The general design

We wish to design a global data broker that gets messages about a particular topic

. This would likely be an actor or a pool of actors.

From there, we would make sub brokers as a custom query. These would have a defined elixir module type with potential arguments. Users can make more sub queries filters from there, having a chain of topic filters for whatever information someone would want.

Let us exemplify this by having 2 examples:

1. I am interested in all spawned workers from the executor

with Identity 0xdeadbeef

.

1. I am interested in all spawned workers with an even ID

from the executor

with identity 0xdeadbeef

.

This is best illustrated by this diagram

[

無標題-2023-06-20-1630(1)

2294×3116 211 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/25399fe79c50b98b2af0633e826d4274ff9dbb98.jpeg)

As we can see here, we have 4 topic

actors that have spawned up. This for the time being will be done much like an inheritance diagram.

- We have a general topic Topic.OnID

that takes a parameter, id

, the specific ID that we wish to subscribe to messages from.

- Topic.OnID.SpawnedWorkers

is a sub topic

. This topic inherits the argument id

, and presents no new arguments. Meaning that if we did:

Broker.subscribe(Topic.OnID.Spawned_Workers, id: "0xdeadbeef")

For consumer 1

this creates two new topics. The Topic.OnID id: "0xdeadbeef"

, and the spawned workers topic.

- Topic.OnID.SpawnedWorkers.EvenWorkers

is a sub topic

of Topic.OnID.SpawnedWorkers

. This also presents no new arguments.

For consumer 2

this create one new topic. The Topic.OnID.SpawnedWorkers.EvenWorkers

. As Topic.OnID.SpawnedWorkers

already exists, and so it need only spawn 1.

If a consumer 3

came around who just cares about messages from 0xdeadbeef

then, a subscription would cause no new topics to be created as the chain is already set in place.

If all consumers unsubscribe then the topics could be garbage collected.

How we deal with subscribing in engines

Further, Engines should note at startup what topics

they wish to subscribe to at startup. This would make serializing for dumping and restarting easier. We suspect a macro can help us declare this.

We also wish to support transient subscriptions, a with_subscription

macro would be helpful here. Further we wish to have a :flush

flag that notes that for this filter, we wish to flush all messages from it at the end.

Other Designs

Another design is that instead of having an inheritance hierarchy, we could instead use a mixin system. Indeed spawned workers

has no relations to onID

. We could utilize method combination as seen in Common Lisp to achieve mixing these in, however this would take more effort to set up. If we do this statically we would not lose any information, but a naive implementation would lose a lot of efficiency.

I'd say we can do this in the future once the idea is realized.

Implementation Notes

1. We shall make the filters a protocol
2. They should be a standard format and we should respect that
3. They should be a standard format and we should respect that
4. We should have each filter be an actor. With the possibility of making them a pool of actors if pressure is high enouh.
5. We want this in the codebase as soon as possibe.

Notes on The Topic meeting

- Networking discussion
- Ray
- Sub behavior where user chooses which kind of info from Engine one receives
- system of tags (may become unwieldy)
- you can't require event consumers to sort through all tags
- Sub behavior where user chooses which kind of info from Engine one receives
- system of tags (may become unwieldy)
- you can't require event consumers to sort through all tags

- Jeremy
- Engines stating what sorts of stuff it usually broadcasts about
- Engines stating what sorts of stuff it usually broadcasts about
- Ray
- only consumers know what they want to hear about
- only consumers know what they want to hear about
- Jeremy
- Define your own filters
- Define filter as an agent that does filtration for you
- You can subscribe to that agent instead
- We could even graph dependencies
- Have sub-brokers
- Define your own filters
- Define filter as an agent that does filtration for you
- You can subscribe to that agent instead
- We could even graph dependencies
- Have sub-brokers

Questions about how mailboxes getting over-flooded.

- Jeremy
- I want to have an unsubscription and flush
- At the end it would match to any message you subscribe to usually and then match
- I want to have an unsubscription and flush
- At the end it would match to any message you subscribe to usually and then match
- Ray
- Do a receive loop to unconditionally flush
- Mailbox semantics: matched in a receive block then consumed otherwise not
- Do a receive loop to unconditionally flush
- Mailbox semantics: matched in a receive block then consumed otherwise not
- Jeremy
- Should be a protocol for the filter system
- Two layers of filters
- Should be a protocol for the filter system
- Two layers of filters

Who should be a global broker for this system?

Q about robustness

These are robust to crushing

Agree on uniform message struct

How does discovery work? Can be queried up top

See actors as modules so can be queries with naming

Further they can be paramartized, such that if we filter by a specific engine, it will remember, it and register itself so there exists one that people do attach to

How to deal with the broker? What to do with swamped requests?

Slow even if its own actor, will bottleneck.

Can be divided among worker pool.

Supervisor that can clone state among its children for the balanced load.

Not going to be 1-1 relationship of topics with engines as we currently have. Instead, we will launch a large topic on Node launch and then every Engine and user can filter the messages from this large topic.

“engine subscription state” section for startup subscription

prefer for messages to be returned at the end of function calls. may be hard to do. Dependency issues.

want to unsubscribe after receiving a specific message.

Ray against “with_subscription”. If something’s asynchronous you should be able to yield.

Discussion about Mempool-Executor coordination.

Additional meetings. Weekly. Discussing codebase.