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.

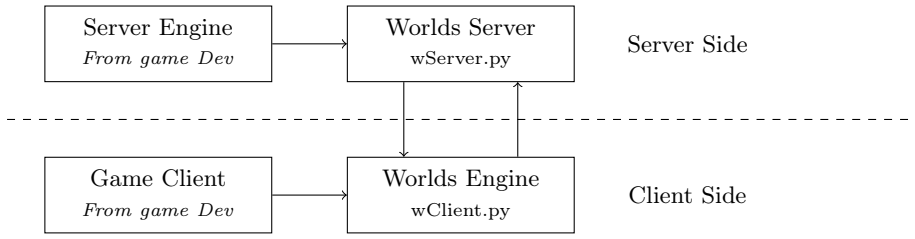
1 Worlds

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. The fairness and security of a game are dependent on it's network forming consensus. This 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. Distributed systems require a very different approach. Blockchains present clear deficiencies, they are too slow and the chain would get too large. An alternate method will now be presented.

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 host servers that form consensus among their world. A world can have up to four adjacent worlds determined by itself. This forms a network, which allows players to explore and share 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. Examples being: player private keys, item ledgers(**??**) and action ledgers(**??**). The world designer has full creative freedom over the game client and server engine.

2 Actions

Actions are piped into the worlds engine from the game client, following this the engine hashes, signs and submits the actions to the worlds server. If these actions are to have a result, the worlds server engine hashes, signs and submits the actions back to the player.

2.1 Action Listing

An action listing is a data construct containing possible player actions. It's possible for a world to make their own action listing, defining an unlimited amount of possible player actions. World reactions are also reside on this list and start at address code: *0x7F*.

Fig. 1. Section of an Action Listing

```
# toolkit/ActionListing.yaml

# Actions
## Movement commands
Move:
 Code: 0x00
 N: 0x00 # Move North
 E: 0x01 # Move East
 S: 0x02 # Move South
 W: 0x03 # Move West
 Up: 0x04 # Move Up
 Down: 0x05 # Move Down
```

2.2 Action Listing Translation

Worlds might not share the exact same action listing. An action listing translation is a data construct used to translate actions from one worlds actions listing to another worlds action listing.

3 Transport

Players, defined as P_k, can enter a world one of two ways. The first being a **player genesis**(3.3) and the second being a **world transfer** (3.2). It is possible for players to transfer to whichever worlds they want. But it's may not be possible to keep their items and stats as typically only adjacent worlds have agreed on exchangeability of items. The world transfer is done using the mechanics discussed below.

3.1 Transport Hash

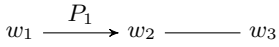
A transport hash is a hash of an action ledger, $h_s(AL(P_k, w_k))$, these are secured by the worlds and are used to prove a player is presenting an honest action ledger. The transport hash is formed when a player leaves the world. A **Forward Transport Hash** is a transport hash kept in a special location. This is explained more in the world transfer section, it is defined as $h_{sf}(AL(P_k, w_k))$.

3.2 World Transfer

A detailed state machine outlines how players can move about neighboring worlds. Honest worlds must follow this procedure, if they misbehave they could be risking a **World Disconnect** (**??**) from their neighboring worlds.

Eg: A player, defined as P_1, wants to move from w_1 to w_2 and then to w_3 (Figure 2), The network values in this scenario are defined as...

Fig. 2. P_1 moving from w_1 to w_2



	w_1	w_2	w_3
$h_s(AL(P_1, w_k))$	$NULL$	$NULL$	$NULL$
$h_{sf}(AL(P_1, w_k))$	$NULL$	$NULL$	$NULL$
Neighbor	w_2	w_1 & w_3	w_2

1. P_1 sends a signed world entry packet to w_2
2. w_2 insures that w_1 is adjasent to itself
3. w_2 must verify that P_1 currently resides in w_1, this is done by ensuring $h_{sf}(AL(P_1, w_k)) = NULL$. This is found by sending a signed data request to w_1
4. P_1 presents $AL(P_1, w_1)$ to w_2
5. w_1 calculates $h_s(AL(P_1, w_1))$ using the AL on the serverside
6. w_2 calculates $h_s(AL(P_1, w_1))$ using the AL provided by P_1, the hashes must match
7. (Optional) An **Action Ledger Traceback** (**??**) can be complete
8. P_1 is now granted access to w_2 and can submit actions
9. w_1 must store $h_s(AL(P_1, w_1))$, $AL(P_1, w_1)$ can be deleted

Fig. 3. P_1 in w_2

w_1 ——— w_2, P_1 ——— w_3

	w_1	w_2	w_3
$h_s(AL(P_1, w_k))$	$h_s(AL(P_1, w_1))$	$NULL$	$NULL$
$h_{sf}(AL(P_1, w_k))$	$NULL$	$NULL$	$NULL$
Neighbor	w_2	w_1 & w_3	w_2

Fig. 4. P_1 in w_3

w_1 ——— w_2 ——— w_3, P_1

	w_1	w_2	w_3
$h_s(AL(P_1, w_k))$	$h_s(AL(P_1, w_1))$	$h_s(AL(P_1, w_2))$	$NULL$
$h_{sf}(AL(P_1, w_k))$	$h_s(AL(P_1, w_2))$	$NULL$	$NULL$
Neighbor	w_2	w_1 & w_3	w_2

3.3 Player Genesis

Player Genesis is the creation of a new player, for this to occur a player must digitally sign a genesis package with unix time. The world must ensure there are no existing values entered for that player. The player is then instated into the world with all the initial player values set to zero.

3.4 Refuge Package

A refugee package is an action ledger that dates to the time that the package is sent. This allows worlds to reinstate players without the need of a world transfer. This is useful for player that have been orphaned into a world. This typically happens during either a **world disconnect** (**??**) or **world outage** (**??**). This has the effect of splitting the player's history into two forks, which opens the possibility of an **action ledger conflicts** (**??**).

4 Experience

Experience is distributed to players using experience packages. This is simply a package that is of the form shown by figure 4. When a player is to received experiance this package is hashed, signed and sent to a player from the world. Players are expected to keep all these packages. This same technique can be used to distribute anything to players that doesn't required transferability or fungability.

Fig. 5. Experience Package

```
ExpPkg{
    PlayerPublicKey
    WorldPublicKey
    ExpAmount
    ExpType
    UnixTime
}
```

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. It is the worlds responsibility to judge which items are credible. This can be done by examining the *WorldPublic* field in the item package, if it has been issued by an adjacent or trusted world the item can be considered real. It is also possible that some worlds might not care if players create items for themselves, in this case the *WorldPublic* would match the *PlayerPublic* field. These options are left completely up to the world creator.

Fig. 6. Item Package i - Wood

```
ItemName = 'Wood'
ItemClass = 'Material'
ItemHash = hash(ItemName | ItemClass);
PlayerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
WorldPublic = 0xb1abdaf3ab936c99f5fd518122cf7d5b811a1a30
Stake = 1WOR
```

5.1 Worlds Smart Contact

The worlds smart contract is a contract that keeps track of player possessions. For a player to own an item, the player must possess the item package and for the hash of the item to live on the worlds smart contract.

5.2 Item Transfers

For a player to transfer an item to another player, the player must call the TxItem(ItemHash, PlayerPublic) function.

5.3 Staked Amounts

In order for the game theoretical elements of the universe to balance, in order to create an item you must stake some WOR to the item.

5.4 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 7. It is up to the worlds to decide on the item names and item classes of these subitems. To forge a sword a player could forge 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. 7. Forged Item i_f

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

5.5 Equipping Items

Depending on the requirement of the world, players may be required to stake items in order to use them. This is a special state where the items enters transient ownership between the player and the world. The world is required to send a signed "pulse" for the player to the WSC to indicate that the player is still alive. If the player is to die, the items will fall under ownership of the world. The world then has the choice to do anything with the items, such as distribute them around the players body for other to collect, return them to the dead player or keep the items as payment for playing in the world. It is wise for world to be transparent as to what happens to player items after the event of a death.

5.6 Using items

If a player wants to use an item, it must be transferred to the world the player is currently residing in.

6 Currency (WOR)

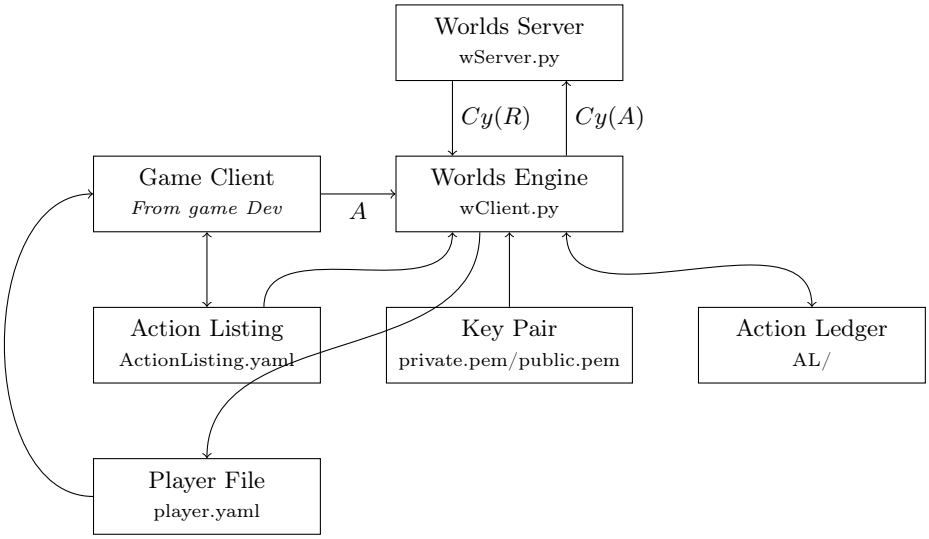
A common currency between worlds will be used for staking and integration into the platform. To prevent abuse this currency must be zero sum, meaning worlds cannot generate it and it must start with a fixes supply. Ideally it is tracked through an existing blockchain. Worlds can introduce mechanics for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

7 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.

8 System Architecture

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



9 Outstanding Issues

9.1 Malicious Worlds