# The Ceptr Revelation

This document is our first attempt to share Ceptr and its underlying design principles. Since Ceptr is so ambitious in scope with many unusual approaches, it is difficult to pack it all into one document which will make sense to everyone. We're going to try to keep things in a non-academic/informal voice and we'll be switching between high-level overviews and low-level overviews, and will link to supporting articles for technical details.

This document is meant for builders and seekers of tools for enhancing collaboration on all scales -- people seeking to upgrade our collective intelligence. We hope it will help you find out whether these are the right tools for your needs, and if they are, provide ways you can help improve the tools or implement your own solutions with them.

The first sections give context of why and how we built Ceptr. Feel free to jump to the what section if you want to dive straight into the nitty-gritty.

For us, many of the discoveries in this process have felt like a kind of revelation -- now we hope to share that experience with you.

Arthur Brock

Eric Harris Braun

**December 2010**

## Table of Contents

## Why Build Ceptr?

This is the part where we're supposed to quote Einstein about how problems can't be solved by the mindset that created them and tell you about how Ceptr solves a bunch of previously unsolvable problems. We see what Ceptr makes possible as much bigger than the problems it solves, but both are important.

**Problems Solved:** Humankind has built a global electronic communications network which enables us to collaborate on scales like never before. However, the gap between theory and practice in this case is immense. Technical barriers that constrain access make it difficult for applications to interface with each other. Weak tools for large-scale collaboration and decision-making, and the power dynamics of who controls shared data and communications, keep this large-scale collaboration far out of reach. APIs and Protocols compete with each other, keeping our computing infrastructure fragmented and brittle.

What if we could elegantly transcend all of the technical barriers at once? After all, shouldn't our technology actually help us work together? **Everything should just work together easily**.

**Audacious Possibilities:** The social hurdles are just as challenging: entrenched economic interests, upside-down incentives, legal jurisdictions, challenges with large-scale decision-making, setting priorities in the face of numerous challenges, and melding diverse viewpoints into a shared perspective, etc.

Humanity is poised on the edge of a quantum leap in evolution, not at the level of individuals, but at the level of our collective social organisms like corporations, institutions and governments. In order to make this leap, we need the same kind of architectures of intelligence that make it possible for trillions of cells to work together in an organism.

Large-scale collective intelligence requires communication to be virtually instantaneous (electronic), peered, decentralized, semantic and designed to evolve in response to rapidly changing needs. Effective collaboration on such a scale would obviate most of the power structures that underpin the social

barriers to change and could make formerly intractable problems (such as climate change, species extinction, resource depletion, or poverty) quite readily solvable.

Ceptr is designed to provide the building blocks of the kind of expressive capacity which embodies nature's architectures of intelligence and enables an explosion of new patterns of collective intelligence on every scale.

## How was Ceptr Designed?

Ceptr emerged from years of work that was rooted in just a few core commitments:

1) Learn from Nature
2) Make it "Unenclosable"
3) Foster Coherence and Follow Convergence

### Learn from Nature (physics, biology & consciousness)

Examples of how things work together at a large scale surround us. We don't worry about whether an atom can interact with any other atom in the universe, or how far light can travel. The world around us contains many layers of protocols (gravitic, electromagnetic, subatomic, atomic, molecular, organic, cellular, neural, hormonal, mechanical, organismic, ecosystemic, etc.) which provide powerful models for how to build universal protocol stacks. Patterns of collective intelligence of the cells in our bodies embody key principles for decentralized rules and holographic data storage. Fractal patterns of trees, leaves, and circulatory systems offer us lessons about distribution and routing.

What if fundamental patterns provided by nature could form the foundation of a computing architecture? It turns out that when we follow some basic principles derived from nature, things which previously looked like an explosion of unmanageable complexity are a pretty manageable variety of compositions.

For example, one of these natural patterns is the fact that there is no "CEO cell" in an organism that controls the activities of all the other cells. Rather, there are a variety of signaling systems that all the cells use to coordinate with each other. Every cell carries a copy of the agreements for working together (in DNA) and checks their own copy for instructions. This provides a model for distributed computing and for coordinating extremely complex groups without centralized control, power imbalances, or central points of failure.

Another example is how meaning is transmitted in the world. There is a pattern which seems to hold true whether at the level of subatomic particles "communicating" with each other, or cells, or natural organisms, or humans. Whether in writing or speaking. Whether analog or digital. In every case, a CARRIER for the communication is transformed to embody variations according to a PROTOCOL. The particular variants embodied on the carrier are the SIGNAL (or data). And there may be a SEQUENCE of these kinds of signals in succession.

You are seeing letters on a page (or screen) that you're reading right now by virtue of your ability to receive the CARRIER of light and an ability to differentiate variations of lightness, darkness or color. The

letters themselves are shaped in accord with a PROTOCOL for latin-based characters. This particular document, as an assemblage of these characters, is the SIGNAL. The fact that you're reading them left-to-right in succeeding lines down the page governs the SEQUENCE.

This may seem kind of obvious. This pattern holds true for electromagnetic fields at the subatomic level, hormonal/chemical communications between cells, auditory and visual communication between organisms, TCP/IP packets on the Internet, device drivers your computer uses to talk to every device and subsystem. If you understand these elements of every communication protocol, it creates the foundation for building a self-describing protocol stack.

In much the same way that XML enables people to share self-describing data, this provides the means for a whole new computing and communications stack which is optimized for flexibility, adaptability and rapid evolution. Having a native framework to specify protocols makes interfacing with any existing protocol quite straightforward. It also makes it very easy to build new protocols for any specialized type of communication or information representation format that may be needed.

In every section below that describes some of what Ceptr is, there will be examples of the natural principles that form the basis of our architecture, and details about how they actually work.

Ceptr has the potential to be a quantum leap in our approach to computing and with it our ability to coordinate and communicate more effectively with each other. It may make whole new kinds of hardware, software and networks possible. We are working to make sure it interfaces easily with existing computing and networking models, but is not constrained by them.

## Make it "Unenclosable" (Open and Peered)

We are committed to having an open architecture, but it turns out our understanding of "Open" had to evolve significantly. The original MetaCurrency web site explained what we were building in four simple elements:

- **Open Identity:** We never wrote much about our design for identity. Our plan was to borrow a lot from folks we thought were getting this right with XDI/XRI/iNames. Ceptr's identity model is similar in many ways with the biggest differences being that we use contextual name resolution instead of having a global top to the namespace.
- **Open Transport:** Most people think of the Internet as already providing an open transport, and much of its success in spreading was because it was much more open than proprietary network protocols. However, the fact is that there is a central authority for addresses and names (even if they delegate some powers to ISPs and registrars). ISPs choices diminishing toward regional monopolies, the structure of root servers for DNS, and governments implementing "kill switch," point to some of the reasons why we need an even more open infrastructure that could operate completely peer-to-peer.
- **Open Data:** Adapting ideas from Ian Grigg about triple-entry accounting and Ricardian Contracts, we made presentations and published information about decentralized data models with signed

transaction chains to ensure Intrinsic Data Integrity. Variations of this approach that have become popular are git as distributed code sharing and versioning system and the bitcoin blockchain as a cryptocurrency platform.

- **Open Rules:** Having a format for sharing the rules, actions and actors in a currency or collaborative system as simple as HTML. We initially implemented this with XGFL (eXtensible Game Format Language[1]). You could drop an XGFL spec into a flowplace and the system would render the correct interfaces and perform the proper operations. We then discovered the issues of interoperability between rule-sets were much more complex than could be sensibly addressed by an expressive capacity akin to XML.

The word "open" as in Open Source or in our four elements above wasn't clear enough. It sometimes means "transparent" or "visible." Sometimes it means "shared." It also seems to mean "interoperable" or something like "own your own" (data & identity). Calling all those things open sounded nice, but we quickly discovered we needed clearer requirements to guide what we built.

Our experiments in building an infrastructure appropriate for the next economy made one thing very clear. In order for next-generation digital currencies to solve the core problems of our existing economic system, immunity to these problems would need to be built into the most fundamental levels of the tools used.

For example, operating on a network with "kill switch" or the ability for a handful of ISPs to filter out "undesirable" packets was unacceptable. So is the power imbalance stemming from banks being the only ones allowed to create money, which enables them to manipulate the economy, our politics and the value that everyone else creates. These problems would not be solved by digital currencies shifting the control from bankers to system admins. Since a system that can be corrupted will eventually get corrupted, we needed to make sure the underpinnings of what we were building had freedom from undue control, influence, manipulation or abuse built into the core of its DNA.

Looking at models in nature made it clear that we mean something like universal (rather than transparent), unenclosable. Finish clarifying "open."

If "open" means fundamentally unenclosable, then it is also clear that Ceptr must be decentralized. This means a fundamental shift toward Peer-to-Peer (P2P) computing architectures.

## Foster Coherence and Follow Convergence

Rooted in these principles (learn from nature and make it open & peered), some very interesting patterns started to emerge. We discovered there were connections that wanted to be made to complete the picture, and it turns out they're not so hard to build.

For example, we didn't set out to build a semantic system, but because Ceptr needed self-describing protocols, and because we preserve coherence contexts for each receptor, it ends up being a fairly small step to build semantics in at the very base level. From doing so, we got a huge leap in the power of our system. We knew we were going to need to build some sort of generalized protocol parser to be able to

[1] See a working sample of currencies specified in XGFL or our pre-XGFL white-paper

work with self-describing protocols, and once we added semantics, it turned out to be pretty easy. We expanded the idea of regular expressions and built Semtrex (SEMantic Tree Regular EXpressions) which provide lexical and syntactic parsing at all layers of a protocol stack.

We have only begun to fathom the magnitude of possibilities for collective sense-making that adding semantics has enabled, and it was a fairly small technical cost for us to include. The trick was in discovering this connection wanting to happen, even though it wasn't a problem we had originally intended to solve.

We understand that patterns in technology adoption could warn us away from changing too many things too fast. There are numerous examples of social adoption of new things veering toward inferior technical solutions rather than embracing something less familiar. However, we began to see that putting certain things in place created a kind of elegant simplicity where a few significant changes at the outset, solved numerous other problems that we hadn't even started to focus on.

This kind of whole systems design is what has led us to the current design of Ceptr. We've had a chance to learn from numerous attempts (by ourselves and others) at assembling a solution. There are hundreds of projects trying to solve these problems. Typically, each one is trying to solve their part in relative isolation of the other parts, building their solution on a flawed underlying architecture. Assembling all these solutions together creates massive complexity and brittle interdependencies.

We're not claiming our team has more expertise than all the people working on these solutions, nor that we could accomplish the same work of all those projects are doing. However, we have discovered that stepping back and altering some underlying assumptions about how to structure computing and protocols create an enormous convergence of the solutions which makes them all more readily achievable.

## Some Domains that Ceptr Addresses:

**DISTRIBUTED COMPUTING**

- validation of data integrity (no matter who possesses it)
- byzantine fault tolerance
- decentralizing control of shared rules/activities/data/projects for groups
- embodied contracts / "code as law"
- power balances between groups and individuals

**PROGRAMMING & DEVELOPMENT**

- reusability of code and data
- modularization / separation of concerns
- coherence management
- signing of code and apps
- deprecation of outdated tools

**IDENTITY & TRUST**

- open identity
- personal ownership (and control) of data
- domain name squatting / namespace competition
- authentication (of almost everything)
- decentralized social networking
- transparency and accountability
- trust networks and authority for certification

**NETWORKING INFRASTRUCTURE**

- P2P/mesh networking
- Workarounds to ensure net neutrality
- network resilience (no off switch)
- decentralized networking architecture (no central addressing / naming authority)

- version management of updated tools
- language flexibility / independence
- native preservation of syntax/structure
- UI for non-tech programmers using flow visualization tools (future)

- privacy, surveillance, and sousveillance
- evolution and replacement of protocols

**SCALABLE FIDELITY** (name?)

- privacy and encryption
- transportability of live systems
- redundancy/resilience/mirroring
- sharding/segmentation of large distributed systems

**ENABLING NEW COMPUTING PLATFORMS**

- Parallel processing
- Analog computing
- Interface w/analog communication protocols?

**BUILDING & SHARING MEANING**

- semantic web / semantic computing
- open and customizable communication protocols
- common data structures and world models
- knowledge building and sharing
- collective sense making
- distributed autonomous corporations

**BUILDING & SHARING VALUE**

- decentralized currencies
- new economic infrastructure
- reputation currencies
- custom transactions
- infrastructure accounting
- mutual (& fractal) sovereignty
- DHT/distributed file sharing (like torrents)

Still to write/expand:

- Yet, we will still need lots of help to complete these tools, build an amazing user experience, and spread the tools.
- Listening for the deeper wisdom of the pattern
- Freedom vs. Agency

## What Makes Ceptr Work?

**DISCLAIMERS:**

**For Geeks:** Kudos for making it past those initial sections which may have seemed annoyingly naïve. If it was so easy to make everything work together, we'd already be doing it, right? We think there are some real reasons this hasn't happened before, and why it is possible to make it work now. In this section, we will get into the technology and how Ceptr works, but as you've noticed this is not exactly a technical white paper. For deeper technical detail, we will link out to other supporting articles and resources. The structure of Ceptr may initially seem unfamiliar or disorienting, but our hope is that, that you'll find it

worth the effort to try to re-orient yourself to its frame of reference so you can crank out some powerful solutions.

**For Non-Geeks:** You may have a strange advantage if you're not too deeply steeped in the way computers conventionally work. We hope to make the architecture of Ceptr understandable with examples of how these structures already work in physics and nature. Let us know if we miss this mark. We're trying to keep the technical gibberish in other places and just link to them from here.

## Distributed Fractal Receptors

One of the fundamental organizing units in Ceptr are Receptors. Receptors are kind of like super-lightweight virtual machines.  Each one performs a particular niche in the computing ecosystem. {See demo}

Since receptors can be set up to run inside another receptor, they can be organized fractally and can function on surprisingly different scales ranging from the level of what you might normally think of as functions (like a bubble sort, or keyword indexer), applications, programs, platforms, and operating systems (unifying multiple platforms and programs), to the level of the whole network itself.

RECEPTOR is what we based the name of the Ceptr platform on -- from the Latin adjectival form of *capere* meaning "to catch; take in, hold; be large enough for; comprehend."

### Our Receptor Breakthrough

One of our big breakthroughs in our system design came when we were looking at how to maximize composability. In contrast to our foray into XGFL, we wanted it to be easy for everything to be functionally mashed together.

We were exploring language and how amazingly composable it is. How, from a traditional computer science perspective of starting with ontological units, the conversation we were having was all constructed out of a couple dozen phonemes, which we used to construct word parts, and in turn constructed words, then phrases, then sentences, then narratives.

This way of thinking seems completely valid – even obvious. However, in another way, it is also completely wrong.

It misses something fundamental that makes human communication so different from computer communication of today. If you spoke with a strong French accent, or had a speech impediment, you would be using a very different set of phonemes. Even if you were unable to say up to half of the sounds the expected way, many people could still probably understand you. And as you continued to speak with someone, their ability to understand you would increase.

So if everything was actually built out of those base-level phonemes, all of later/dependent layers of processing and understanding would be broken and you wouldn't understand anything. Garbage in, garbage out. This would make the prospect of mutual understanding quite brittle and fragile, which is indeed the case with most computer interfaces, where unexpected symbols can mess up everything. The

ability to do somewhat independent sense-making at each of these layers is critical. And the layers themselves seem to be connected to different expectations, symbol sets or layers of meaning.

In fact, it turns out that first and foremost, we have the ability to receive a general carrier for the symbols rather than the symbols themselves. In this case, the hairs in your cochlear receive sound waves by vibrating to different frequencies. They receive the general carrier of sound, not just specific phonemes. Just like our eyes receive frequencies of light, not just specific letters or objects.

Receiving the carrier first allows us to attempt pattern matching for phonemes, words or phrases on soundwaves, or for different objects or patterns on lightwaves. In fact, we have the capacity to hear a word or phrase and if we don't recognize it, ask what it means, and "install" the ability to understand it in the future. Computers (other than AI learning systems) don't normally have this capacity to say, "Unknown symbol or protocol. Please install it in me so I can continue processing the message you already sent." Ceptr enables this process. Not by being an AI/learning system, but by having a built in ability to define and receive semantics, protocols, and carriers.

Once we started thinking this way, its importance grew. We started seeing how RECEPTIVE CAPACITY organizes the world: like our "five senses," how RNA receptively holds the pattern that builds DNA, and how atoms have valences which want to receive a certain number of electrons which become “slots” for molecular bonds.

When we walked into a coffee shop, we'd see how some surfaces were receptive for walking (able to bear full body weight, clear of obstacles, etc.), others for butts to sit on (at a comfortable height, stable, soft to sit on, etc.), and others for setting food and drinks on (smooth, stable, not likely to get kicked, sat on, or spilled, etc.). We started seeing how these types of receptivity shape all interactions, from subatomic particles to solar systems, from social interactions to technology systems.

For us it was a beautiful inversion. Instead of only seeing the objects that exist, we started seeing the receptivity that gave them the space in which to come into existence. The subtle power of yin became clearer instead of just the solidity of yang.

And we embarked on a mission to have computers work this way.

### How Receptors Work

Receptors are the “atoms” of the Ceptr computing architecture, and the basis for the name of the platform. A receptor is a lightweight virtual machine, which can run processes, manage its state, and most importantly RECEIVE (and send) signals. They embody a fundamental unity, and hold coherence:

- for the version of their executable code
- for the state of their data
- for relationships within collections of data
- for how they interface with other receptors (including other distributed instances of themselves)

A running receptor is always instantiated in the (address) space of another receptor. In other words, receptors can contain other receptors. This extends, in a fractal manner, with each receptor able to manage a particular realm of coherence as an integrated service inside other receptors.

You may be thinking: "Every receptor in a receptor? It has to bottom out somewhere. It can't just be 'turtles all the way down.'" This is made possible by a trick of constructing a strange loop in how receptors interface with computers and networks akin to biological genotypes and phenotypes. Every receptor runs by virtue of process threads allocated to it by a Virtual Machine Host. These VMhosts are themselves a special kind of receptor which interfaces with the operating system of the host machine.

### Bootstrapping Ceptr via a Strange Loop

The easy part: Suppose you grab a receptor from the Ceptr Compository that does something useful that you want. (Think of this like getting an app from an App Store.) It will have a unique Compository ID, and when you install it in your VMhost, it will assign it an instance ID.

Where it starts getting tricky: Suppose you installed a shared instance receptor for a cryptocurrency. Like DNA in cells, each user of that currency runs their own instance of that receptor in their own VMhost, and the receptors synchronize data with each other to keep track of everyone's currency balance. Each instance shares the same Compository ID which is part of how the instances keep track of each other, but they also have a unique instance address within the VMhost running it.

The strange loop part: I mentioned that VMhosts are a special kind of receptor. They have the ability to provide services for interfacing with the operating system (to reach physical devices or the network) they're running on. The VMhost uses a UUID algorithm to generate the address it is known by in the Ceptr Network. But just like every other running receptor, every CeptrNet receptor actually runs inside a VMhost. Every CeptrNet receptor has the same Compository ID (genotype), but each runs as an instance (phenotype) inside a VMhost. Each VMhost (genotype) is a node at an address (phenotype) in the CeptrNet. (Write a standalone post on this with diagrams)

==== End Strange Loop Section // which may be too geeky and could be moved elsewhere ====

Back to receptive capacity as the foundation upon which everything in Ceptr is built – the ability to specify carriers and protocols makes it possible for a receptor to receive a signal, discover that it doesn't know how to interpret the signal, and then install the protocols required to process it and respond to it appropriately. This adds a level of resilience to computer communications that seems sorely missing today.

Still to write/expand:

- Example: Q&A grammar
- Fractal structure and coherence:
- How receptors are VMs that are lightweight enough to be able to nest fractally… Reference shared system defined structures, symbols & processes in the VMhost. Shared data engine handling, memory, disk, network & synchronization.

- Distributed data engine and byzantine fault-tolerance:

NEW OUTLINE: HOW RECEPTORS WORK:

- State & Coherence (link to data engine for persistence/replication among instances?)
- Fractal structure (link to with solution to turtles all the way down)
- Semantic Trees to represent Carrier/Signal/Protocol (a few paragraphs with links to semantic…/semtrex? also to self-describing protocol stack)
- Relational scaping (one paragraph with link to section)
- Processing (it is a virtual machine after all)
- Inter-Receptor signaling within one VMhost (link to network for between hosts)

## Relational Scaping

Each receptor has its own coherence context and one significant part of coherence is maintaining the relationships between different atomic data elements. In Ceptr data is stored not in a structured SQL database, but as atomic objects or elements. Scapes are used to manage collections of elements as well as represent the relationship between those elements.

Scapes embody the dimensions of variability that are used for making sense of data. In other words, if you need things arranged by size, or by direction, or by alphabetical order, or whatever the geometry of sensemaking that is embedded in the data, Scapes are encoded to organize the data according to that relational geometry.

There is a deep need for relational intelligence between receptors because it's our observation that natural systems have ways of embodying healthy relational wisdom in many different contexts.

Amongst the human body's trillions of cells there is also a vast differentiation of roles. In the immune system there is an identification of healthy participants vs. unhealthy participants within the body, bacteria can mutate to fill different niches, and more broadly it seems like there are patterns that help make a "healthy state" and patterns that are unhealthy. What would be the criteria for the body to be able to assess whether something is a bacterial infection that should be filtered or attacked or it is helpful? What are the relational dimensions the white blood cells could be using to make a determination of that?

(Don't know: unclear how white blood cells determine.) Explore.

- How relationship is characterized by the realm of possible relationships, rather than X to Y
    - What is relationship? It seems when you think of it as two things, it seems arbitrary. This is southeast of that, it's heavy, etc. arbitrary relational desc. The problem with traditional semantics / what they're doing with RDF etc. is that they're just making arbitrary assertions. What is missing in that picture for real sensemaking you don't want just arbitrary assertions, you want to understand what differences make a difference, the dimensions of variabilities that matter.

Would that be possible if there isn't a relational scaping happening within the body? That is why scapes seems like a better way to do this rather than RDF assertions.

Another profound example that guides our thinking about relations scaping comes from geometry.

    - Data Geometries: sequence (nth/index, len), list (prev, next, len), map, hash, array, tuple (x,y), tree, enum
    - Process Geometries: set (logical) membership, Lat/Long (maps x,y to a sphere)
    - More at this link (another doc also exists with more details on scape geometry)

- Ex: Take geolocation data: doing a couch matching example. You want to say who is nearby? Nearby changes a lot based on certain types of context. WHen you're out in a rural area in a car, maybe a 25-50 mile radius. In NYC, taking transit, maybe a 3 mile radius. Function and mode that makes something "nearby." But the big point is the context. Depending on the data set density nearby can be very different things. Even if you're putting in data that has very meaningful constraints, (ex: lat -90 to 90 degrees). You know something about its relational context even when that data isn't in there.
    - The thing that gives you the sense making is the context of the range and dimension of change that matters, not the items themselves in the data set. So in RDF (done with triples) item + item = relationship (defined) between them. So RDF has to explicitly declare the relationship between (every) elements in a data set instead of naming or identifying a range of variability within the elements and then being able to evaluate an element according to that relational context.
    - This is the need for scapes. The sense making with trad. semantics is not identifying contexts, ranges, and variability in that range.
- The nature of "relationship" → scapes
- The knitting of texture in collections of data instances / landscape / also like protein folding
    - I think DNA --we have names for different types of chromosomes because of the shape the DNA takes in that form of the chromosome (the way it folds on itself). Wondering if there aren't similar kinds of layers of encoding and complexity that you can get out of scapes that is sorta like protein folding or you have a number of relationships that when folded into another relationship folded into another relationship you get these emergence of higher orders. It seems like we need an architecture that will relate that sort of relational sophistication.
- Scape Algebra
    - We think it is going to be possible to perform a kind of "scape algebra" where you have crossing and intersections and topological mappings using scapes. Think of in the context of couch surfing. Where you have a whole bunch of criteria. You have a "comfort scape" how well does the place fit your comforts? Allergies-privacy-. A range of compatibility, as well as location, as well as social compatibility. When you return results on a search you want to be doing a kind of blending, crossing, intersecting (algebra) blending.

## Semantic Coherence

Still to write/expand: Until this section is written, you should check out our MIT Webinar about Ceptr here: MIT/KIT Ceptr Webinar

- Semantics woven into the fundamental levels. Cannot create an instance of an Int, but can of an Age (a semantic use of the structure of an int)
- Semantic trees as basic structure of storing all information
- The pattern of semantic alternation in the world. How we implement this pattern. Structure / Symbol / Structure / Symbol / etc.
- Hiding a multitude of sins in an ASCII string (example, no structural coherence for versions)
- Local / System / "Global" namespaces
- Semantic Stack
- Ability to include "configuration" elements of a receptor in the manifest process by referencing semantic parts in the scope of its parentage or the system.
- With proper coherence & membranes, privacy becomes an issue of configuration

## Self-Describing Protocol Stack

Still to write/expand:

- Pattern of Signal/carrier/protocol/flow & geometry of addressable space or existence scape that repeats throughout all of computing (and nature)
- [DIAGRAM] Carrier / Signal / Protocol + Flow/Sequence/**Pattern**/Series (reference CAS working paper) C →S/P
- Make a chart of examples -- both in technological terms and bio/physical
- Base Ceptr protocol which identifies the protocol being used as well as semantic identifiers for the symbols. Anything which can speak the base Ceptr protocol can then communicate with / interact with anything else that speaks the base protocol
- What is typically called a "protocol" in a specification is for us an EMPTY protocol template which does needs to have its endpoint connection slots be connected upon instantiation. A full Ceptr protocol specification actually has the ability to function -- to be installed and do what the protocol is supposed to do. What good is a self-describing protocol stack if the protocol don't actually have functioning descriptions which make the protocol work.
- Slots in: connections, roles, goals & usage - **descriptions of all this**
    - basic gist is like this: In an abstract definition of a protocol there are always slots which need to be filled when that protocol is installed or instantiated for use.
    - CONNECTION: is an endpoint binding for a protocol (could be used at either end)
    - ROLE: specifies the RECEPTOR_ADDRESS of an agent (including another receptor as agent)
    - GOAL: specifies a PROCESS to be run
    - USAGE: specifies some semantic data to be used/passed in the signal
- protocol chaining & embedding - **description of how that works using CONNECTIONS and such**

- Semantic Context for interpretation of symbols. Local symbol tables or contextual reference via Ceptr Compository for non-local (Ceptr network) communications. Always able to receive carrier, but not necessarily signal. Could install the components to process received/logged signals.
- SemTrex (SEMantic Tree Regular EXpressions) provide lexical and syntactic parsing at all layers of a protocol stack.
    - In order to have a self-describing protocol stack that can interact with ANY protocol, we needed to build a universal protocol parser.
    - It turns out that wasn't as difficult as it might seem because Ceptr relates to data natively in semantic trees.
    - In typical computing, we regularly dehydrate and rehydrate data… like when we serialize objects or read objects back into memory from serialized storage. The Ceptr base protocols provide consistent methods for serialization/deserialization of semantic trees, so we can always relate to data in its semantic tree form.
    - ASCII text is used as a generic linear storage format for things like writing programs. For those programs to run, a compiler or interpreter needs to identify all the meaningful sequences of carriers (lex the tokens) and then build a representation of the relationships between those tokens (parse them to build the Abstract Symbol Tree).
    - When your signals come as semantic trees, a universal parser is easier because we already have the signal lexed and parsed (tokens are identified by semantic IDs and the parse structure is embodied  in the tree structure)
    - This allows us to simplify the representation for parsing any signal into a format similar to how regular expressions match sequences in ASCII strings
    - With this we can match signal components by 3 dimensions: 1) their semantic identifier (such as SHOE_SIZE) 2) their value (such as 45) or 3) their grammatic placement in the symbol hierarchy of the semantic tree (such as /HTTP_REQUEST/*/HOST="ceptr.org" … find any HTTP_REQUEST with a child node with a semantic ID of HOST where its value equals "ceptr.org").
- EXPECTATION Statements: ON <carrier> EXPECT <semtrex_expression> [DO <action>] [WITH <Parameters>] [UNTIL <time_condition>]
- Receptor aspects can be bound to protocols
- Expectations can be placed on particular aspects… or even be requested to be planted as remote listeners in other receptors.

## Intrinsic Data Integrity

Still to write/expand:

- Breaking data access from data integrity through cryptographic structures. We call structuring data this way "Intrinsic Data Integrity" [intro para]
    - Signed transaction chains with multiple levels of integration. (akin to bitcoin blockchain) [one para w/ the below simply identifying other approaches]
    - Merkle Trees with accumulating hash akin to git repositories

- Applications can enable individuals to be primary authorities over their own historical data by storing progressive hashes of that data in a shared DHT with adequate redundancy to make tampering impractical without controlling most of the nodes. [couple short paragraphs]
- Signing is a low-level, always available Ceptr system function [sentence]
- All CeptrNet communications are signed, it is possible to keep signed transaction/interaction logs/chains for various purposes
    - right now a log is pretty bogus record. just an ASCII file. if you're sys admin you can very well tamper with that log
    - Non-repudiation: if you want to keep tracks of audible histories of things you can create audible histories that show that at least someone had the person's sig / public keys. Doesn't mean for sure that person, it does mean they were using that person's public keys. Makes for a smaller number of people that could potentially have those keys.
- Byzantine Fault Tolerance/distributed consensus across instances of a receptor.
    - Multiple realities (versions of hash tree history which are different across nodes) are an option and are sometimes exactly what is desired. The integrity of each forked reality can be assured/maintained. Sometimes the discrepancies may be easily resolved, other times not.
- Configurable levels of data assurance/redundancy in distributed patterns based on application requirements

At the core of cryptocurrencies is the tamper proof nature of historical record. Accomplished through cryptographic data structures which have self-referencing hashes to ensure the data can't be changed. Such as Hash Chains and Merkle Trees. This enables us to handle data access distinctly from data integrity, because people can be allowed to host/access data without being able to break its integrity.

This means we no longer need to hide important data behind firewalls. We can share it openly and even in distributed storage structures such as DHTs (like bittorrent, IPFS, etc.)

Historically, computing architectures have conflated data access with data integrity. If you can write to it, you can change it. So we build sophisticated security systems to keep unauthorized people out. Imagine what people would do if they could alter their bank's database.

However, structuring data in countersigned, linked chains creates a kind of Intrinsic Data Integrity where the integrity of the data is structured right into the data itself instead of the firewalls and access controls separating you from that data. If you tried to alter your most recent transaction, you'd break the signature of your countersignor. If you tried to hide or remove or manufacture some past transaction, you'd break the continuity of your hash chain.

Since you can't tamper with your chain of transaction records, you are able to be a trusted authority to represent the standing of your account. Quite unlike the world of bank accounts where the bank is the only authority of your account's standing with them, this provides autonomy without any central authority.

## Fractal Sovereignty

Still to write/expand:

Cells & DNA… distributed governance. Currently political gridlock of group vs. individual and no ability to self-represent, trust/run local copy

- Basic thing about fractal sovereignty is that there is mutual sovereignty at every layer and level. So it is possible to provide for mutual sovereignty at every layer and level. And because receptors are possible to structure fractally, you get this pattern of fractal sovereignty.
    - Mutual sovereignty: Something that is not currently available to us because we don't have the tools to embody this power dynamic between groups and individuals. The group maintains the order or rules of its domain and for people to participate in that domain they have to participate according to those rules. But the individual is not subservient/subjugated -- their liberty is not subjugated by the group in that since the architecture in Ceptr for these patterns involves every individual running and holding their own copy of the group's agreements encoded in a receptor they always interact through their own copy. Which means they always have their own history and records and copies of anything they've contributed when those assets are digital. so the individual can revoke or remove their participation at any point while still keeping the value they contributed to the group. What that allows for is a shift in the power dynamics around which most of our politics is polarized around the power of groups vs. the power of individuals. Allows for the individual to say "Hey this group no longer works for me!" If they can't adapt, or if they're stuck or going in a direction you don't want to go I can go off in another direction and take my toys with me. And invite any portion of that group to come along as well. (Hey I'm forking and changing the rules in this way, and I'll honor the status of anyone else who wants to come with me.) Prevents the group from control individuals' assets. The individual controls their own assets, but they're also shared by anyone who has a copy of those group's receptors.
- Network runs via sharded distributed synchronization
    - Imagine if Wikipedia was hosted by having all the users host five pages. You'd have a couple hundred redundant copies out there, and you could retrieve any of those pages that were nearby. But the hosting load would be very low. It would allow you to have a massive high-volume site w/out centralized hosting infrastructure or a centralized organization.
    - Ceptr allows receptors to be configured to use a DHT style storage with configurable levels of redundancy or data integrity and insurance so that data can be shared in this way. But it's a modified version of that way in that you also have not only whatever randomly assigned portions you get by hashing but additional portion as well of whatever you authored as well. So you always keep your contributions.
- Instances of distributed receptors as Mirror Neurons

    - Distributed process rather than distributed consensus. Much smaller scale consensus can be used to validate integrity of distributed process.
    - The model is: Trust your own node.
    - There are structures that can be able to check the integrity of the code that you are running against the hashes in the compository so you can be sure you're running the exact code everyone agreed to so you can know that at least your instance is secure/valid/working so you can relate to yourself as an authority. So an application based on levels of data assurance can validate your process through random of hashing through a high level of other peers (high level 80-90%).
    - This approach is actually scalable and the way that nature does it.
    - IRL We don't store global ledgers of every sentence being spoken and have to get consensus about their validity to store them. Instead we carry the processes for generating and understanding language. [check crypto for (maybe) a reference]
- Synchronization messages compressible by means of shared dictionaries/scapes
- Local identity - proxies as entanglement?

# The Sovereign Accountable Commons

[This is an abstract outlining some stuff we need to write up about the Sovereign Accountable Commons as a pattern enabled by Ceptr. I wrote this up to submit it for a book about Decentralizing the Commons via the P2P Foundation site.]

The Sovereign Accountable Commons (SAC) is akin to Decentralized Autonomous Organizations on the blockchain but they leverage a different distributed architecture, Ceptr. Ceptr is a new technology platform for building Distributed Apps and for enhancing collective intelligence in the process of being released by the MetaCurrency project in early 2017.

**Sovereign:**

An SAC runs as a collectively distributed application. As long as at least one person wants to run it, it can't be shut down. Two or more nodes running it make it a collective. It's algorithms set availability and redundancy based on size and scale of the network.

Everybody in a group is an equal peer, subject to the same rules, processes, and procedures, because everybody is running a copy of the same instruction set. This "code as law" applies to holding of assets, management of assets, interactions, transactions, governance, releasing new versions of "code as law," and every aspect of ongoing operation of the SAC.

Mutually Sovereign: Because each participant holds copies of their own assets, they can "take their toys and play elsewhere" (also known as forking), inviting others to bring theirs along as well. Yet since assets are typically stored in a DHT with redundancy, the group also maintains their copy. The group cannot threaten to withhold value which was generated by a participant in order to coerce them to continue to participate. Individuals cannot coerce others or the group, by threatening to take (digital) assets that they've shared.

**Accountable:**

SACs have membranes to membership. They can set whatever criteria they choose in allowing new members. This will often include validation of identity at whatever threshold the group requires.

Every contribution, communication, or action is signed. There is always a trail of evidence left by your actions.

You interact with the group, by interacting with your copy of the group's receptor (processes) which synchronizes, shares data, and processes interactions/transactions with other group members. All interactions in the group context are constrained to the encoded shared agreements of the group.

Ceptr is designed to support multiple inter-operating distributed reputation and other currencies via Intrinsic Data Integrity (data structures with cryptographic assurance to prevent alteration once stored). An SAC should be coded to measure the activities treasured by the group.

**Commons:**

All data and assets shared into the commons are structurally held in common. They are not (by default) held in a globally identical ledger, but rather in shards Distributed Hash Table (DHT) with algorithms to assure adequate redundancy and availability.

Authors (contributors, signors, or parties to a transaction) always keep a local copy of whatever they've put into the commons. The commons also keeps copies.

**Some Key Technical Differences from DAOs**

Ceptr enables distribution of reliable shared process. Therefore consensus and data verification are byproducts which serve to validate that a node’s processing has not become corrupted. Corruption is tracked and compromised nodes are blacklisted from further access.

Cells each carry instructions (in the form of DNA), not a record of the actions of every cell, which enables them to construct whatever future actions are needed. Language is carried by all language speakers as the processes to decode and encode the meaning of language, rather than a global ledger of every sentence everyone has spoken which needs some kind of crazy consensus to be committed into our brains.  Ceptr is structured the same as these patterns of collective intelligence found in nature.

Usage is highly scalable because it is constructed via a fractal localized architecture instead of having the bottleneck of a global ledger which requires consensus about interactions and data which simply should not require any consensus to be validly committed.

There's no wasted processing power from Proof of Work or undesirable Pareto Effects from Proof of Stake. For information that actually needs wider spread validation or consensus, it is done by high agreement threshold of randomized peers (via hash) making it virtual impossible to corrupt data. Even if you control almost the whole network, the original contributors will have their signed and valid copies which will expose the corruption and enable network immune response.

For the needs of most communities and commons, we believe Ceptr's approach will prove to be many orders of magnitude superior to blockchain consensus on global ledgers in terms of composability, adaptability, reliability, and storage and processing efficiency.

## CeptrNet and Routing

- "Strange loop-ness" (I exist in your processing space and you exist in my address space.) [Unclear the best order to explain these next subsections.]

Note from Arthur: I have a hunch that my first attempt to explain this will be a bit confusing because my thoughts are not even assembled on what is the best strategy… so this initial outline is just a first attempt to muddle through it. And I'm going to abbreviate to "CeptrNet" instead of writing "Ceptr Network" each time.

If you recall the strange loop section above, there's an unusual logical inversion in the way the network communication in Ceptr is implemented. Each VMHost (that wants to communicate with any other VMHosts via the CeptrNet (Maybe standalone devices don't have to do this?)) must get a CeptrNet address. You could say that each VMHost instance *exists* in the address space of CeptrNet.

Yet CeptrNet is itself implemented as a receptor. All the code for managing routing, communications, filtering, address and key management, etc. are implemented in a CeptrNet receptor which is installed in any VMHost that wants to communicate via the network.

[Diagram of Network of Physical Machines, VMHosts, CeptrNet receptors goes here.]

- "Protocol": Speaking across VMhosts via replication between sharded CeptrNet instances

There are two completely different types of communication within Ceptr: front-end protocols, and back-end protocols. Front-end protocols are all those things we think of as protocols for receptors to send messages to each other: like HTTP, SMTP, FTP, etc. Back-end protocols are how different instances of the same receptor talk to themselves -- for synchronization, verification, validation, mirroring, redundance, etc.

Front-end protocols are customizable, programmable ways to send communications in various formats. Back-end protocols are built into Ceptr for managing its internal functions.

Normally, network protocols are all thought of as these front-end things… and indeed they are when Ceptr interacts with people or programs via Internet protocols (like serving web pages via HTTP, or sending email via SMTP). But remember, within Ceptr, we don't have to speak differently structured protocols, receptors communicate via the self-describing Ceptr protocol. So every message is a Ceptr style message which just identifies the different protocol it's using semantically.

But when a receptor mirrors or synchronizes with instances of itself, it uses *special* internal maintenance protocols "within" the receptor, even though the data synchronization is happening over the network. This is exactly what the CeptrNet receptor does to send messages. It synchronizes, routes, and manages other instances of itself to have the message pop out of a CeptrNet receptor on the VMHost on the network to reach a local receptor there.

Ceptrnet addresses are huge UUIDs. They're in a big unique address space, that's a sparse space (there are gaps/holes in it). And the UUIDs are based on agency (who created the receptor, who is in charge, NOT physical topology of the network). So, your different streamscape instances are all a part of the "jarod holtz" agency and ID space, but they could physically be very remote from each other. Even though on one level they have an equivalent address (and can be considered delivered if it reaches any one), the physical topology is scattered around the world.

Efficient routing of non-topological addresses can be a very challenging problem.

Ceptr's back-end synchronization protocols track a lot of information about the physical topology and routing of the network and paths to "sibling" nodes. The processes for synchronization use non-sparse / tighter instance/sibling addresses, and retain routing information to peers. So, CeptrNet uses this "internal" peer management system to accomplish the delivery to "external" addresses.

Via the back-end "I'm talking to myself" protocol

(Bookmark explaining: the problem we're solving that others find difficult to solve.)

There's a protocol for inter-receptor communication on a single VMHost [link to message format?]. To communicate between receptors on different VMHosts there are a few additions to the local Ceptr protocol which require CeptrNet source and destination addresses and for messages to be signed (for authentication, validation, data-integrity purposes).

When a receptor sends a message to a remote receptor it actually sends the message to its local CeptrNet receptor to provide a destination address. The CeptrNet receptor handles the network communication.

- Triangulation for joining/registering

CeptrNet addresses are not managed by a central authority or even delegated to Internet Service Providers, they are structured as a UUID so that unique addresses can be generated with no central authority. However, you can't exactly create a new address all by yourself either. It actually takes two friends to "triangulate" you onto the CeptrNet.

Here's how it works. You, Carl, would like to create an identity/address that can use CeptrNet. You have your friend Alice generate a new UUID address for you via her CeptrNet receptor, and your friend Bob, register your address with its associated public signing key, authorized identity service, and key revocation process into his CeptrNet receptor. This begins your ability to start to communicate via CeptrNet.

- Routing strategies and algorithms for non-topological UUID space

Probabilistic algorithm… Not a guaranteed route, but a high-probability method for discovering routes.

Examples of what info is kept and tracked about physical network topology, bandwidth, throughput, reliability, latency, etc.

## Optimized for Evolution

In the late 1980s, it was already clear that we were going to run out of IPv4 addresses (Internet Protocol version 4). Work began in earnest in 1992 on designing the next version of the Internet Protocol. There was brief experimentation with IPv5, then in 1996 IPv6 was released (which solves many known issues with IPv4). Today, in 2015, we've known about the need for change for almost 30 years, and published the standard to change to almost 20 years ago, but IPv4 is still the basis of over 96% of Internet traffic.

Even though network nodes on IPv4 and IPv6 cannot talk directly to each other, requiring gateways and infrastructure providers to route and bridge both protocols, it looks like they will operate alongside each other for the foreseeable future. Based on the current trajectory, it seems likely that it will have taken us 40 years to upgrade the Internet protocol.

Given the pace of change happening today, and how quickly our needs are evolving, we suggest that it is a big problem to take four decades to move to a new protocol even when everyone agrees it is needed. This kind of problem exists not only for communication protocols, but also for operating systems, software, libraries, plugins, and modules which can create a rat's nest of inconsistencies and compatibility issues.

In Ceptr, we have built versioning and deprecation right into the low-level design of the architecture.

As we've mentioned, Ceptr data is structured as semantic trees. The first branching under the root of a receptor in the Compository is its version. So, when you reference a receptor by its Compository ID, by default you would receive the latest version. However, you can also reference a specific version by a sub-address on that ID. Semantic markers are used to note when versions maintain compatibility with prior protocols and structures or when backward compatibility is broken.

It is also possible for the author of a composition to mark the whole thing (all versions) as deprecated with a forwarding address to a receptor which replaces it. This is important when the next improvement is not simply the new version of the same receptor, but the receptor has been forked to a new composition address or its functions are better performed by another composition altogether.

We see this low-level integration of deprecation and versioning as vital to the evolution of Ceptr itself since we're not assuming we're going to get it all right the first time. We're tackling far too many things at once to be experts at all of it. So, as domain experts start to take interest in Ceptr and contribute better solutions, we expect to update many things (like scape indexing, key management, certificate management, routing algorithms, synchronization methods, GPU processing, and many performance optimizations).

Another example of the importance of this is how it will be used by the Ceptr Network itself. As the network grows, the sharding/routing/distribution algorithms will need to change for better load

balancing and to keep each VMhost's share of processing to manageable and sensible levels. We can publish a new version of the CeptrNet receptor, with a scheduled changeover date. All connected nodes will be notified of the changeover and have an opportunity to install the new protocols before it goes live. Even if some nodes were offline for a long time and tried to reconnect, other nodes would identify the version of the network protocol they were speaking and send the new protocol upgrade to them using the old version of the protocol (which is still available to them).

Did you know that Wikipedia uses open source tools and shares its databases publicly so that anyone could come up with a way of doing what they do better and would have the means to replace them? In similar manner, we've built in the capacity for competing network protocols to operate in parallel with Ceptr or even replace it altogether if people opted over to that network. This too, is a part of optimizing for evolution.

## Trust-Based Agency

Still to write/expand:

- No centralized address authority. Self-Generated UUIDs validated via triangulation process of trusted peers "lending" their credibility.
- VMhost UUIDs could also connect to CPUid /MAC addr? And also show ID of agent who spun them up.
- Redefining Freedom. Decreasing degrees of freedom to increase degrees agency.
- Eleutheria: meaning freedom from the word roots: arriving (eleu) to where one loves (eran)

## A Strong Immune System

Still to write/expand:

- Persistence of identity. Connecting identity to agency. Fingerprinting on agency.
- Reputation. Erosion of reputation. "SPAM" filtering at network level.

## No Global Top / Contextual Namespaces

Still to write/expand:

- All receptors exist inside the context (existence scape) of another receptor.
- The Ceptr network itself is an address among possible networks/// So that someone can use our tools to bootstrap a better one // Wikipedia example
- No absolute namespace. Contextual namespaces, just like in real life. How to resolve contexts. Proxy mapping to UUIDs. Name resolution.

## Hyper Reusability: Sharable Models and World Views

Still to write/expand:

- Ceptr Compository (more contextual naming, UUIDs and trusted triangulation), Semantic namespace resolution for shared protocols, enumerations, receptors, models and world-views. (B.C. computing. C.E. computing)

- Semantic Code lets us render a narrative about the intentions of the code. A Semantic Stack, lets us render a narrative of its running state (if paused for debugging, for example).
- Authentication of trusted components. Authorization of authoritative sources or processes for performing commits and version updates. “Signing” new components into compository via code-review.

# Appendices

## Some other Projects Working on Parts of the Problem

Organize this section by the problem domains we said we're solving:

Organize the following into the categories above. Add MANY more.

**Some sources to collect things which should be added to this list:**

- redecentralize.org - https://github.com/redecentralize/alternative-internet (scroll down for list)
- P2Pfoundation.net - http://p2pfoundation.net/Category:Resources (scroll down)
- Distributed social network tools: http://en.wikipedia.org/wiki/Comparison_of_software_and_protocols_for_distributed_social_networking
- Cryptocurrencies: http://en.wikipedia.org/wiki/Cryptocurrency#List_of_crypto_currencies
- https://web.archive.org/web/20080410193834/http://www.dp2p.net/
- Torrent and Filesharing (e.g. dropbox & google drive) tools
- See IIW and personal data ecosystem lists http://pde.cc/directory/ http://www.capterra.com/identity-management-software
- From Wikipedia @ identity management: Technologies, services and terms related to identity management include Directory services, Service Providers, Identity Providers, Web Services, Access control, Digital Identities, Password Managers, Single Sign-on, Security Tokens, Security Token Services (STS),Workflows, OpenID, WS-Security, WS-Trust, SAML 2.0, OAuth and RBAC.[5]
- SAML, OAuth, OpenId, DataPortability, etc.
- UpRight: https://code.google.com/p/upright/wiki/UpRightFAQ

### Currencies

- Blockchain Cryptocurrencies: Bitcoin, DogeCoin
- Blockchain non-currency uses Namecoin, etc.
- Blockchain Computing: Ethereum

### P2P Solutions

- Diaspora
- MaidSafe ?
- Synereo
- IPFS (The Permanent Web) http://ipfs.io
- http://www.sharksystem.net ?
- http://indiewebcamp.com/
- DAOWISP http://www.daowisp.io

### Communication Protocols

- IPv4, IPSec, IPv6, TCP

- ACL http://en.wikipedia.org/wiki/Agent_Communications_Language
- RINA https://en.wikipedia.org/wiki/Recursive_InterNetwork_Architecture_%28RINA%29 (I'm not sure whether we can use any of the C code they've built, but they're operating on similar networking principles. ProtoRINA on github. IRATI RINA Linux implementation on github )

### Semantics & Shared Meaning Making

- Semantic Web
- Hypothes.is

### Identity

- XRI/XDI, iNames
- Leola Group & Nymbles (an XDI implementation)
- OpenID
- Higgins
- Liberty Alliance… blah blah blah
- connect.me
-
- OpenPDS
- Open Notice: Consent Receipts
- Certifications / Trustmarks
- Computational Law (Dazza in Sandy's MIT Human Dynamics Lab / socialphysics.org)
- Law is Code / Code is Law
- ID3 / Open Mustard Seed
- DemocracyOS
- Bribe-Hackers
- OpenID
- OpenID Connect
- Blockchain ID

## References / End Notes