DAG Architecture

Deep Breakdown

A comprehensive description of the Directed Acyclic Graph architecture as envisioned for the Extropy Engine protocol — where each person is a DAG, points are events or convergences with other DAGs, and the structure enables retroactive updateability and convergence-based minting.

Extropy Engine Protocol Layer

Revision: May 2026, aligned with Extropy Engine v3.1.2 (R=Rarity, F=Frequency-of-decay correction)

Co-written and curated by Randall Gossett

> “Each person should be a DAG and points are events or convergences with other DAGs and that’s part of the minting process and tracking and what makes it retroactively updatable.”
>
> — Creator’s vision statement

Contents

1. The Core Vision: Person as DAG

The foundational insight of the Extropy Engine DAG architecture is that identity is not a static record—it is a living, growing graph of causally-ordered events. Every person in the system is represented not by a row in a database, but by an entire Directed Acyclic Graph. Their DAG is their history, their reputation, their contributions, and their relationships—all encoded as cryptographically signed, immutable vertices linked by causal edges.

This is a radical departure from traditional user-profile architectures. In a conventional system, a user has a profile with fields: name, score, badges. In Extropy, a user IS their DAG. The "score" is not stored—it is computed by walking the graph. The "reputation" is not a number in a column—it is the accumulated confirmation weight of all the vertices that person has produced and had validated by others.

Three Axioms of Person-as-DAG

- Axiom 1: Events, not state. The DAG stores events (a paper was uploaded, a review was submitted, a loop closed) rather than current state. State is derived by replaying the event history. This enables retroactive recomputation when new information arrives.
- Axiom 2: Convergence is value. When two person-DAGs reference the same vertex—a co-authored paper, a peer review, a governance vote—that creates a convergence vertex. These convergences are where minting happens, because they represent verified multi-party entropy reduction.
- Axiom 3: Graphs nest fractally. A person-DAG can participate in a DFAO-DAG (Decentralized Fractal Autonomous Organization), which itself nests inside a larger DFAO-DAG. The same vertex structure and causal ordering applies at every scale—from an individual paper review to civilization-wide governance.

2. Existing Implementation: dag-substrate

The dag-substrate service (port 4008) is the current implementation of this vision. It runs as a standalone microservice within the Extropy Engine docker-compose stack, backed by PostgreSQL, and is inspired by the IOTA Tangle's permissionless DAG ledger design.

Architecture Overview

The service is 1,573 lines of TypeScript implementing a full DAG ledger with these core components:

Component	Purpose
Vertex Store	PostgreSQL table with content hashing, Lamport timestamps, signature fields, confirmation weights
Tip Selection	Weighted random walk algorithm (IOTA-inspired) that selects 2 parent tips for each new vertex
Confirmation Propagation	Recursive weight propagation up to depth 20, where each new vertex increases the confirmation weight of its ancestors
Auto-Vertex Handlers	Redis pub/sub listeners that automatically create vertices for every system event (loop open, XP mint, governance vote, etc.)
Content Hashing	SHA-256 hash of vertex payload + parent IDs + Lamport timestamp, forming the content-addressable identity of each vertex
Signature Verification	Ed25519 signature verification on vertex creation (secp256k1 stub present for future Ethereum compatibility)
REST API	POST /vertices, GET /vertices/tips, GET /vertices/by-entity/:entityId, GET /vertices/causal-history/:vertexId

Event Bus Integration

The dag-substrate listens to every Redis pub/sub channel in the system. When any service emits an event—a loop opening, an XP mint, a governance vote—the dag-substrate automatically creates a corresponding vertex. This means the DAG is a complete, causally-ordered log of everything that has ever happened in the Extropy Engine. Nothing escapes the graph.

3. Vertex Types and the Event Ontology

Every vertex in the DAG has a type that classifies what kind of event it represents. The VertexType enum defines the complete event ontology—the vocabulary of actions the system can record.

Vertex Type	Event Meaning	Triggered By
`LOOP_OPEN`	A new causal loop has been opened—a claim has been submitted and decomposed into verifiable sub-claims	Epistemology Engine
`LOOP_CLOSE`	A loop has closed with verified entropy reduction. This is the fundamental value event.	Loop Ledger
`MEASUREMENT`	An entropy measurement was recorded (before or after a claimed action)	Loop Ledger
`XP_MINT`	Experience points were minted. Provisional minting, final confirmation, and burn each create a vertex.	XP Mint Service
`GOVERNANCE_PROPOSAL`	A governance proposal was created, passed, rejected, or implemented	Governance Service
`GOVERNANCE_VOTE`	A vote was cast on a proposal, weighted by the voter's reputation	Governance Service
`DFAO_CREATE`	A new Decentralized Fractal Autonomous Organization was formed	DFAO Registry
`DFAO_MEMBERSHIP`	A person joined or was admitted to a DFAO	DFAO Registry
`TOKEN_MINT`	A token (CT, DT, IT, EP) was minted	Token Economy
`TOKEN_BURN`	A token was burned (inactivity, penalty, or conversion)	Token Economy
`CREDENTIAL_ISSUE`	A badge, title, level, or CAT certification was issued	Credentials Service
`SEASON_START`	A new competitive season began	Temporal Service
`SEASON_END`	A season ended with final rankings computed	Temporal Service
`GENERIC`	Catch-all for events that don't map to a specific type (reputation accrual, signal routing, etc.)	Various

In the person-as-DAG model, each vertex in this table is also an event in someone's personal graph. A LOOP_CLOSE vertex in the system DAG is simultaneously a vertex in the person-DAG of every validator who participated in that loop.

4. IOTA Tangle Mechanics in Extropy

The dag-substrate adopts the IOTA Tangle's core insight: in a DAG ledger, every new transaction validates two previous transactions. There is no mining, no block producer, no leader election. Validation is the cost of participation.

Lamport Timestamps

Each vertex carries a Lamport timestamp—a logical clock that enforces causal ordering without requiring synchronized wall clocks. When a new vertex is created, its Lamport timestamp is set to max(parent_timestamps) + 1. This guarantees that if vertex A causally precedes vertex B, then A.lamport < B.lamport. Two vertices with no causal relationship may have arbitrary relative timestamps—they are concurrent events in different parts of the DAG.

Weighted Random Walk Tip Selection

When a new vertex needs to choose which existing tips (unconfirmed vertices) to reference as parents, the dag-substrate uses a weighted random walk algorithm:

1. Collect all current tips (vertices with isTip = true).
2. If fewer than 2 tips exist, pad with the genesis vertex or available tips.
3. For each candidate tip, compute a weight based on its confirmation weight and age (older tips get lower weight to prevent stale references).
4. Perform a biased random walk: start from a random tip, walk backwards through parents with probability proportional to confirmation weight, then select the 2 tips reached at the end of the walk.
5. The selected tips become the parentVertexIds of the new vertex.

This mechanism is critical to the person-as-DAG vision because it means every action a person takes (creating a vertex) simultaneously validates the work of others (by referencing their tip vertices). Participation IS validation.

Confirmation Weight Propagation

When a new vertex V references parents P1 and P2, the confirmation weight of P1 and P2 increases by 1. But propagation doesn't stop there—the dag-substrate walks up to 20 levels of ancestors and increments their confirmation weights too. This creates a natural finality gradient: the deeper a vertex is buried under subsequent vertices, the higher its confirmation weight, and the more “final” it is considered.

In the contracts, a DAGSubstrateConfig.confirmationThreshold defines how much weight a vertex needs before it is considered “finalized.” This is analogous to block confirmations in a blockchain, but operates continuously rather than in discrete blocks.

5. Person-DAG Topology

"Each person should be a DAG." This is the central architectural statement. Here is what it means concretely:

What a Person-DAG Contains

A person-DAG is the filtered subgraph of the global DAG containing all vertices where that person is either the creator (signer) or a referenced participant. It includes:

- Genesis vertex: Created when the person first authenticates (Google OAuth to DID binding). This is the root of their personal DAG.
- Loop participation vertices: Every LOOP_OPEN they submitted, every MEASUREMENT they recorded, every consensus vote they cast.
- XP mint vertices: Every XP_MINT event where they were a recipient—the immutable record of value earned.
- Credential vertices: Badges, titles, levels, CAT certifications—each is a CREDENTIAL_ISSUE vertex in their DAG.
- Governance vertices: Proposals submitted, votes cast—their democratic participation history.
- DFAO membership vertices: Which organizations they joined and when.
- Convergence vertices: The critical category—vertices shared with other person-DAGs, representing collaborative events.

Querying a Person-DAG

The existing REST API already supports this view. The endpoint GET /vertices/by-entity/:entityId returns all vertices associated with a given entity (person, loop, DFAO). When the entityId is a person's ValidatorId, the result is their person-DAG—the complete causal history of everything they've done in the system.

The endpoint GET /vertices/causal-history/:vertexId walks backwards from any vertex, returning all ancestors up to a configurable depth. This allows you to answer questions like: "What was the chain of events that led to this XP mint?"—tracing causality from a reward back to the original claim, through measurements, validations, and consensus.

Visual Shape of a Person-DAG

Imagine a river delta viewed from above. The genesis vertex is the source. Each branch represents a thread of activity—one branch for a paper uploaded, another for a governance proposal, another for peer reviews. Where branches rejoin (convergence vertices), the person's DAG intersects with others. The width of the delta represents their breadth of participation; the depth represents their history; the convergence density represents their collaborative impact.

6. Convergence Vertices: Where DAGs Meet

"Points are events or convergences with other DAGs." This is where the architecture becomes truly novel. A convergence vertex is a vertex that appears in two or more person-DAGs simultaneously.

Types of Convergence

Convergence Type	Mechanism	Example
Peer Review	Person A submits a paper (LOOP_OPEN). Person B reviews it (MEASUREMENT + consensus vote). The LOOP_CLOSE vertex references both A's submission and B's review.	Alice uploads a manuscript; Bob's AI assessment and Carol's peer review create convergence vertices linking all three DAGs.
Co-Authorship	Multiple persons are listed as contributors on a single claim. The LOOP_OPEN vertex references all contributor IDs.	A multi-author paper creates a convergence vertex in every co-author's DAG simultaneously.
Governance	A proposal vertex is in the proposer's DAG. Each GOVERNANCE_VOTE vertex creates a convergence between the voter's DAG and the proposer's DAG.	A DFAO parameter change voted on by 12 members creates 12 convergence edges.
DFAO Formation	A DFAO_CREATE vertex lists founder IDs. Each DFAO_MEMBERSHIP vertex links the joiner's DAG to the DFAO's collective DAG.	Three researchers form an AcademicXP review collective—their personal DAGs converge at the DFAO creation vertex.
Token Transfer	TOKEN_MINT and TOKEN_BURN vertices reference the recipient/source. Cross-domain XP exchanges create convergence between the exchange parties.	XP earned in a code loop is converted to CT, creating vertices in both the minter's and the token economy's DAGs.

Why Convergence Matters

Convergence vertices are the atomic unit of social proof. A person-DAG with many convergence vertices is a person deeply embedded in collaborative knowledge creation. A person-DAG with few convergences is isolated—potentially a sybil or a free-rider. The convergence density of a person-DAG becomes a natural, unforgeable metric of genuine participation.

In the minting formula, convergence is captured through Frequency-of-decay (F) and the Rarity coefficient (R). Both are derived from DAG topology, but in different ways. F is a multiplicative dampener computed from the submitter's recent history of same-shape claims, fingerprinted by (submitter_id, claim_type, primary_domain, DFAO_id). R is the action-class scarcity of the loop being closed: a property of the loop, not the actor. Reputation never enters XP. Reputation density (ρ) lives only in CT, where validators apply weight to claims they confirm.

7. Retroactive Updateability

"That's part of what makes it retroactively updatable." This is one of the most powerful properties of the DAG architecture, and it distinguishes Extropy from blockchain-based systems where history is frozen once confirmed.

How Retroactive Updates Work

The DAG never mutates existing vertices—they are cryptographically sealed by their content hash (SHA-256 of payload + parent IDs + Lamport timestamp). Instead, retroactive updating works by appending new vertices that reinterpret existing ones:

- Recomputation vertices: When new evidence arrives that changes the Bayesian posterior of a claim, a new MEASUREMENT vertex is appended referencing the original LOOP_CLOSE. The XP formula is recomputed with updated inputs, and a new XP_MINT vertex (with positive or negative delta) adjusts the balance.
- Confirmation weight shifts: As new vertices accumulate and reference different ancestors, the confirmation weights of old vertices naturally shift. A once-confirmed vertex that stops receiving descendant references sees its relative weight decrease—a form of social consensus erosion.
- Credential revision: If a paper is later found to be flawed, a new LOOP_OPEN targeting the original claim creates a counter-loop. If the counter-loop closes with verified evidence of the flaw, the original author's reputation density (ρ) is penalized at the CT layer, but the original XP vertices remain in the DAG as permanent history.
- Reputation density (ρ) decay: The Temporal service periodically emits decay events. These become GENERIC vertices in person-DAGs, retroactively adjusting the effective ρ weight of older contributions in CT computations, without altering the underlying vertices themselves.

Event Sourcing, not State Mutation

The DAG is fundamentally an event-sourced system. The "current state" of a person's reputation density (ρ), XP balance, credential set, or governance weight is always computed by replaying their DAG from genesis to present. This means any new vertex that adds, corrects, or reinterprets past events automatically changes the computed state, without touching the immutable event log.

This is what makes the architecture retroactively updatable: the events are permanent, but their interpretation is always current. A paper reviewed in 2024 might receive a citation convergence in 2027 that retroactively increases the original reviewer's reputation density (ρ) at the CT layer, because the new convergence vertex adds confirmation weight to the original review vertex. The original XP mint stays fixed.

8. Minting at Convergence Points

"That's part of the minting process." In the Extropy Engine, value is never created in isolation. Minting happens at convergence points—where multiple person-DAGs intersect through verified collaborative events.

The Minting Invariant

The core invariant from the contracts is: XP is minted if and only if a Loop closes with verified entropy reduction greater than zero (verified delta-S > 0). No loop closure means no value, means no mint. This is enforced at the type level in the contracts.

Convergence in the XP Formula

The full XP formula is: $XP = R \times F \times \Delta S \times (w \cdot E) \times \log(1/T_s)$. Each factor maps to DAG topology:

Factor	Symbol	DAG Interpretation
Rarity	R	Action-class scarcity of the loop being closed. Property of the loop, not the actor. Math is invariant under actor swap. R is read from the action-class registry at LOOP_CLOSE time, never accumulated from a validator's history.
Frequency-of-decay	F	Multiplicative dampener on repeated same-shape claims. Computed from the submitter's recent history fingerprinted by (submitter_id, claim_type, primary_domain, DFAO_id). Bounded (0,1]. Anti-grind, anti-spam, anti-Goodhart. NOT falsifiability and NOT consensus density.
Entropy Reduction	ΔS	The measured before/after difference. Stored as MEASUREMENT vertices in the DAG.
Domain Weight	w	Governance-adjustable. Changing w creates a GOVERNANCE_PROPOSAL + GOVERNANCE_VOTE convergence pattern.
Essentiality	E	How critical the task was. Higher-essentiality loops create heavier convergence vertices.
Settlement Time	T_s	Time between LOOP_OPEN and LOOP_CLOSE vertices. Faster closure = higher XP. The Lamport gap between these vertices measures causal distance.

The Irreducible Form

The contracts also define an irreducible form: $XP = \Delta S / c_L^2$, where c_L is the causal closure speed for the domain. This is the physics floor—the minimum XP that must be minted for a given entropy reduction. It ensures that no amount of gaming the convergence topology can mint XP without actual entropy reduction.

9. DFAO Nesting: Fractal DAG Hierarchies

"Shouldn't it use a DAG architecture? Nest by convergences with other DAGs." This vision extends naturally through the DFAO system.

What is a DFAO?

A Decentralized Fractal Autonomous Organization (DFAO) is a self-governing collective that exists at a specific scale—from nano (individual project) to planetary (civilization-wide). Each DFAO is itself a DAG, and it nests inside larger DFAOs through the parentDFAOId / childDFAOIds tree structure defined in the contracts.

Scale	Scope	Example in AcademicXP
Nano	Individual project	A single paper review circle
Micro	Team / small group	A research lab's review collective
Meso	Department / community	A university department's AcademicXP instance
Macro	Organization / city	A multi-university research consortium
Planetary	Civilization-wide	Global open-science governance

DAG-in-DAG Nesting

Each DFAO has its own sub-DAG—the filtered view of the global DAG containing vertices tagged with that DFAO's ID (via the optional dfaoId field on DAGVertex). When a person acts within a DFAO, their vertex appears in both their person-DAG and the DFAO's collective DAG. This is the literal implementation of "nest by convergences":

- A person-DAG is a subgraph filtered by signer public key.
- A DFAO-DAG is a subgraph filtered by dfaoId.
- A convergence vertex appears in both subgraphs.
- A parent DFAO's DAG is the union of all child DFAO DAGs plus its own top-level vertices.

The fractal nesting means governance at each scale operates on its own sub-DAG while inheriting confirmation weight from the parent. A proposal that passes in a micro-DFAO adds confirmation weight that propagates up to the meso-DFAO's DAG, and so on.

DFAO Lifecycle as DAG Events

The progression from SHADOW (non-binding governance) through HYBRID to ACTIVE (full autonomy) is itself recorded in the DAG through DFAO_STATUS_CHANGED events. Each phase transition is a convergence vertex—it required governance votes (multiple person-DAGs converging) to approve the transition.

10. Identity Layer: DID-to-DAG Binding

For the person-as-DAG model to work, each person needs a cryptographic identity that binds their Google OAuth login to their position in the DAG. The envisioned 3-layer identity architecture:

Layer	Technology	Purpose
1. Authentication	Google OAuth 2.0 (credentials service, port 4013)	Human-friendly sign-in. Produces a session token.
2. Self-Sovereign ID	DID method did:extropy:<dag-hash>	Maps the OAuth identity to an Ed25519 keypair. The DID document is the genesis vertex of the person-DAG.
3. Privacy	ZK proofs (future)	Selective disclosure: prove you have a credential without revealing your identity. Enables anonymous peer review.

The DID Method

The envisioned DID method is did:extropy:<dag-substrate-hash>, where the hash is the content hash of the person's genesis vertex in the DAG. This creates a self-referential binding: the DID points to the DAG, and the DAG's first vertex contains the DID. The Ed25519 public key embedded in the genesis vertex is the same key used to sign all subsequent vertices in that person's DAG.

Signature verification in the dag-substrate already supports Ed25519 (with a secp256k1 stub for future Ethereum/wallet compatibility). When a vertex is submitted, the service verifies that the signature matches the public key, and that the public key maps to a known entity.

Why DID Matters for Person-DAGs

Without cryptographic identity, a person-DAG is just a filtered database query. With DID binding, a person-DAG becomes a self-sovereign, portable, verifiable record. A researcher could take their person-DAG from one AcademicXP instance to another, and the receiving system could verify every vertex's signature against the originating DAG's DID document.

11. The AcademicXP Vertical: DAG in Practice

AcademicXP is the first frontend vertical built on top of the Extropy Engine DAG substrate. In the Extropy ontology, a manuscript IS a claim—it belongs to the informational entropy domain. Here is how the DAG architecture manifests in the academic peer review workflow:

Paper Upload as Loop Opening

When a researcher uploads a paper, the system creates a Claim in the informational domain. This triggers a LOOP_OPEN vertex in the DAG. The vertex payload contains the paper metadata, the submitter's ValidatorId, and a content hash of the manuscript. This vertex becomes a new node in the researcher's person-DAG.

AI Assessment as Measurement

The LLM assessment of the paper produces a MEASUREMENT vertex. The measurement source type is 'algorithm' (as defined in the MeasurementSource interface). The entropy value represents the information-theoretic novelty of the paper—how much it reduces uncertainty in its field. This vertex references the LOOP_OPEN vertex as a parent, establishing causal ordering.

Peer Review as Convergence

When another researcher reviews the paper, they create MEASUREMENT and consensus vote vertices. These are convergence vertices, they appear in both the author's person-DAG and the reviewer's person-DAG. The review action validates the author's claim while simultaneously building the reviewer's own reputation density (ρ) graph for use in CT calculations.

Loop Closure and XP Minting

If the weighted consensus (V+ = sum of approving reviewers' reputation density ρ, V- = sum of denying reviewers' ρ) favors approval and the measured entropy reduction is positive, the loop closes. A LOOP_CLOSE vertex is created, followed by XP_MINT vertices distributing XP to both the author and all reviewers. The XP amount is computed by the full formula $XP = R \times F \times \Delta S \times (w \cdot E) \times \log(1/T_s)$. R is the action-class rarity of this loop. F is the submitter's frequency-of-decay penalty. Reputation density ρ weighted the consensus, but never enters the XP mint itself.

Citation as Retroactive Convergence

Later, when another paper cites the original, the new paper's LOOP_OPEN can reference the original's LOOP_CLOSE vertex. This creates a retroactive convergence—a new edge in the global DAG connecting two previously independent research threads. The confirmation weight of the original paper's vertices increases, potentially triggering additional XP minting for the original author and reviewers.

12. Causal History and Walk Algorithms

The dag-substrate implements a causal history walker that can traverse backwards from any vertex, reconstructing the chain of events that led to it. This is the foundation for both auditing and retroactive computation.

Causal History Walk

The GET /vertices/causal-history/:vertexId endpoint performs a breadth-first walk backwards through parent vertices, up to a configurable depth. For each vertex visited, it returns the vertex type, payload, Lamport timestamp, and confirmation weight. This produces a tree (technically a DAG subgraph) showing everything that causally contributed to the queried vertex.

Use Cases for Causal History

- Audit trail: Given an XP_MINT vertex, walk backwards to find the LOOP_CLOSE, the MEASUREMENTs, the LOOP_OPEN, and the original claim. Every minted token is traceable to its root cause.
- Reputation density (ρ) computation (CT-side only): A validator's current ρ is computed by walking their person-DAG and summing weighted contributions from all loops they participated in, with temporal decay applied by Lamport distance. ρ feeds CT = C × F × ρ × Δ × E only. It never feeds XP.
- Convergence analysis: Walk from a convergence vertex outward to discover which person-DAGs are connected through it. This reveals collaboration clusters and can identify sybil patterns (multiple “person-DAGs” that never converge with independent ones).
- Impact tracing: Walk forward (downstream) from a LOOP_CLOSE to see which subsequent vertices referenced it—revealing the citation and influence network of a piece of work.

Content Hashing and Integrity

Every vertex's content hash is computed as SHA-256 of a canonical representation of: the vertex type, the full payload, the parent vertex IDs (sorted), and the Lamport timestamp. This means any tampering with a vertex's content would invalidate its hash, and since child vertices reference the parent by ID (which incorporates the hash), the entire downstream subgraph would become inconsistent. This provides Merkle-tree-like integrity guarantees across the entire DAG.

13. Data Model Reference

DAGVertex Interface

The canonical vertex type from @extropy/contracts:

Field	Type	Description
`id`	`VertexId`	Branded string ID, globally unique
`vertexType`	`VertexType`	One of 15 event types (see Section 3)
`signature`	`string`	Cryptographic signature of content
`publicKey`	`string`	Signer's public key (Ed25519)
`algorithm`	`CryptoAlgorithm`	'ed25519' or 'secp256k1'
`lamportTimestamp`	`number`	Causal ordering clock
`wallTimestamp`	`Timestamp`	ISO-8601, secondary ordering
`parentVertexIds`	`VertexId[]`	Parents this vertex validates (min 2 for tip selection)
`contentHash`	`string`	SHA-256 of canonical content
`confirmationWeight`	`number`	Cumulative weight from descendants
`isTip`	`boolean`	True if no vertex references this one yet
`payload`	`VertexPayload`	Event-specific data (loop, mint, vote, etc.)
`dfaoId`	`DFAOId?`	Optional DFAO context for nesting
`propagation`	`VertexPropagation`	Origin node, hop count, validation status

DAGSubstrateConfig

Parameter	Default / Type	Meaning
`minParentCount`	2	Each new vertex must reference at least 2 tips
`maxTipAge`	Lamport ticks	Tips older than this are excluded from selection

Parameter	Default / Type	Meaning
`confirmationThreshold`	number	Weight needed for a vertex to be considered finalized
`tipSelectionAlgorithm`	weighted_random_walk	IOTA-inspired biased walk toward heavier branches
`walkDepth`	number	How many hops the random walk takes before selecting

Token Types in the DAG

Six token types flow through the DAG, each with different properties:

Token	Name	Transferable	DAG Role
`XP`	Experience Points	No	Minted at loop closure. Non-transferable. Decays at rate rho per 30 cycles.
`CT`	Contribution Token	Restricted	Cross-platform. Formula: CT = C × F × ρ × Δ × E (Capability × Frequency-of-decay × reputation density × entropy delta × Eight-domain weighting). F is the same frequency-of-decay penalty used in the canonical XP formula. Has lockup period.
`CAT`	Capability Token	Portable	Skill certification. Log-scale levels (10/30/90/270 validations).
`IT`	Influence Token	No	Governance weight. Decays monthly.
`DT`	Domain Token	Limited	Subject-matter expertise marker.
`EP`	Emergence Points	Local	Merchant loyalty. EP = XP x L (local multiplier).

14. From Current State to Full Vision

The dag-substrate already implements the core DAG ledger mechanics—vertex creation, tip selection, confirmation weight propagation, content hashing, signature verification, and causal history walking. The path from current implementation to the full person-as-DAG vision involves three phases:

Phase 1: Entity-Indexed Person Views

The by-entity endpoint already exists. The first step is to make it the primary way of querying user data: a person's profile page is their person-DAG rendered as a timeline. Reputation, XP, credentials, and convergence count are all computed from the graph on read.

- Build person-DAG view API: filtered subgraph for a ValidatorId
- Compute reputation from DAG walk instead of a stored column
- Render convergence density as a first-class metric on profiles
- Visualize person-DAG as an interactive graph (D3.js or similar)

Phase 2: DID Binding and Portable DAGs

Introduce the did:extropy DID method. Generate Ed25519 keypairs on first login, bind them to Google OAuth sessions, and create genesis vertices. All subsequent vertices are signed with the person's key, making the person-DAG cryptographically verifiable and portable.

- Implement did:extropy DID method and resolver
- Generate and store Ed25519 keypairs in the credentials service
- Create genesis vertex on first authentication
- Sign all subsequent vertices with the person's private key
- Build DID document from genesis vertex for external verification

Phase 3: Full Convergence-Based Minting

Rework the XP minting pipeline to be fully DAG-native. Instead of minting from service events, mint from convergence vertex patterns detected in the DAG itself. The minting trigger becomes: a convergence vertex pattern matching a closed loop with sufficient confirmation weight.

- Implement convergence vertex detection (multi-signer vertex patterns)
- Trigger minting from DAG topology analysis, not just event bus
- Enable retroactive minting when new convergences reference old loops
- Build ZK proof layer for anonymous convergence verification
- Implement cross-instance DAG federation for portable person-DAGs

Source: Extropy Engine codebase (packages/dag-substrate/src/index.ts, packages/contracts/src/types.ts). https://github.com/00ranman/extropy-engine