

Engineering a Digital Ownership Protocol

From Practical Ownership Problems to Verifiable Modular Infrastructure

Wayne Glass
Auctoritas Systems
September 2026

Working Paper

Abstract

This paper describes constitutional engineering: a methodology of discovering the relationships that must remain fixed, encoding them as a small immutable core, and allowing operational systems to compose freely upon it — observed through the development of a working protocol, and in part enforced by running verification.

This paper describes a working, source-verified implementation. The specification was extracted from the code, not the code derived from it.

Note on method: the observations in this paper are grounded in an implementation whose structure was independently verified from source. The specification was extracted from the code; the code was not generated from the specification.

1 Introduction

DOP-P100 introduced a constitutional model for digital ownership in which ownership, evidence, provenance and delegated authority are treated as enduring architectural principles.

This paper examines a different question.

How can such principles be translated into an operational protocol without allowing implementation technology to redefine the underlying constitutional model?

The Digital Ownership Protocol did not begin as a theoretical research project.

It began with a practical objective: enabling genuine ownership within a digital games marketplace.

Observed implementation behaviour repeatedly demonstrated that solving this practical problem required progressively more fundamental architectural abstractions.

Marketplace behaviour became constitutional ownership.

Trust became evidence.

Historical records became provenance.

Commerce became a specialised engine.

These observations gradually produced an engineering methodology in which implementation follows constitutional principles rather than defining them.

This paper documents that methodology and the architectural evolution that emerged during implementation.

The broader implications of applying this methodology to digital infrastructure are considered in DOP-P300.

2 Engineering Problem

Many software architectures begin by modelling applications.

The implementation subsequently evolves to satisfy those application requirements.

While effective for many systems, this approach may gradually couple architectural decisions to the first successful implementation.

Observed implementation behaviour during development of DOP suggested a different possibility.

Rather than expanding the original marketplace architecture, development repeatedly identified more fundamental abstractions capable of supporting increasingly diverse applications.

This produced an engineering process centred upon discovering constitutional relationships before expanding operational capability.

The resulting methodology differs from traditional implementation-first development by continually reducing architectural complexity before allowing the surrounding ecosystem to grow.

3 Background

Engineering disciplines have traditionally developed increasingly sophisticated methods for managing software complexity.

Layered architectures.

Object-oriented design.

Service-oriented architectures.

Microservices.

Domain-driven design.

Each approach addresses different aspects of scale, maintainability and evolution.

Observed implementation behaviour during development of the Digital Ownership Protocol suggested that an additional perspective may be valuable.

Rather than organising software around applications or technologies, architectural decisions repeatedly converged towards constitutional relationships that remained stable despite continued implementation change.

This observation became the foundation of the engineering methodology described throughout this paper.

4 Proposed Engineering Methodology

Observed implementation behaviour consistently suggested that architectural complexity was reduced by identifying more fundamental abstractions rather than expanding existing implementations.

This paper proposes an engineering methodology founded upon that observation.

The process begins with a practical problem.

Implementation provides technical evidence.

Architectural review identifies constitutional relationships.

Specialised engines emerge where recurring operational responsibilities justify stable architectural boundaries.

Applications subsequently compose these capabilities without modifying constitutional principles.

The methodology is examined through the architectural evolution of the Digital Ownership Protocol.

5 The Discipline of Nouns and Verbs

The most useful heuristic to emerge during development concerned the distinction between what a system *is* and what a system *does*.

The constitutional core was found to be most stable when it defined only the enduring nouns of ownership.

Operational modules were found to be most flexible when they defined only the verbs.

The nouns rarely changed.

The verbs changed constantly.

When a new capability was required, the correct engineering question was whether it introduced a new verb composed from existing nouns, or whether it demanded a new noun in the core.

The first was routine.

The second was rare, and treated as a constitutional event.

This discipline provided a practical test for architectural change.

A protocol that must add a noun to support each new verb cannot remain stable.

A protocol whose nouns are sufficient can absorb unbounded operational variety without constitutional revision.

Much of the methodology described in this paper reduces, in practice, to protecting the nouns.

6 Foundational Concepts

This paper distinguishes between constitutional architecture and operational architecture.

Constitutional architecture defines enduring relationships that are expected to remain comparatively stable throughout the life of the protocol.

Operational architecture implements those relationships through specialised engines, bridges, adapters and applications.

Engineering therefore becomes an exercise in preserving constitutional stability while allowing operational systems to evolve.

In the reference implementation this distinction is not left to discipline alone.

The constitutional core is compiled into exactly one boundary module, and every engine, bridge and external system reaches it only through that boundary.

No engine, bridge or provider embeds the core directly.

The separation between constitutional and operational architecture is therefore enforced by the dependency structure itself, and cannot be violated by convention or by accident.

7 Discovering Simpler Abstractions

One of the more consistent observations during implementation was that architectural complexity often indicated the existence of a simpler underlying abstraction.

Rather than continually expanding operational behaviour, development repeatedly sought the more fundamental relationship responsible for that behaviour.

Marketplace transactions became ownership.

Trust became evidence.

Application history became provenance.

Specialised functionality became engines.

This process gradually reduced architectural complexity while expanding the capabilities of the surrounding ecosystem.

Observed implementation consistently reinforced this pattern throughout development.

Complexity, in this methodology, is not a cost to be managed.

It is a signal to be read.

8 Constitutional Engineering as Enforced Practice

A methodology that depends entirely upon discipline will eventually be violated.

The reference implementation therefore encodes part of its constitution as running verification.

Every module declares itself, and a register continuously checks a set of constitutional properties: that each module remains independently valuable, independently testable, independently releasable, independently replaceable, independently understandable, and confined to a single domain.

A module that ceases to satisfy one of these properties produces a recorded architectural event.

Constitutional engineering, in this implementation, is therefore not solely a design philosophy.

It is in part an automated check.

This does not enforce every constitutional principle.

Several remain matters of architectural review.

But the presence of any automated constitutional enforcement suggests that constitutional relationships, once identified, can be protected by mechanism and not only by intention.

9 Observed Behaviour

Implementation history demonstrated a recurring architectural pattern.

Complex operational behaviour frequently indicated the existence of a simpler constitutional abstraction.

Once identified, the abstraction generally remained stable while new operational capability continued evolving around it.

This pattern was observed repeatedly during development.

Marketplace functionality evolved into Commerce.

Commodity abstraction emerged from broader tokenisation considerations.

Tiered modularity clarified relationships between constitutional, operational and application responsibilities.

The constitutional core remained comparatively small throughout.

Observed implementation consistently reinforced architectural simplification rather than architectural expansion.

10 Implications

The engineering methodology presented throughout this paper suggests that software architecture may benefit from identifying constitutional relationships before expanding operational capability.

Rather than treating complexity as an inevitable consequence of system growth, observed implementation indicates that architectural review can often reveal simpler abstractions capable of supporting broader ecosystems.

It further suggests that constitutional relationships, once identified, may be protected structurally — through dependency boundaries and automated checks — rather than through documentation and discipline alone.

If these observations prove applicable beyond the Digital Ownership Protocol, constitutional engineering may provide a useful perspective for future protocol development and long-term architectural evolution.

11 Limitations

The engineering methodology described throughout this paper emerged from a single protocol implementation.

Although observed implementation repeatedly demonstrated similar architectural behaviour, additional implementations are required before broader engineering conclusions can be drawn.

The implementation presently targets a test network, and certain capabilities are built ahead of deployment.

Future work should investigate whether comparable constitutional engineering processes emerge within different technological domains and independent implementations.

Such investigation would strengthen understanding of the methodology beyond the Digital Ownership Protocol itself.

12 Future Research

Future research should investigate whether similar constitutional engineering processes emerge within unrelated software domains.

Comparative analysis involving alternative architectural approaches may help clarify the circumstances under which constitutional engineering provides measurable architectural benefit.

Further implementation experience may also refine the methodology itself, improving understanding of constitutional abstraction, specialised engines and long-term architectural evolution.

Particular attention may be warranted regarding the extent to which constitutional properties can be automatically enforced rather than merely reviewed.

Such investigation will strengthen the evidence supporting the engineering approach proposed throughout this paper.

13 Conclusion

This paper has examined an engineering methodology that emerged through the practical development of the Digital Ownership Protocol.

Observed implementation repeatedly suggested that reducing systems to their simplest constitutional relationships produced architecture that remained comparatively stable while operational capability continued to expand.

It further suggested that these relationships, once identified, may be protected by structure and by mechanism, and not by discipline alone.

Further investigation will determine whether similar engineering principles emerge within other software domains.

For now, one question appears worthy of continued exploration.

What if durable software architecture begins by discovering the relationships that should remain unchanged, rather than the features we first intend to build?