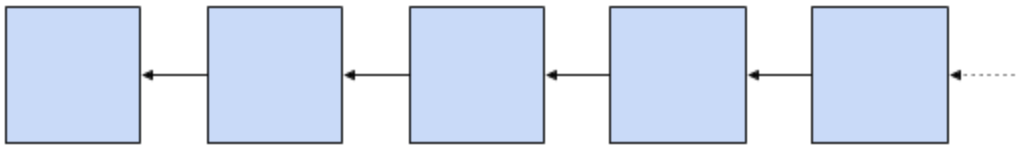
- Verification attributes

An interesting part of block definition design that is often not considered of is the question of *when* the block is formed. In Bitcoin or other PoW systems, a block is formed when the PoW algorithm is solved for the current network difficulty. This may happen after 30 seconds or 30 minutes. It is variable and random, but the network seeks to tune the difficulty in order to tune the average block time to 10 minutes.

In trust based systems, there would be no absolute need to maintain a variable time based block. In many real world systems, it is in fact seen as a detriment to its use that Bitcoin cannot offer a fixed or faster average block time. In this architecture, we may desire a more appropriate block time, such as a fixed time in seconds (e.g. 5 seconds). The definition of such fast and fixed block times leads to a question of consensus, that is, how does the entire network come to consensus every 5 seconds? With context based verification and the concept of a “blockchain of blockchains” (discussed below), the system comes to gradual consensus with risk controlled by individual business users.

Verification and Consensus

We introduce here the concept of “context based verification” to the blockchain discussion. To illuminate, consider Bitcoin or any other existing blockchain implementation. They primarily use a set proof algorithm (e.g. PoW) to assemble blocks over time and to come to consensus over which blocks and which chain is the common, agreed truth.

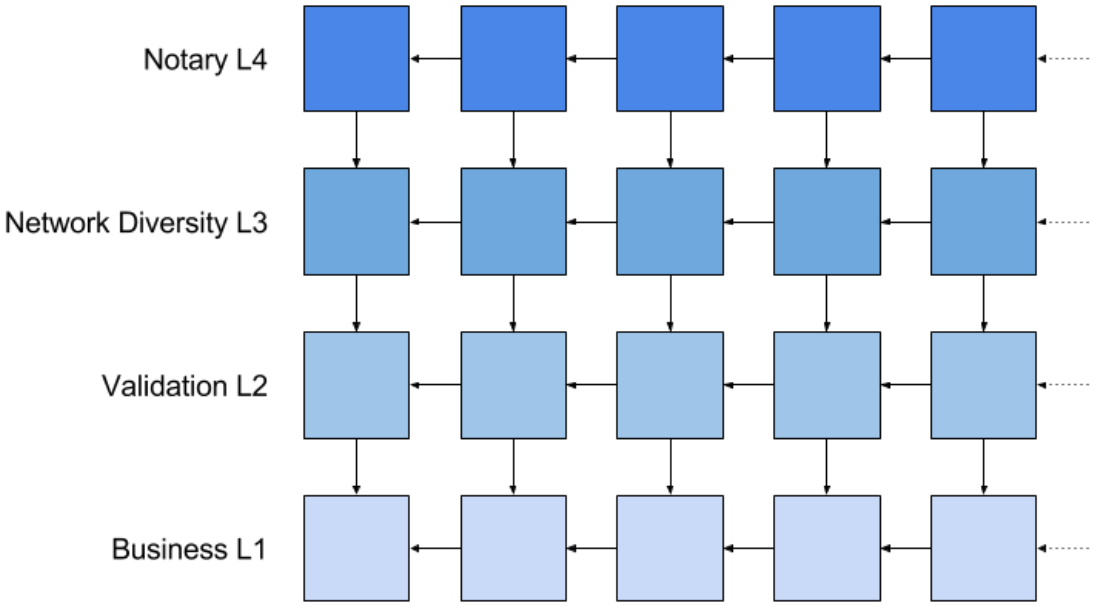


Vanilla Blockchain Structure is One Dimensional

With context based approval, we add another dimension to that design. So the first level is achieved in a purely business context, that is with business logic implemented to provide transaction approval and system logic to arrange those transactions into blocks which are chained together. These blocks will have an abstract proof integrated such as PoW, PoS, or trust. This first level of verification can be considered as analogous to other blockchains.

It is when other verification contexts are added that we see added value to even trust based systems.

Any given node should ideally allow configuration to support or execute one or more of the verification phases described below.



Dragonchain Structure is Linked Between Blocks and Verification Contexts

Level 1 - Business (Approval) Verification

Approval functionality is implemented and configured by the business integrator. This is the placement for integration of “real world” value. Business logic defined by an organization or blockchain platform user is configured to be executed by a blockchain node.
Here also is where the transaction payload is defined by the business to be what is needed for their purposes.
Transactions are arranged and passed to the provided business logic which will determine approval or denial. Approved transactions will be assembled into a “block” generically referred to as a “verification record”.
The payload field of every transaction may be stripped before or after assembling the final block in order to maintain control of the distribution of actual business data. That is, no business payload data will be disbursed as part of the consensus process, and data will remain local on a Level 1 node unless the business owner explicitly pushes the data to another node (e.g. for backup/DR), or explicitly allows an authorized node to pull the data via a subscription feed.

Level 2 - Enterprise (Validation) Verification

This context is defined Enterprise or network wide, and checks for block and individual transaction validity in form, signature, and required data elements.

Verified elements:

1. Block (verification record) construction and signature
2. Individual transaction signatures
3. Individual transaction header elements (that all required header fields are present)

A Level 2 node will assemble a new verification record which will contain:

1. A list of valid transactions and a list of invalid transaction, and in this manner vote on the validity of individual transactions.
2. The hash of the prior Level 2 record created by this node for the same origin (Level 1) node (thus creating a Level 2 blockchain)
3. The hash of the Level 1 block which was validated (thus providing a second dimension to the blockchain)
4. Node owner identity information
5. Node deploy location (data center)
6. Node key management authority information

Level 3 - Network Diversity Verification

Defined enterprise wide, a Level 3 node will verify diversity of validation (Level 2) verifications. That is, a Level 3 node will check the following criteria:

1. Count of Level 2 verification records have been received
2. That those records have come from (configurable count) of unique business units
3. That those records have come from (configurable count) of unique deployment locations
4. That those records have come from (configurable count) of unique key management authorities

This verification context will ensure that validations of transactions are coming from a sufficiently diverse set of distributed sources. It also provides control and measurement of network effect and provides distributed security as an attacker would be required to attack multiple systems, businesses, and data centers in order to tamper with existing data.

A Level 3 node will assemble a new verification record containing:

1. Remnants of criteria met (e.g. Level 2 verification record count, set of business units, set of data centers).
2. The hash of the prior Level 3 record created by this node for the same origin (Level 1) node (thus creating a Level 3 blockchain)
3. The hash of the Level 2 verification records which passed the criteria (thus providing a second dimension to the blockchain)

Level 4 - External Partner (Notary) Verification

Defined network wide (Enterprise+), a level 4 node will provide a notary functionality to the consensus process. Hosted by an external partner, a level 4 node would cryptographically sign any level 3 verification records that it receives. This function allows the Level 4 node to act as an independent witness to level 3 verifications.

Level 5 - Public Checkpoint

A Level 5 node will provide a bridge to one or more public blockchains and allow clients to interact with them (e.g. Bitcoin, Ethereum, Litecoin, etc.).

An important feature that this would provide is that of checkpointing, or placing a hash of an artifact for “proof of existence” on a public blockchain. For checkpointing operations, the Level 5 node will accept a transaction, a block verification of any level, an arbitrary string, or an arbitrary hash. The argument will be hashed and this hash added to a transaction placed on the public blockchain(s). The existence of this hash can be used to prove that the artifact was in existence and at a certain state using public blockchain data. An organization may use this proof to measure and mitigate risk based upon the estimate or calculation of hashpower expended since that time (in the case of a proof of work blockchain). For example, a $2 Million transaction may be passed to a Level 5 node to be placed as soon as possible on the Bitcoin blockchain, and at some point in time later, a party may use that information as a source to measure the amount of hashpower that has been expended since that time, calculate the probability that an attacker could successfully counterfeit that Bitcoin block given a particular percentage of global hashpower, and extrapolate or estimate the cost to expend that hashpower (as well as the sacrifice of hardware and/or currency due to network collapse). If this process results in a risk evaluation that is satisfactory to the business, the transaction can be trusted and accepted.

Another important aspect of the public bridge functionality is the ability to track assets between the private and public side. That is, given an internal currency implemented to use Bitcoin addresses (see currency section elsewhere in this document), a token may be issued on Bitcoin using public APIs or services and this token may live in both the private blockchain and the Bitcoin public blockchain. Owners of the keys or wallet would be able to transfer the token or asset with either public or private blockchain interactions. The Level 5 node may be used to track this asset between the blockchains as well as keep them in sync with each other.

Level X - Proprietary context verification

It should be possible for a business, Enterprise, or the entire network to define a custom verification context and have it executed to meet business needs.

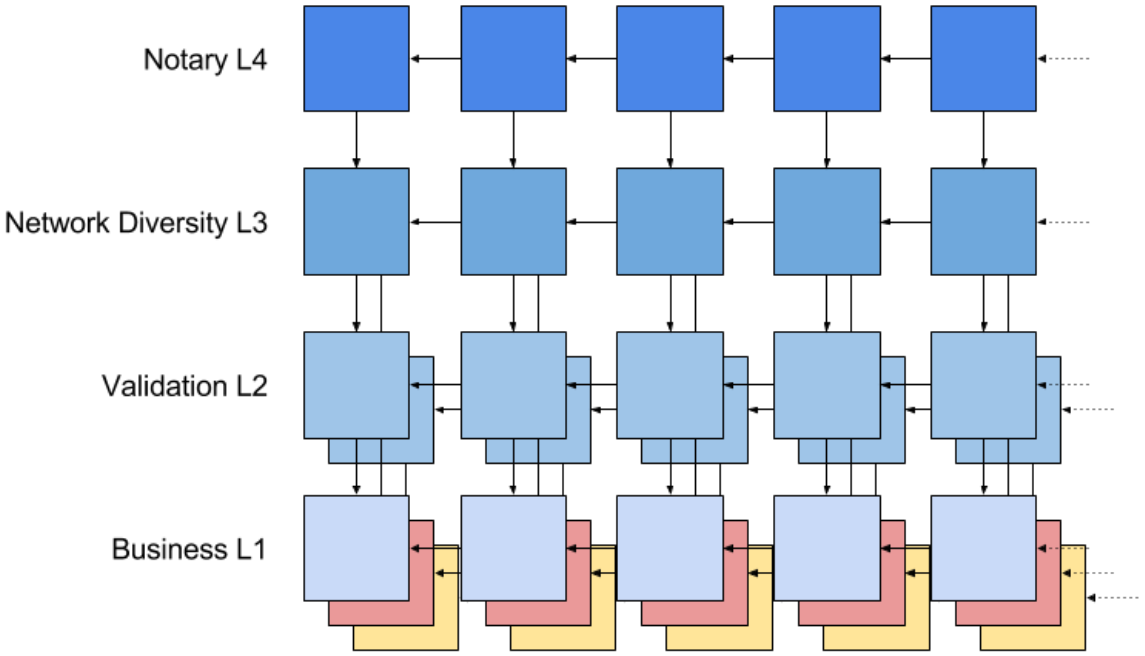
A node may be configured to run an arbitrary verification context which would be triggered in the following manners:

1. By the receipt of a broadcast from another node (in sequential or non-sequential phase)
2. By a timed or periodic trigger (e.g. cron)
3. By notification via observer pattern

Blockchain of Blockchains Concept

This architecture may be best understood as a “blockchain of blockchains”. That is, a business

approval function node (see Level 1 below) functions much as a standard blockchain on a 1 dimensional level. However, each business concern will generally have its own node to do this work, each with its own blockchain. It is where these blockchains become combined that consensus is reached.



Blockchain of Blockchains

Currency

This architecture should be multi-currency capable. That is, generally that if a currency use case is defined, that a node(s) may define a currency and support its use. More than one currency may be in use concurrently on the network as a whole.

That said, this architecture should not define a “base” currency, or one that the system itself runs upon. If such a use case arises (as indeed it is very likely to see value in the availability of a currency whereby nodes may pay each other for verifications), it is the philosophy of this architecture that a node should be configured to create and maintain that currency. This will allow a more flexible development of marketplaces than any attempt to define that early in the development of the platform.

The implementation of this architecture should likely provide one or more templated or configurable currencies for ease of deployment. Such implementations may define such things as mining or minting algorithms, addressing, wallet management, and etc. They should also be extensible by users as possible for further experimentation and customization.

Currency Modeling

The architecture allows a user to model currency and monetize late in the game. It is possible for a user to place information from many sources atop the blockchain, watch its use over time, and determine its value based upon business and customer priorities. At this point, assets and activities can be monetized and a mining or minting algorithm may be developed which will incentivize a business' employees, teams, or customers. This process of quantify, monetize, and incentivize may be one of endless tuning, but would provide a transparent economic system.

There is potential for this framework to provide agility in organizations where data providers are enabled to provide immediate and early access to research data, reports, and other information to projects that would not otherwise ever see such data, as it would typically require top-down organizational approval at great cost and risk. The use of such data could be tracked and its value determined, leading to direct monetization and a bottom-up funding mechanism.

Bitcoin Addressing

The base implementation should likely default (for the current time) to utilize Bitcoin addressing and cryptography in order to leverage the ever growing external Bitcoin ecosystem. For example, use of Bitcoin cryptography in a private currency will enable the transparent use of hardware signing wallets for internal use (e.g. KeepKey, Trezor, Ledger). Another example is that of tokenization, whereby one may directly integrate Bitcoin tokenization provider technology for use with an internal blockchain (e.g. Counterparty or Tokenly).

Interoperability

With the transaction class header field, it is possible to wrap foreign cryptocurrency transactions within a private blockchain transaction.

Smart Contracts

Many questions arise in regards to blockchain based "smart contracts"...

Turing Completeness

Top of this list is Turing completeness. Bitcoin is purposely *not* Turing complete. It is as complex as necessary to provide the capabilities that are needed for its application.

Some blockchain implementations such as Ethereum are Turing complete. This allows some very interesting applications, yet brings some risk and difficulty. The script must be executed within a special and verified container or virtual machine to insure deterministic results no matter the hardware or operating system upon which the node is running. Various aspects of the script must be monitored (e.g. loops) for failure or otherwise unexpected or unauthorized action.

Transactions, Atomicity, and Rollback

A smart contract system may need to provide some level of controlled or automated transaction and rollback capability to ensure that any given action will only take effect if the entire transaction operation is successfully completed.

Distributed Execution

Where is the smart contract executed? Is it on the distributed (calling) node?

Dragonchain Smart Contracts

The architecture calls for definition of “approval context” code to be configured or deployed on Level 1 business nodes. This approval code can be considered a smart contract. By default, this smart contract will only be executed on that Level 1 node(s), and under the direct control of the business owner. This provides a familiar client/server interface to the creation of a smart contract, and simplifies the risk evaluation. The attack vectors are more common and known to modern engineers. Turing completeness is provided just as within any web service platform.

A smart contract may well be distributed, and may be executed on a pay per play basis. In this case, many of the risks with smart contracts described above apply, yet it can be assumed that the parties will have a trust relationship inside of an organization to draw on and the provisioning of the code may include necessary legal agreements.

In much the same way, a group may host smart contracts as a IT support business, and provide varying plans for chargeback/payment.

A smart contract may be employed on a node in the following manners:

- Hardcoded in a forked codebase
- Configured (with smart contract code deployed on system)
- Delivered via blockchain (admin would send multi-signed transaction with smart contract, start date, etc.)

Subscription Data Feed

As transaction payloads are stripped prior to any broadcast within the consensus process, nodes will share business data as necessary via subscription data feeds. This amounts to a push or pull mechanism where some subset of a nodes transactions are continuously fed to another node.

In the case where a node needs another node’s data in order to “mash up” with its own data to serve a customer, that node would configure to request access from the “origin node” which owns the data. This request may come with authentication or authorization information and the

origin node may approve or deny the request. The requesting node may also provide criteria for the data feed such as only a certain transaction type, only transactions that have surpassed a certain level of verification on the network, or only involving a certain identity. In this way, the requesting node will have a local cache of only the data that it needs, and it will be able to answer its customers with confidence that the data is unaltered and verified without the need to reach out to the origin business services in the future.

In the case where an origin node would like to distribute its data (i.e. for backup or disaster recovery), the node may pay for such a service and continuously push the data to that node. The backup node will be able to verify that the data is error free upon receipt and periodic audit.

Network Management

For a blockchain network system, many typical elements will necessarily exist such as provisioning, discovery, maintenance of available and quality nodes, etc. The architecture will however take some philosophical positions on the configuration and maintenance of the network.

All considerations for network and communications management should in this architecture be made from the individual node context. Such decisions as which nodes to connect with should be decentralized decisions made by each node independently. There may well be mechanisms for central management of hints or starting lists of guaranteed available nodes within an Enterprise, but no attempt should be made to centralize the connection requirements of the network.

Data distribution

By default, no transaction payload (business) data should leave an origin (owning) node nor be propagated across the network. As part of the consensus and network communications, all transaction payloads are to be stripped prior to broadcasting.

Only when explicitly authorized by the origin node are the payloads and full transactions caused to be broadcast to authorized nodes.

Node Discovery

Node discovery should take place via peer to peer request.

Network Marketplace

Any notion of marketplace or currency with which nodes may trade or hire out verifications should not be implemented in the system infrastructure, but rather as a currency implementation (add-on) on a node within the blockchain network. In this manner, the architecture will remain flexible and open to new ideas or unforeseen requirements.

Node Quality Assessment

A node should track and ascertain the quality of connected and disconnected individual peers. Considering currently connected and disconnected peers separately, a node should track attributes such as:

- Average latency of the connection (periodically assessed)
- Signing success rate
- Rebroadcast counts
- Average verification time
- Deploy location (for diversity criteria)
- Owner (for diversity criteria)
- Unsuccessful connection attempts
- Last connection attempt

A node should conduct periodic assessment of connected nodes more often than disconnected nodes, however over a reasonable span, the node should assess quality of disconnected nodes in order to have necessary intelligence to prioritize peer connections should a large portion of connected nodes become unavailable.

Verification Receipt

A broadcast of a verification record to a higher level verification node should provide a receipt message notifying success or failure, and if successful, should include the verification record as signed by the higher level node. This same mechanism should be passed further down the line to reach the origin node. Although it is a design consideration, something in the manner of an asynchronous reply to another broadcast call may be an appropriate mechanism to provide such capability.

Implementation Options

- Custom (e.g. Apache Thrift)
- RESTful service calls
- Distributed database framework with replication

Interoperability and Proposed Standards

There are many avenues to consider for the question of interoperability with other blockchain systems.

Checkpointing vs Wrapping

One may choose to connect transactions or other artifacts via checkpointing (see Level 5 - Public Checkpoint above) or by wrapping a foreign transaction or artifact within a Dragonchain

transaction (see foreign currency transaction header above).

Subconsensus

A business may choose to employ another blockchain at any level of the verification process. For example, to provide a decentralized Level 1 approval implementation, one may choose to employ Bitcoin or other proof of work based blockchain to come to consensus on a currency transactions.

