Pegging $ATOMs to ERTP

Background Content:

https://github.com/Agoric/dapp-pegasus
MarkM's presentation in Berlin June 2019 on ERights Pegging on IBC
Simple Exchange Dapp
Network API
Agoric Peg and Simple Exchange
https://github.com/Agoric/dapp-pegasus/blob/master/index.d.ts

Purpose

To enable an individual to transfer the value of their Atoms, currently on a Cosmos Zone, into Agoric and use them in any Agoric contract (running on an Agoric Zone) with value constant to its value in Cosmos. And then whoever is holding the Atoms representation on Agoric can transfer them back to Cosmos and redeem them.

(i.e. be able to offer to exchange Atom vouchers for Quatloos or similar ASTs on Agoric)

User Functionality

(from the user's viewpoint, what operations can they do and how should our UI enable/reflect them?)
Transfer Atoms from Gaia (Cosmos zone) to an Agoric wallet.
Transfer Atoms back from Agoric wallet to Cosmos, possibly to a different Gaia address.

Be able to use their Atoms in Agoric contracts.

Do the SimpleExchange Dapp with moola backed by Atoms.

Method Functionality

(from the internal perspective, what high level operations will we need to implement? As a first approximation, what API commands would we want to make available, such as "transferFromCosmos(amount)", that'd let us implement the user functionality?)

transferFromCosmos(amount) //"amount" might be a Payment containing an Amount

transferToCosmos(amount) //”amount” might be a Payment containing an Amount

Contract Functionality

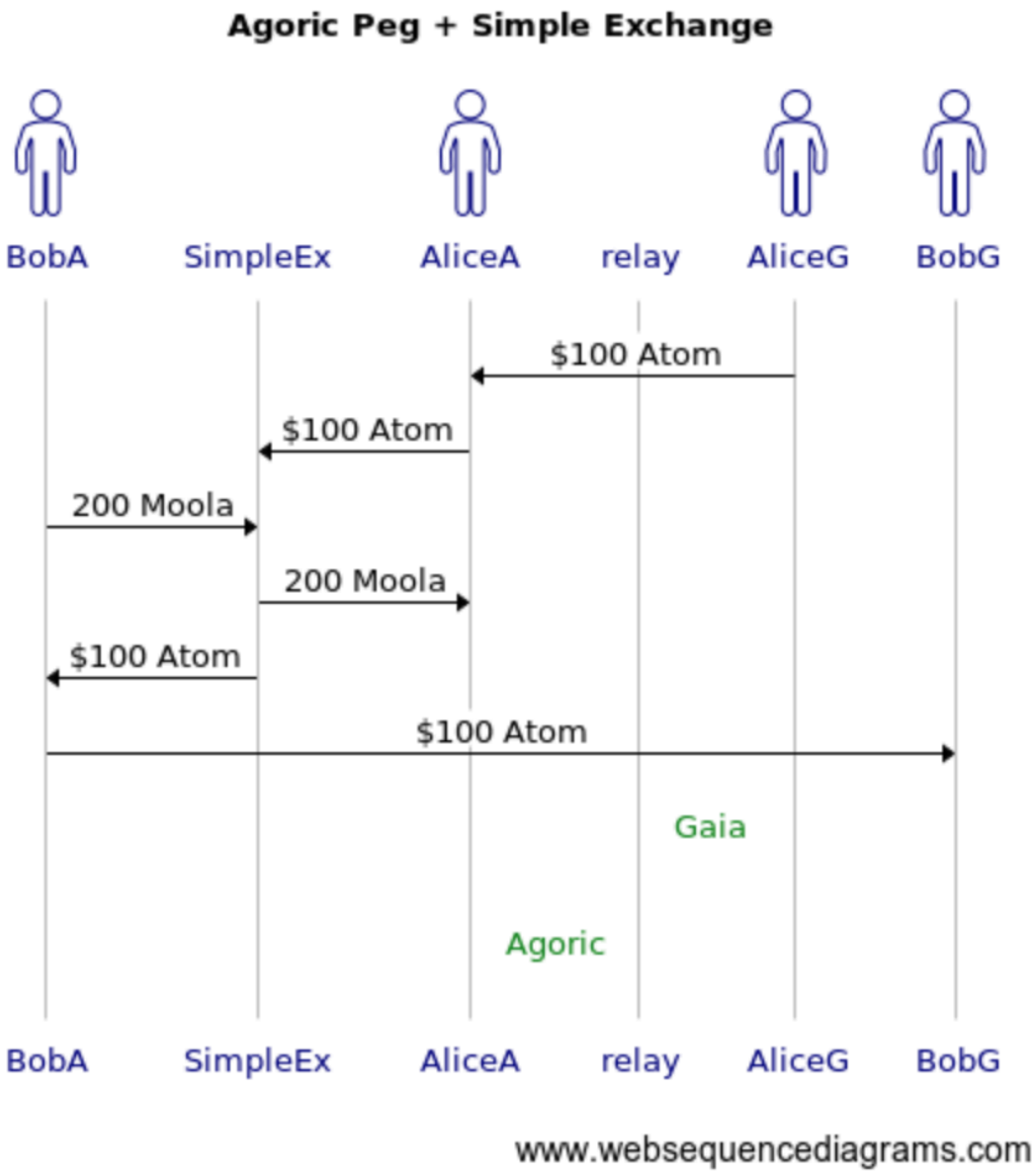
(start breaking it down to what we want to have happen from a contract/ERTP perspective to enable the above). Modify SimpleExchange contract/Dapp to use Atom-backed moolas.

Needed Data Structures

(i.e. place to store user balances or other pertinent data)

Wallet
Payment

Sequence Diagram



Current Next Steps/Todos

1. Get [SimpleExchange dapp]up and running
    a. ~~agoric init --dapp-template=dapp-simple-exchange demo2~~

    b. git clone git://github.com/Agoric/dapp-simple-exchange DONE
2. Zoe peg contract, and binding the $ATOM vouchers to a specific IBC connection DONE
3. `rly tx xfer` to transfer from Gaia to particular Agoric wallet (possible generalization PR)
4. Use the $ATOMs in trades on SimpleExchange.
5. Send the $ATOM from Agoric to a specific Gaia account address

UI improvements:

1. Pop up the wallet
2. Remove orders that are cancelled
3. Click on a row to populate the fulfilment order

Presentation

- We're going to demo pegging. Pegging is..
- Very brief: Prerequisites/Background: What is pegging? What is transfer?
- Demo: Is flow from sequence diagram.
- What did we do?
    - "Pegging is the name, transfer is what we do"
- How did we do it?
    - Include usage note about not wanting to keep pegged assets around for long as risks increase the longer an asset is pegged.
- Why did we do it?
- What did each of us learn?
-

https://peg-as.us (go here for instructions for how to do it and at least start the process)
Dapp with links to Agoric and Gaia wallets if lucky.

UI will have selectable currency and chain. Will then contact wallet and initiate, then user fill out destination and depositAddress.

Send Gain chain's $ATOM to Agoric as pegged erights

Use erights in Simple Exchange

Send Pegged $ATOM back to Gaia

1. Ask Pegasus to create transfer invite bound to a specific address,
2. Exercise the invite with erights to transfer
3. When approve a Zoe offer for that invite, saying taking these e-rights and give them to Zoe.
4. The contract picks it up, sends it, and gives a “Transfer Complete” or “Transfer Failed, Refund Made” or “Oh crap, the transfer protocol screwed up and yours and everyone else’s pegged tokens lost all their value”

Documentation

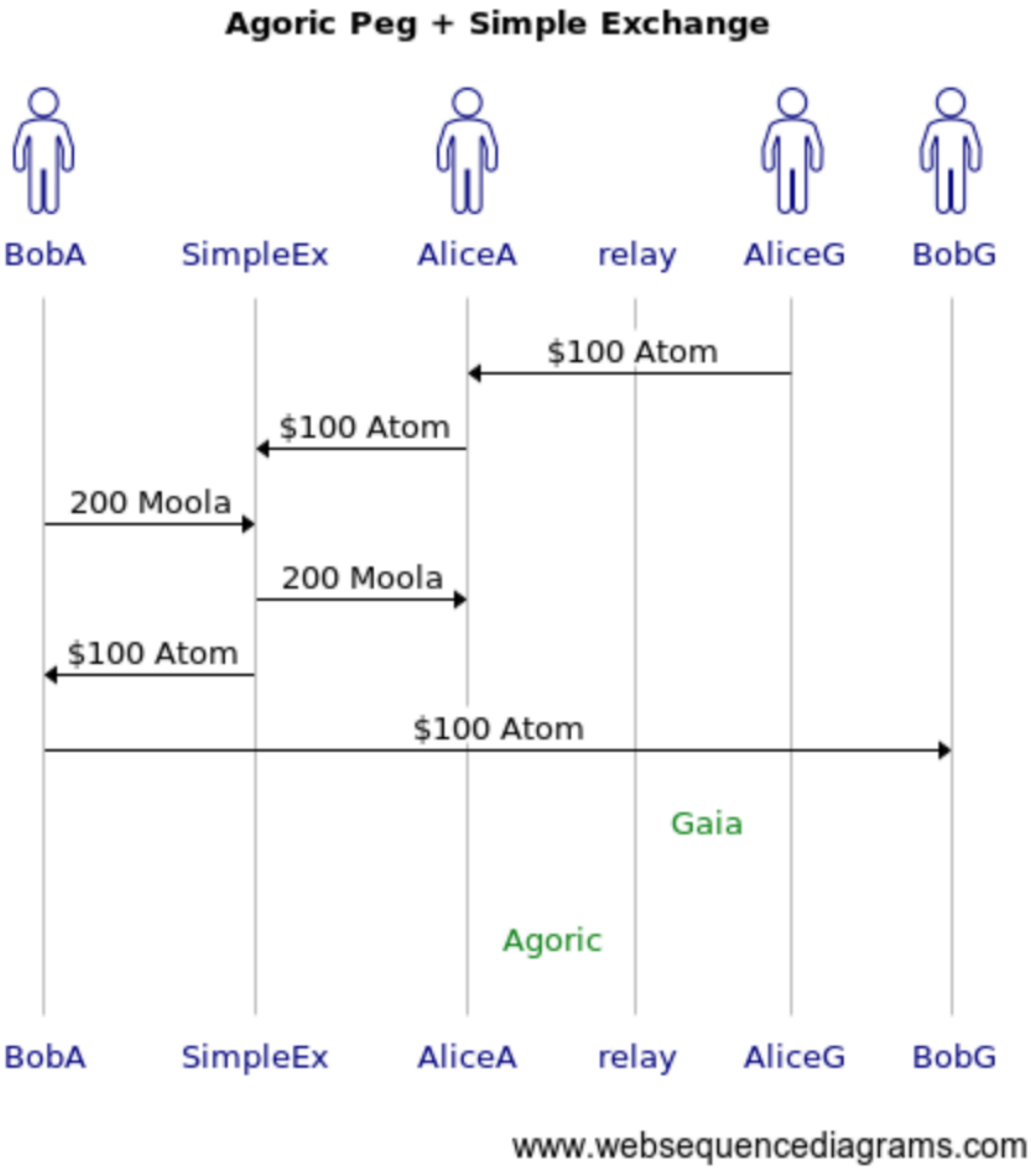
Pegasus: Pegging $ATOMs to the Agoric Environment

Introduction and purpose

This document covers **Pegasus**, a system to peg $ATOMs from Cosmos/Gaia to the Agoric system/environment. (And yes, MarkM now owns the domains `peg-as-us.com` and `peg-as.us`) Users can transfer the rights to their $ATOMs from Gaia to Agoric wallets and then use them like any other asset in the Agoric environment. $ATOMs can be swapped for other digital assets, gifted to other users, used to make bids in auctions, etc.

However, Pegasus is really more a proof of concept. What we hope we’ve provided is a framework and platform for pegging any external-to-Agoric digital asset over a network connection by any of multiple transport protocols. While Pegasus uses IBC (Inter-Blockchain Communication) as its protocol, it should be easy to modify it to use either another single protocol or offer users a choice of protocols.

Actions sequence



The above diagram shows a typical sequence of Pegasus actions. There are two actors, Alice and Bob. Each has a presence in both the Agoric and Gaia zones. Their Agoric presences are both denoted with an “A” suffix; AliceA and BobA. Their Gaia presences are denoted by a “G” suffix; AliceG and BobG. Their actions are:

1. AliceG transfers the rights to $100 Atom to herself on Agoric (AliceA) via an Gaia to Agoric relay. AliceG can no longer use those $100 Atom in the Gaia zone.
2. AliceA uses the Simple Exchange Dapp to exchange her $100 Atom with BobA’s 200 moola.

3. Reallocation occurs, and AliceA now has 200 moola and BobA has $100 Atom.
4. BobA transfers $100 Atom to himself on Gaia (BobG) via an Agoric to Gaia relay. BobA can no longer use the $100 Atom in the Agoric zone, BobG can now use it in the Gaia zone, and neither AliceA nor AliceG has any access to the $100 Atom.

UI/Code example
****tyg todo: Needs filling in****
```js
//Show code that does a basic $Atoms transfer to Agoric and back
//Any prep on Gaia side.
//Transfer $Atoms to Agoric
//Transfer $Atoms to someone else (“Bob”) via Agoric Dapp
//Transfer $Atoms from Bob to himself on Gaia.
```

Concepts and Types

Pegasus defines or significantly uses these non-Agoric types:

- `Connection` `{ Object }`
    - Represents the connection between two zones, in this case Agoric and Gaia.
    - Specifically, the same `Connection` object as in the Network API.
- `Endpoint` `{ String }`
    - A unique identifier for a connection, as defined by the Network API. For example, `/ibc-channel/chanabc/ibc-port/portdef`
- `DenomUri` `{ String }`
    - Name of a denomination scoped to a connection and the transfer protocol in use. For example, `'ics20-1:portdef/chanabc/uatom`
    - Names start with the protocol in use (`ics20-1:`). These are only used when calling `getDenomUri()`, so you don’t need to worry about how to construct one.
- ``Denom`: `{ String }`
    - The denomination ($Atom, moola, etc.) being transfered. Specifically for systems that treat currencies according to string names, it’s the token denomination as a string e.g. ‘uatom’ (see Cosmos). Used due to the mismatch of the name on one side of the connection with the name used on the other side.
- `DepositAddress` `{ String }`
    - Where the asset is being sent to.
- `TransferProtocol` `{ String }`
    - What protocol (IBC, https, etc.) is used to transfer the assets.
- `Peg` `{ Object }` (below)
- `Courier` `{ Record }`
    - Consists of a `transfer` field which is a record of a `payment` key with a `Payment` value and a `depositAddress` key with a `DepositAddress` value, and a `Promise`.

The Pegasus API

A `Peg` object is associated with a given asset and connection has three methods:

- `getAllegedName(): string`: Get the debug name for this peg.
- `getLocalBrand(): Brand`: Get the issuer brand the local pegged ERTP asset.
- `getDenomUri(): string`: Get the remote pegged denomination identifier.

The Pegasus Zoe Contract PublicAPI methods are:

- `makeInviteToTransfer(peg, depositAddress)`
    - Create an invite for a single transfer over the Peg.
    - `peg`: `{Peg}`
        - The peg over which to transfer
    - `depositAddress`: `{DepositAddress}`
        - The remote address to transfer to.
    - Returns: `{Promise<Invite>}`
- `pegRemote(allegedName, c, denom, amountMathKind?, protocol?)`
    - Peg a remote “root” to a local “shadow” over a network connection
    - `c`: `{Promise<Connection> | Connection}`
        - The network connection (IBC channel) to communicate over
    - `denom`: `{ String }`
        - The denomination (“uatom”, etc.) being transfered.
    - `amountMathKind?`
        - One of `nat`, `str`, or `strSet` specifying the kind of extent values this brand’s associated `amountMath` methods work on. Defaults to `nat`.
    - `transferProtocol`
        - What data protocol (IBC transfer, etc.) is used to transfer assets. Defaults to `ics20-1` (IBC Fungible Token Transfer).
    - Returns: `Promise<Peg>`
- `pegLocal(allegedName, c, issuer, protocol?)`
    - Peg a local “root” to a remote “shadow” over a network connection
    - `c`: ` { Promise<Connection> | Connection}`
        - The network connection (IBC channel) to communicate over
    - `issuer`: `{ Issuer }`
        - Local ERTP issuer whose assets should be pegged to `c`
    - `transferProtocol?`: `{ protocol }`
        - Protocol to speak on the connection. Has default value of `ic20-1`
    - Returns: `{Promise<Peg>}`
- `makePegConnectionHandler()`
    - Returns a handler to use with the Network API
    - Returns: `{ConnectionHandler}`

- `getIssuer(brand)`
    - Look up pegged issuers by their brand.
    - `brand`: A `brand` object
    - Returns: `Promise<Issuer>`
- `getNotifier()`
    - Be notified of the current set of Pegs
    - Returns: `Promise<Notifier<Peg[]>>`