Worlds: A distributed MMO *(DRAFT)*

Ryan Walker
ryan.cjw@gmail.com

Abstract. A protocol defining how anyone can contribute to an unbounded universe could allow the flexibility to organically grow an MMO faster than any proprietary system. This paper overviews a protocol that enables developers to bolt their game into in common universe.

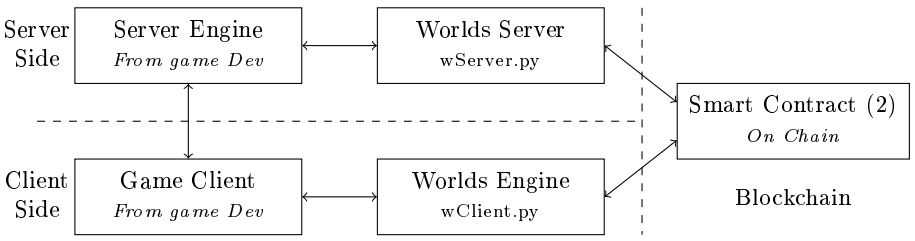
1 Worlds

1.1 Introduction

For an open software ecosystem to grow organically there should be outlets for contribution. In the context of a massively multiplayer online game the outlets become more complicated then a simple code repository. Fair game mechanics are built on a fragile ecosystem that have negative consequences if managed improperly. Forming consensus on a network is trivially done with conventional methods. A server maintains a secure connection with a player who pipes actions to the server. The server then provides ground truth for the network. Our model replacing a centralized organization with a smart contract which forms consensus and balances the economics of the universe. The user experience is designed by anyone who wishes to contribute.

1.2 Overview

A world, defined as w_k, is a node that designs a user experience and hosts players. Worlds provide the game client which the players interact with. In addition they govern their world and create the elements that make it unique. A world can have up to four adjacent worlds determined by itself. Thus forming a network, which allows players to explore and move common items (5), experience (4) and currency (6) between worlds.



The worlds engine is an application that manages the elements that are common throughout all worlds, such as player keys, hashing+signing and fetching the files required to run the game client. The world designer has full creative freedom over the game client and server engine. This is what will enable the highest flexibility and the most unique universe. It is possible for clusters of worlds to share a single game client, yet design their own experience. Conversely, adjacent worlds might have completely different game clients, where players might move from a text based MUD into a fully 3D world.
The worlds client takes care of fetching the executable game client from the worlds server.

2 Worlds Smart Contact (WSC)

The worlds smart contract is used to form consensus among the network. It contains functions and data elements that are used as various forms of proof that require interoperability between worlds. An appropriate blockchain with the following requirements has not yet been selected.

- Low networks fees
- Fast transactions for in game purchases
- Decent data capacity to hold merkal proofs

3 Actions

Actions are defined as anything a node can do to effect the network. Messages and data packages are to be considered actions in this context.

3.1 Direct Action

A direct action is any action that is travels between the Server Engine and the Game Client. These actions have the lowest latency and lowest security. An example of an action like this might be the Server engine telling the Game Client of some in game event.

3.2 Proof Action

A proof action are actions that require the use of private keys for signing, but then relay the signed package from the Game Client to the Server Engine. For nodes that require higher security it is possible to hold the Worlds Server or Worlds Engine offline therefor keeping their keys offline.

3.3 Contract Action

Contract actions are any actions that communicate with the Worlds Smart Contract. They can be issues from the Worlds Engine directly, or the Game Client can pipe action requests into the worlds engine.

4 Experience

Experience is distributed to players using experience packages. These are of the form (Figure 1). When a player is to received experience this package is hashed, signed and sent to the *Worlds Smart Contract* (2)(WSC). The world must stake some WOR in order to grant experience. Experience is distributed like items, although it is not transferable. It is possible for experience to be liquidated back into WOR by the players. Worlds can decide the effect of player experience, typically the more WOR that is staked the larger the effect can be. It is possible for players to stake their own WOR and buy experience. However this experience is stamped with the public key of the issuer and it's possible that worlds might not accept this during the worlds transfer.

Fig. 1. Experience Package - Sword Fighting

```
ExpClass = 'SwordFighting'
ExpHash = hash(ExpClass);
PlayerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
WorldPublic = 0xb1abdaf3ab936c99f5fd518122cf7d5b811a1a30
IssueTime = UnixTime #Time of Issue
Stake = 0.1WOR
```

4.1 Receiving Experience

In the interest is keeping latency low, experience is distributed to players when they leave the world. Although players may use gained experience immediately in the current world.

5 Items

Items can be introduced by any node on the network. To spawn an item the item package, defined as i, must be signed by the node and submitted to the *worlds smart contract* to prove ownership and lock the funds associated with the item. The smart contract verifies that the WorldPublic field is the public key of the node issuing the item, assigns current unix time and then moves the WOR into the smart contract. Worlds have the option to credit or discredit certain items based on the WorldPublic field. It is possible for items to be liquidated back into WOR by calling a function on the WSC. As only the hash of the item package is kept on-chain players are required to keep the entire item package.

Fig. 2. Item Package i - Wood

```
ItemName = 'Wood'
ItemClass = 'Material'
OwnerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
PreviousOwnerPublic = 0xgf7dsa7ejdb5370e59fddb5f6079a354ugf84bdv
WorldPublic = 0xb1abdaf3ab936c99f5fd518122cf7d5b811a1a30
UnixTime =
Stake = 1WOR
ItemHash = hash(ItemName | ... | Stake);
```

5.1 Item Transfers

By calling the TxItem(ItemHash, PlayerPublic) function on the WSC it is possible to transfer items to other players.

5.2 Staked WOR

In order to have a balanced economy the monetary system needs to be net zero 6. The same holds true for experience and items, this balance is achieved by staking WOR into items and EXP. Without this mechanic worlds would be able to create as much wealth as the want resulting in over powered items and EXP.

5.3 Forging

It is possible to forge items by combining multiple items, thus forming a derivative item. In literal sense the act of forging it to call the function *ForgeItem(Item1, Item2, ...)* on the WSC. This will then make a derivative item which is shown in Figure 3. It is up to the worlds to decide on the item names and item classes of these subitems. As an example, a player may forge a sword by combining metal, fabric and a sharpening stone. As long at the world recognizes that the resulting hash of these items do in fact form a sword, then the player has crafted a sword for use in these worlds. The power of the sword would be proportional to the amount of stakes WOR in all the items that were used to craft the sword. Once items are forged into something they can no longer be used for anything else. It is possible to unforge an item.

Fig. 3. Forged Item i_f

```
ItemHash = hash(ParentItemHash_0|...|ParentItemHash_n)
PlayerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
Stake = sum(ParentItemStake)
```

5.4 Using Items

In order reduce the latency of item usage, things must be taken off-chain. Please see the section 8.1 for more information on this.

6 Currency (WOR)

A common currency between worlds will be used for staking and integration into the platform. To prevent abuse this currency must be zero sum, meaning worlds cannot generate it and it must start with a fixed supply. Ideally it is tracked through an existing blockchain. Worlds can introduce mechanics for generating revenue, like entrance fees, subscriptions, purchasing items or keeping players items upon a death.

7 Transport

To prevent players from playing on multiple worlds concurrently a player location is kept on the worlds smart contract. This location is the public key of the world that the player currently resides. In order for this to be valid it must be signed by the player and the world the player resides in. If a player wishes to travel to a world...

- The player must send a signed entry request package must be sent to the world.
- If the world only allows travel from adjacent worlds, the player must reside in an adjacent world.
- The player provides proof of items and experience.
- The world might check the public key stamped the items/exp to ensure they are from creditable sources.
- Items or experience are staked if required to be.
- Player is granted access, WSC is updated with player location.

8 World Incentivisation

If worlds want to distribute items or experience they must stake some WOR into items. As WOR has monetary value worlds need a way to generate some revenue. In order for a world to profit, the sum of all revenue generation methods must be less their their distributed items.

8.1 Staking

Some worlds may require you to stake some experience, items or WOR before entering. This is effectively temporarily moving these things into possession of the world. This allows the game to move at a realtime pace that can be much quicker than the blocktime. If the player is to die, the staked things will fall under ownership of the world. The world then has the option to...

– Redistribute the items among other player.
– Liquidate the items for WOR.
– Give the items, or some of the items, back to the player.

8.2 Fees

Just like conventional games, it is possible the worlds might charge an entrance fees. These could be one time or reoccurring.

8.3 In Game Purchases

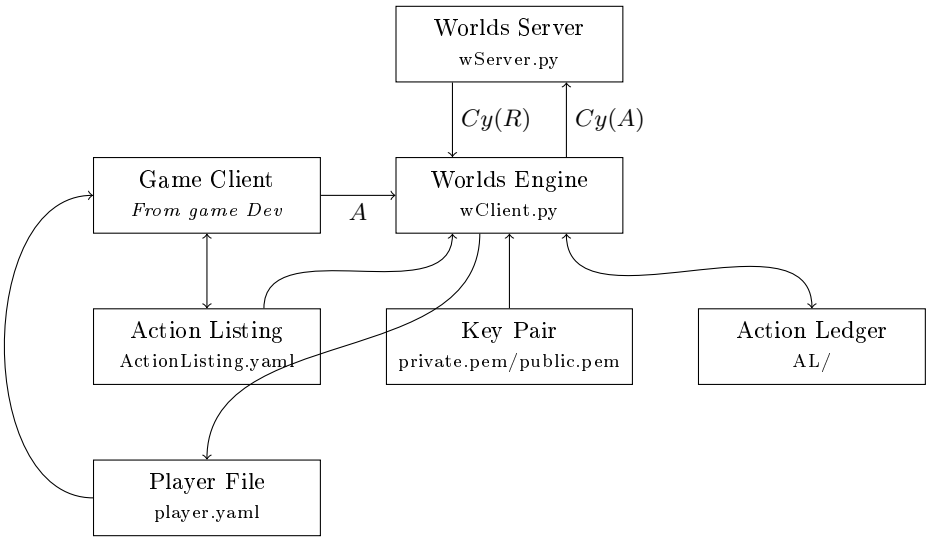
Worlds might wish to sell cosmetic or non-cosmetic items for WOR.

9 Engine Mechanics

The mechanics below are simply suggestions. As the engine is completely open, worlds are free to impose whatever mechanics they wish. Worlds with drastically different game mechanics will probably not be bordering, this limits gameplay but maintains fairness. Players are able to play in whatever worlds they wish - but they must start from scratch in non-adjacent clusters.

10 System Architecture

A: Action
$Cy(A)$: Signed Action
R: Response
$Cy(R)$: Signed Response



11 Outstanding Issues

11.1 Malicious Worlds