Engines are multi mailbox actors. In particular the sender decides what mailbox of the receiver receives the message.

type EngineMsg M := mkEngineMsg@{ sender : EngineID; target : EngineID; mailbox : Option MailboxID; pattern : CommunicationPattern; kind : EngineMsgKind; msg : M; };

MailboxCluster (S M : Type) : Type := Map MailboxID (Mailbox S M);

type EngineEnv (S Msg : Type) := mkEngineEnv@{ state : S; mailbox : MailboxCluster S Msg; acq : AddressBook; };

type Engine (S E M C R : Type) := mkEngine@{ status : EngineStatus; cfg : EngineCfg C; state : EngineEnv S M; behavior : EngineBehaviour S E M C R; };

In the types above we can see that the EngineMsg

contains mailbox

which specifies which mailbox in the MailboxCluster

it ought to go to. This I argue goes against the design princes of the event system within Anoma, but before we discuss how, let us discuss what the event system in Anoma is.

The even system within Anoma was designed to allow consumers of events to properly filter what they think priorities are. This means that the creator of the event does not have to predict the potential use cases of the users and preemptively categorize their events. This gives power to the consumers to write custom filters that work precisely for their use cases.

A good example can be found in the Elixir Implementation of the event broker:

deffilter LoggingFilter do %EventBroker.Event{ body: %Node.Event{body: %Anoma.Node.Logging.LoggingEvent{}} } -> true

```
%EventBroker.Event{body: %Node.Event{body: %Mempool.TxEvent{}}} ->
  true

%EventBroker.Event{body: %Node.Event{body: %Mempool.ConsensusEvent{}}} ->
  true

%EventBroker.Event{body: %Node.Event{body: %Mempool.BlockEvent{}}} ->
  true

_ ->
  false
```

end

def logging_filter() do %**MODULE**.LoggingFilter{} end

Used like this

```
EventBroker.subscribe_me([
  Node.Event.node_filter(node_id),
  logging_filter()
])
```

This creates a filter that filters through “logging” of the system namely TxEvents

ConsensusEvents

and BlockEvents

. The subscribe_me

filters for all messages for a particular node and this filter.

Now if we buy into the argument for the EventBroker

design then by the same line of argumentation for mailbox clusters that the Engine

should determine in what mailbox the message should be put inside. This is because this prevents other Engines

from caring about the internal details that the particular engine cares about, and that the Engine who receives messages can classify the kinds of messages it cares most about. Further we can apply the same kind of deffilter

in the Elixir code above to achieve the intended affect.

Therefore I argue that to unify the system design the receiving engine ought to decide what mailbox goes into what mailbox, further I will argue in a follow up post that we should open up to engines in a reflective way for them to determine how they read their own mail.

Thank you for reading