# Rethinking The Blockchain Account Concept

Bitcoin introduced the decentralized, consensus-based ledger concept, and how to use it to represent a cryptocurrency. Bitcoin used a simple scripting language to enable the management of the ledger concept, enabling the exchange of cryptocurrency between entries on the ledger (i.e. accounts).

Ethereum added to the decentralized, consensus-based ledger concept, by generalizing the simple scripting language to a turing complete language, while using the concept of "gas" to solve the non-termination problem inherent in turing complete languages.

I think everyone would agree that these two concepts have had an amazing impact worldwide. But as great as these concepts are, possibly a watershed moment in history, we think there is yet another evolutionary stage in blockchain technology which will further increase its adoption.

To explain our new technology concept, lets start with a simplifying, somewhat inexact, analogy between blockchain technology and a more well understood technology, spreadsheets (e.g. MS Excel, Google Sheets).

## Spreadsheet analogy

If we view Bitcoin as a single (virtual) spreadsheet, we can consider each account as a row in the spreadsheet. Transfers of bitcoin between accounts are transfers between the rows. The pay-to-script hash ("P2SH"), can be thought of having a script associated with an individual cell on a row of an individual account. Since this is a virtual spreadsheet, one useful feature is that the number of rows is essentially infinite. This means that users can create as many accounts as they would like, which conceptually are as many rows as they would like in the virtual spreadsheet. I would note that other than the P2SH, which is associated with an individual transfer of bitcoin, the concept of an account storing state is not part of Bitcoin.

Ethereum introduced the concept of the smart contract. If Bitcoin is a single virtual spreadsheet, we can consider the Ethereum smart contract as introducing multiple virtual spreadsheets. Each smart contract is its own virtual spreadsheet. A token smart contract (i.e. ERC 20), has an array that contains accounts and the amount of token that each account owns. Thus, each token smart contract can be thought of as having a row for each account and an associated count of tokens. Since a token smart contract is represented by an individual account on the blockchain, and again, accounts are essentially unlimited, there can be an unlimited number of individual smart contracts accounts, representing unique types of tokens, on an Ethereum blockchain. It should be understood that each smart contract is essentially stand-alone, isolated. That is, although one smart contract may call another smart contract explicitly, in particular as a library call, there is no generalized means of sharing code. To create a new token or other data representation on Ethereum, a new smart contract must be registered.

In summary. from an account viewpoint, Bitcoin introduced the account as an entry on a single global virtual spreadsheet. Ethereum introduced the smart contract account, which enables multiple virtual spreadsheets, while regular accounts continue to be more like Bitcoin accounts.

What if we revisited the concept of account and made all accounts smart?

In Ethereum, a smart contract has state data associated with it. What if every account has state associated with it?

What if an account could create and contain arbitrary objects in its state?

What if, using an object-oriented paradigm, the code for such objects came from classes, and those classes were registered once on the blockchain and reused by accounts?

What if the object-oriented concept of inheritance was supported such that objects were instances of classes, which in turn could inherit most of their code from ancestor classes.

**Ramifications**

What are the ramifications of accounts owning objects instead of just a count of native token?

What are the ramifications of creating (instantiating) new objects without needing to register new code?

What are the ramifications of new classes once registered on the blockchain becoming an immediately useable, essentially integral part of the blockchain for all users?

To consider these, we need to propose new ways we would like to use a blockchain:

As a user of a blockchain, I would like to be able to create new assets in my account on the blockchain and be able to sell them. Similarly, I would like to be able to buy other assets and resell them. I would like to be able to do such transactions both on marketplaces and exchanges as well as person-to-person (i.e. wallet-to-wallet).

Since I can't know in advance the type of asset I might buy or sell, I want all assets to have some commonality between them, but support unique features for each type of asset. I want my account to support all assets, regardless of type, and I want my wallet to be able to support all assets, regardless of type. As a user and asset owner I don't want to know about or touch and software code. I do want assurances that the code that manages my assets is valid, but I no more want to look or touch that code than I do the code that runs my cellphone or laptop. As it turns out, in computer science we solved this problem with the object-oriented concept and class inheritance.

If accounts can have arbitrary asset objects associated with them, and the objects are instances of classes that contain the code to manage the assets, then we can deliver to the user these capabilities. This is a direct result of the ramifications of accounts owning objects, objects as instances of classes, and classes as the main concept for code on the blockchain.

**The DataGrid Blockchain Account Model**

The DGB Account Model generalizes the account concept first introduced in Bitcoin, and extended to include Smart Contract Accounts with Ethereum to all account entries including state data, not just smart contracts. On the DGB, every account entry now includes a state hash instead of just the smart contract accounts. In essence this makes every account "smart". Each account can now include objects that can represent assets.  Further, through object-oriented inheritance all asset objects can inherit common behavior from a parent class, (e.g. Class Asset).  Complex relationships can be easily represented by the state information of each object instance which is captured in the state information of each account.  Since accounts are essentially unlimited, there can be an unlimited number of accounts contains both object instances and code for inheritable classes stored on the blockchain.

Returning to my desire as a user to never need to touch or know explicitly about code, through the magic of inheritance, I can simply create new asset objects of a known type (i.e. class) by sending transactions to an account I own, referencing the type of the object I want created. Further, I can buy and sell such asset objects, by sending transactions referencing these objects. I don't need to know about Bitcoin's Forth scripting language, or Ethereum's Solidity smart contract language as a user.

The Prasaga DataGrid Blockchain creates this new approach to blockchains, tokens and assets ownership through the introduction of the following complimentary technologies:

- The DataGrid Blockchain Extensible Blockchain Object Model (XBOM)
- The Extensible Smart Object Asset ("XSOA") Classes
- The DataGrid Blockchain Scalable Sharding Consensus Protocol

To learn more about Prasaga's patent pending technology contact us through our website at prasaga.com