Smart Object Assets, Smart Object Asset Ownership and Fractional Smart Object Asset Ownership

with the

DataGrid Blockchain Extensible Blockchain Object Model

David Beberman, CTO

Prasaga, LLC

June 4, 2019

Smart Object Assets, Smart Object Asset Ownership and Fractional Smart Object Asset Ownership

With the DataGrid Blockchain Extensible Blockchain Object Model

Overview

The DataGrid Blockchain ("DGB") introduces a new concept to blockchain solutions: The Smart Object Asset ("SOA").

Using the DGB Extensible Blockchain Object Model ("XBOM")[1], virtually any reference to an asset whether a virtual or physical asset, may be owned by, and stored in a DGB account. Further it may be transferred (i.e. sold) to any other DGB account. As is described below this innovation enables a wide variety of asset ownership concepts including, but not limited to:

- Asset ownership and transfer
- Fractional asset ownership
- Asset ownership cashflows
- Voting rights
- Use and access rights

Examples of Assets

An asset may be a single physical item such as a boat, car, house. It may also be an entity such as a business, or financial instrument.

A Smart Object Asset instance represents the asset on the DGB in a one-to-one correspondence.

Asset Creation

An account holder creates a new asset by creating an object instance of ClassAsset or a subclass. The new smart object asset is created with the description information of the asset providing proof of ownership of the asset. The description information and proof of ownership of the asset that the asset object represents are specific to the asset, and as such are determined by subclasses of the Class Asset.

The newly created asset object instance is stored in the asset list in the creating account.

[1] The Pat. Pend. 62843392- *Extensible Blockchain Object Model (XBOM) Foundation Classes and Objects and Supporting Data Structures* is an exciting new technology from Prasaga that defines and implements the smart class and smart object model in place of the smart contract model.

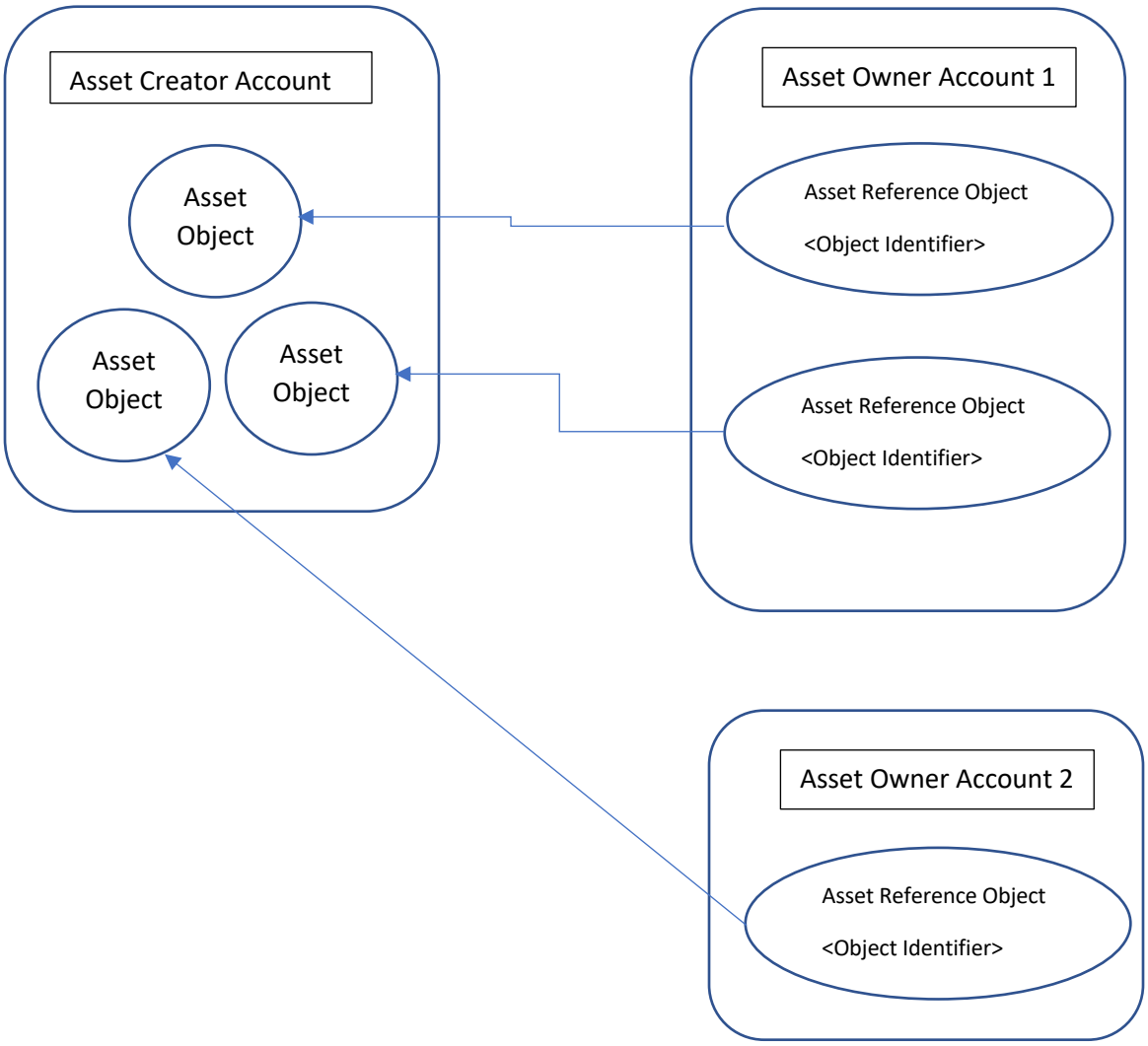
Asset Object Reference

Once an asset object has been created, if the creating account wants to enable transfer of ownership of the asset, the account holder creates a reference object to the asset object by creating an instance of ClassReferenceAsset. A ClassReferenceAsset object instance contains the object identifier of the asset object instance. These object instances act as proxies to the asset objects, and are used to show ownership of the asset object.

Specifically, owning an asset object means by definition an account that contains an instance of a ClassReferenceAsset object that contains a reference to the asset object.

Implementation note: ClassReferenceAsset objects are passed between accounts for transfer of ownership. The underlying asset object is permanently stored in the creating account's state space, even if the creating account no longer owns the asset. As a result, the object identifier stored in the ClassReferenceAsset object is itself immutable.

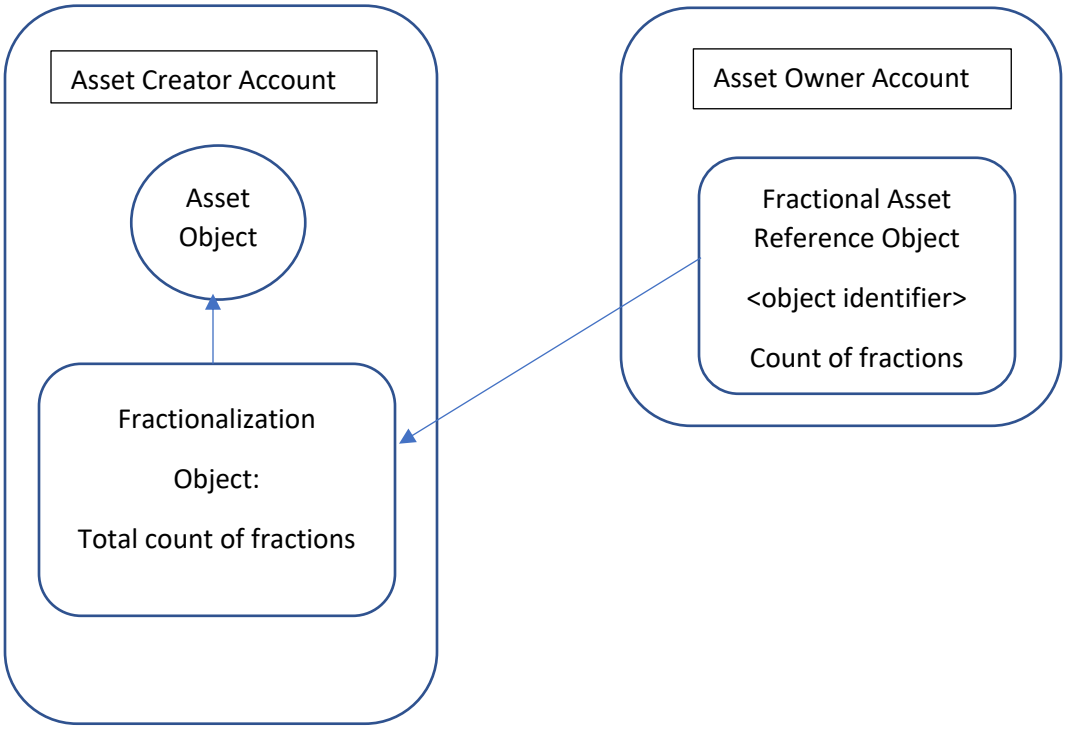
The following depicts the fundamental asset ownership concepts:



Each rounded rectangle represents an account on the DGB. The Asset Creator Account contains the asset objects that it instantiated. As is shown an account may create any number of asset objects. Two accounts are shown that own some of the assets created by the Asset Creator Account. Each owning account contains an Asset Reference Object instance with the object identifier of the owned asset.

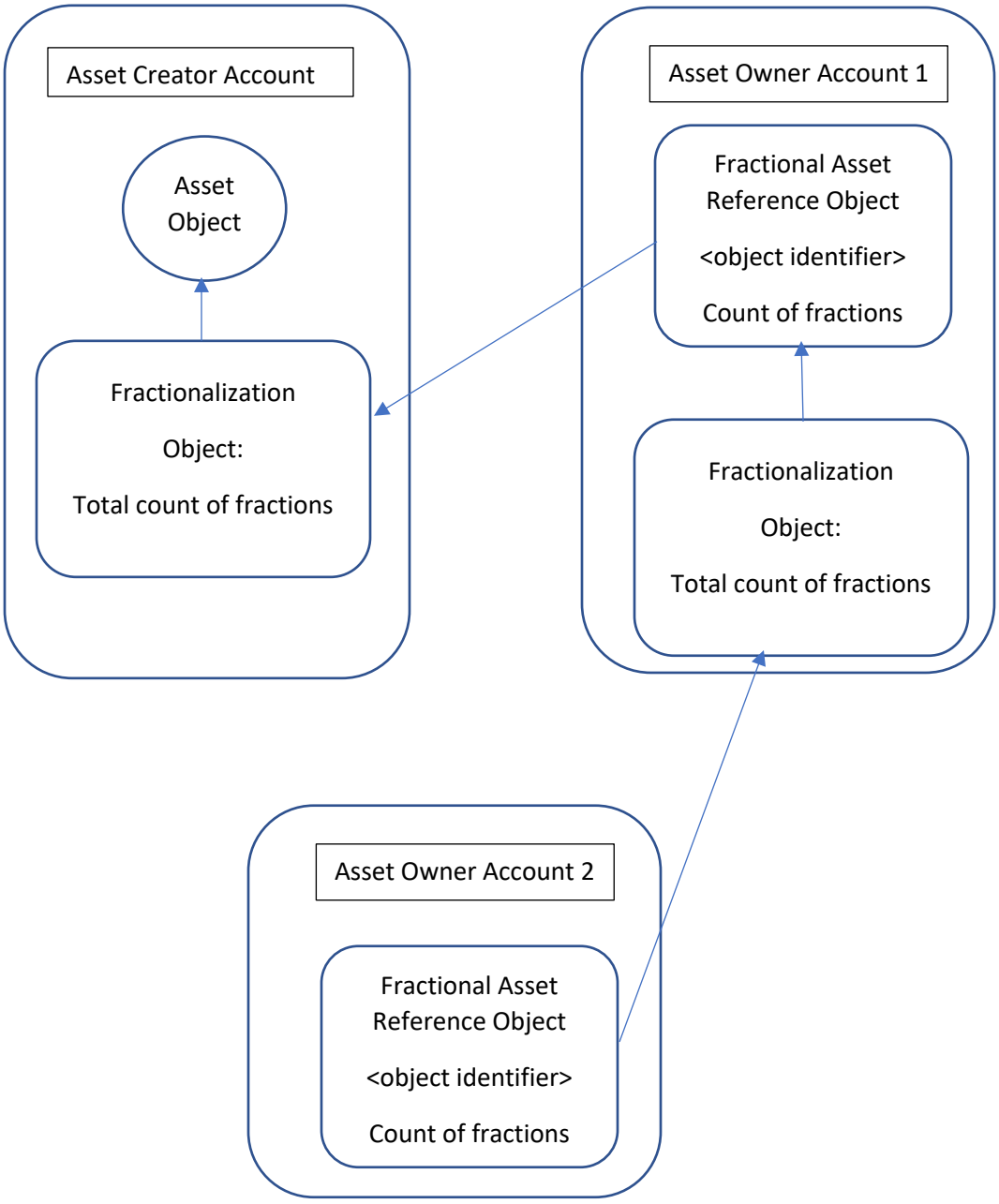
Fractionalization of Assets

The term fractional asset ownership refers to the general concept of owning a fraction of an asset. The most common use of fractional asset ownership is shares in a corporation. Each share represents a fraction of ownership in the corporation. Other types of assets may be fractionally owned such as a yacht, corporate jet, vacation timeshare, etc. The DGB classes support fractional asset ownership similarly to general asset ownership. The difference is that a fractionalization reference object is used to create the fractional ownership. This is depicted below:



A fractional asset reference object can be traded in a transaction just like an asset reference object and behaves functionally identically. For example, cashflows for a fractionalize asset would flow to the owners based on the fractions they own.

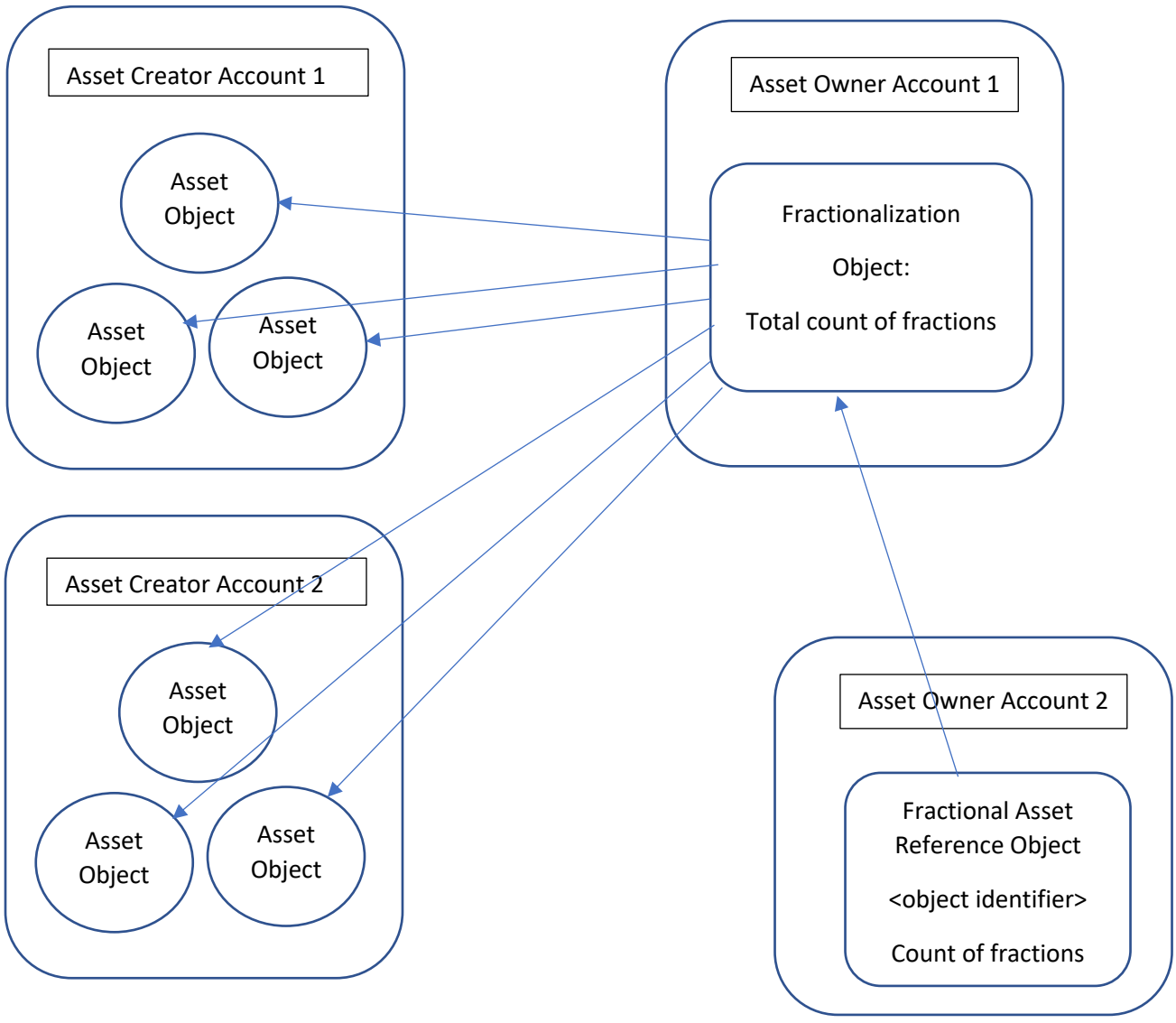
Multitiered Fractionalization of Assets



In this model, a fractional asset reference object is used as the asset object by a second fractionalization object. This new fractionalized asset can now be referenced by a new fractional asset reference object which can be transferred to new account. All ownership rights and cashflows would flow through the references.

Creation of Baskets of Assets

Assets may be grouped together into a “basket” and fractionalized. A fractionalization object may contain multiple asset object references from multiple accounts. This is depicted below with 2 create accounts.



As with the multitiered fractionalization of assets above, a basket of assets could consist of fractional assets as well as asset objects. All of the flow-through would behave identically.

Example Use Case

Business Shares As Assets

Given the above fractional asset model, a business might do the following:

- create an account on the DGB
- create an instance of asset object to represent the business
- create an instance of a fractionalization object, initialized with N fractions

The business could then sell the fractions of ownership to any other account, with the new account containing a fraction asset reference object. The new account owner may sell the reference object to other accounts including back to the original business account.

If/when the business declares a cashflow (i.e. dividend), the account holder of the fraction asset reference object sends a transaction to the fraction asset reference object to collect its cashflow from the fractionalization object, which in turn collects the cashflow from the underlying business asset object.

Brief Comparison of Smart Object Assets to Smart Contracts

A smart object asset, and the associated asset object references as described above differ from the smart contract model significantly and enable new ways of working with assets on blockchains.

Smart contracts are implemented as a single account, with associated code (e.g. Solidity code), and a single data space. If a Smart contract implements a token, such as an “altcoin” or a “security token”, any account that owns a token is recorded as an account address in an array list in the smart contract account. Each new token is a new smart contract, with separate code and a separate single account data space.

The smart object asset model stores the asset reference objects in each individual owner accounts data space. This in sharp contract to the smart contract approach where the account address (which functions as a reference) is stored in the single smart contract. As described above, the smart object asset concept implicitly supports multiple relationships without adding or changing any code. All that is required is creation of the relevant objects as standard transactions. This flexibility enables near limitless relationship structures between asset owning accounts without introducing the opportunity for programming mistakes.

Brief Comment on Sharding and Scalability with Smart Object Assets

The account ownership of smart object assets model has a fortuitous side effect with respect to scalability. The transfer of ownership of a smart object reference asset between two or more accounts is a local matter between the accounts. That is, if multiple smart object reference assets are being transferred between unrelated disjoint accounts, all of the transfers may happen simultaneously on separate blockchain shards. For example, if account A is transferring to account B, and account C is transferring to account D, both transfers may happen simultaneously on separate shards.

In comparison, a smart contract account must process transactions serially, which implies the state transition must take place on a main blockchain or a single shard. Therefore, in the above example only one of the two transactions can take place at a given time. Sharding does not provide any scalability in the smart contract situation.

Conclusion

The Smart Object Asset model introduces the concept of ownership of arbitrary object to the blockchain and cryptocurrency model. Since a SOA can represent virtually anything and be traded with any account, many blockchain applications that are currently implemented as separate smart contracts and individual tokens may be far more naturally be implemented as SOAs. Using classes defined with the XBOM, transactions for all SOA's may be handled uniformly. This extends naturally to wallets containing smart object assets as well as buying and selling of assets. The above diagrams and descriptions are not meant to be exhaustive. The SOA model can be readily extended to support arbitrarily complex relationships between accounts and types of assets.

References

"Fractional Ownership", https://en.wikipedia.org/wiki/Fractional_ownership

"Fractional Ownership", https://www.investopedia.com/terms/f/fractionalownership.asp