Anoma

Anoma

is a distributed operating system, in the same vain as Urbit

and Erlang

. However as Urbit

took Erlang

to a hierarchical user system, Anoma

takes Urbit

to potentially byzantine actors. Further, rather than basing the creation myth on Noun

’s, Anoma

bases it’s creation myth on resources

and verified content addressed data

.

A resource can be thought of much in the same light as an object in an Object Oriented

system, in that it’s format is general enough to express hierarchical desperate concepts like classes and methods.The content addressed data part of our primitive gives us the ability to have pointers to other functions inside some kind of AST. Text too can be represented with this system quite easily. What makes the system unique is that all storage is verified via the [

resource-machine

](/tag/resource-machine), and is stored in a distributed manner.

What is starkly different from a system like Unix

or inter lisp

is that we do not have files as primitive, the concept can be derived ontop of Anoma

. However this is not much different from how programming languages derive concepts of functions and modules that mimic the hierarchy of folders and files. For a programming system based on Anoma

, we have to conceptualize modules, functions, objects, etc. ontop of our primitives.

Thus I suggest we design the Anoma Level programming system, this system is intended to be the systems programming language of Anoma and will act as a platform that other languages should be compatible with when designed.

Anoma Level

I think a good initial start to a conceptual design of an AL programming language is by looking at systems that are able to conceptualize all conceptual ideas in terms of itself. For these kinds of systems, I’d argue Forth

, Smalltalk

, and Hoon

are the most consistent. Forth

represents everything as an a stack on some underlying array, definitions are stored in a very plain way allowing meta reasoning to happen. Hoon

does a very similar techniques but over trees of nouns rather than a stack on an array. Smalltalk

represents everything as objects, using the underlying VM to bring native integers and array to the system, but in a way that are just objects. For Anoma I think Smalltalk

has the most compelling argument.

A good image showing off how various concepts look like in Smalltalk

[

objects

1168×1106 66 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/e4656df4ea011c2a182e5820d1110e74e032eb6d.jpeg)

People go as far as to say that Smalltalk

isn't a language, but rather a format layout for how objects are represented.

Likewise for Anoma, I suggest we try to model an object store using resources and content addressed data.

A very WIP on Concepts in this system

- Nested keyspaces, let us have a hierarchy for inheritance of meta objects (known as classes).
- Stored data format: resource machine <> storage - #3 by cwgoes
- Stored data format: resource machine <> storage - #3 by cwgoes
- Objects can be stored either in resources or content addressed blobs
- the executable body of an object can be stored in a content addressable manner
- this means that methods/functions are threaded objects, where they point to a fixed content blob for compilation in other entities
- a function f

that calls g

effectively has a strong pointer to g

by nature of referring to it's hash when the code is actually stored (names are a UX feature)

- this means that methods/functions are threaded objects, where they point to a fixed content blob for compilation in other entities
- a function f

that calls g

effectively has a strong pointer to g

by nature of referring to it's hash when the code is actually stored (names are a UX feature)

- Writes to memory must go through the resource machine (RM), however reads only have to go through consensus
- We can have late binding on the content addressed functions
- We can snapshot the system

A very WIP macro deriving some key ideas in terms of resources

Extending the standard Library

;; EXAMPLE: ;; We want to extend Int, in our code ;; We like Bob's prototypes over the stdlib

;; Automatically generated when making one's own extension

(extend a (bob std))

;; If we want to programatically take someone's extension ;; Also generated from the GUI

(include-from bob (parents a))

Defining Classes (Half written code)

(defmacro (defclass name &rest class-info) (let ((dynamic-fields (parse-dynamic class-info)) (static-fields (parse-static class-info))) ;; define what can be in the resource namespace ;; We do this through permissions `(progn (defr ,name (resource) ;; we grab the ambient key at creation time, and check it ;; against the created resource (satisfies ,(get-user-key) (signed (key resource) resource)) (contains ,@(map has static-fields))) ;; define the new relation (defr (new ,name) (resource) ;; the anchor becomes a fixed pointer to the one we mea (contains (anchor ,name) ,@(map has dynamic-fields))))))

A plea

With all the ideas presented I do not know the optimal system ontop of Anoma, however I think a few of us brain storming here could go great lengths to defining a consistent system utilizing the Anoma system.

I'd be interested in seeing what can we really derive ontop of Anoma.

Happy hacking