

COMPUTABLE AUTHORITY

A Runtime Reference Architecture for Machine-Executable Law and Governance

POSITION PAPER

Submitted to the OECD Consultation on the Digital Provision of Law:
Towards a Shared Reference Framework for Law as Code — Phase 1

Allan Sendagi

Founder, SafeHaven AI / NOMOS Protocol
August 2026

Abstract

The emergence of autonomous AI agents changes the problem that Law as Code needs to solve. Existing Rules-as-Code and Law-as-Code initiatives address the transformation of legal and regulatory text into machine-readable or machine-executable representations. This is necessary infrastructure for digital government, automated compliance, and increasingly for AI-mediated public services. The OECD's current consultation explicitly frames Law as Code as the state-authorised provision of authoritative law in force as machine-executable representations, while preserving the authority of the underlying legal text. But a further problem emerges when software is no longer merely executing a narrowly programmed workflow and can instead interpret objectives, select tools, formulate actions, and execute them autonomously: **How does an institution express, in a form a machine can evaluate at runtime, the boundary of what that machine is authorized to do?** A machine-executable rule answers what follows when specified conditions are satisfied. It does not, by itself, establish the complete authority relationship between an institution, an agent, an action, a delegation, a version of the governing rules, and the evidence available at the moment of execution. This paper proposes **Computable Authority** as a complementary architectural layer for Law as Code. Computable Authority does not replace authoritative legal text, legal interpretation, institutional discretion, or existing Law-as-Code representations. It makes a different proposition: **as software becomes capable of autonomous action, institutional authority itself must become computationally representable if it is to govern software operating at machine speed.** The proposal distinguishes between **compilation** and **execution**. Compilation is a governance process: authoritative sources are transformed into a versioned computational representation, reviewed, approved, and sealed under institutional authority. Execution is a runtime process: an autonomous system proposes an action and a deterministic evaluator determines whether that action falls within the authority encoded in the applicable artifact. The resulting architecture treats authority as a first-class computational object. A governance artifact can carry its issuer, jurisdiction, scope, delegation, version, effective period, constraints, evidence requirements, dependencies, and integrity information. At runtime, the artifact provides the machine-readable boundary within which autonomous action may occur. The paper proposes a reference architecture spanning authoritative sources, computational representation, authority compilation, sealed authority artifacts, runtime authorization, and institutional feedback. The proposal is grounded in a working implementation, **NOMOS Protocol**, which implements a sealed .nomos governance-artifact format and deterministic evaluation runtime. The .nomos media-type registration request has undergone IANA designated-expert review and received approval for registration as `application/vnd.nomos+json`; the published specification defines .nomos as a JSON-based governance artifact rather than executable code. The central proposition is therefore: **> The transition to autonomous software requires more than machine-readable law. It requires machine-computable authority.**

Keywords: Law as Code · Computable Authority · autonomous agents · governance artifacts · runtime authorization · deterministic evaluation · institutional delegation

Submission note. This paper is submitted as an independent contribution to the OECD consultation. It does not represent the position of the OECD or any government authority and does not claim endorsement by any authority referenced herein.

1. The Next Problem After Law as Code

The question is no longer whether law can be represented computationally.

Increasingly, it can.

Governments, researchers, standards organisations, and technology communities have spent years developing the foundations for machine-readable and machine-executable law. The OECD's 2020 *Cracking the Code: Rulemaking for Humans and Machines* identified Rules as Code as a potentially transformative approach in which governments produce rules in machine-consumable form alongside human-readable law.

Several substantial technical approaches already exist.

Akoma Ntoso provides a mature framework for structured, machine-readable legislative, regulatory, parliamentary, and judicial documents. It became an OASIS Standard in 2018.

LegalRuleML provides a formal representation for legal norms, rules, policies, and reasoning, including temporal, jurisdictional, defeasible, and normative features. Its Core Specification became an OASIS Standard in 2021.

Catala takes a different approach, providing a domain-specific programming language for deriving high-assurance executable algorithms from legislative texts, particularly socio-fiscal law.

Australia and other jurisdictions have conducted sustained Rules-as-Code initiatives, while the OECD's current consultation seeks to develop a shared international reference framework for Law as Code as public digital infrastructure.

The field has therefore moved beyond the question:

Can law be encoded?

A second question now matters:

What happens when the thing executing that encoded law is itself capable of autonomous action?

A system may possess correctly encoded, machine-executable rules and still lack a computational representation of the institutional authority within which it is permitted to act.

A rule may answer:

"If these conditions hold, this outcome follows."

But an autonomous system also needs to answer:

"Am I authorized to cause this outcome?"

That distinction becomes increasingly important as AI systems move from recommending actions to selecting and executing them.

2. Rules Are Not Authority

Law as Code and Computable Authority address related but different problems.

A simplified Law-as-Code pipeline can be represented as:

```

AUTHORITATIVE LEGAL TEXT
↓
COMPUTATIONAL REPRESENTATION
↓
EXECUTABLE RULE
↓
OUTCOME

```

Computable Authority adds an institutional authority layer around that execution:

```

AUTHORITATIVE SOURCE
↓
RULE COMPILATION
↓
COMPUTABLE AUTHORITY ARTIFACT
↓
AGENT REQUEST
↓
AUTHORIZATION EVALUATION
↓
DETERMINISTIC DECISION
↓
ACTION
↓
EXECUTION RECORD

```

The distinction is not that Law as Code is insufficient.

It is that **encoding a rule and authorizing an actor to act under that rule are different institutional functions.**

This distinction can be expressed through the following questions:

Question	Law as Code	Computable Authority
What does the authoritative rule say?	✓	✓
Can a machine process it?	✓	✓
Can the rule be executed?	varies	✓
Which institution issued the authority?	sometimes	✓
Which version applies?	sometimes	✓
What is the authority's scope?	sometimes	✓
Who delegated authority to the acting system?	usually outside scope	✓
Which agent is acting?	usually outside scope	✓
On whose behalf is the agent acting?	usually outside scope	✓
What evidence is required?	varies	✓
Is this action within the delegated boundary?	not necessarily	✓
What happens when authority is ambiguous?	varies	explicit runtime state
Can the resulting authorization be independently checked?	varies	✓

The proposal therefore does not seek to replace Law as Code.

It seeks to establish what sits **between computational rules and autonomous action**.

3. Authority as a First-Class Computational Object

The central proposition of this paper is simple:

An autonomous system should not merely be able to determine what a rule permits. It should be able to operate within a machine-computable representation of the authority that permits it to act.

Computable Authority is therefore the representation of an institution's operational authority boundary in a form software can evaluate.

This is different from treating authority as an instruction buried inside an application.

An ordinary software system might contain:

```
if condition:
    execute()
```

A system based on Computable Authority instead operates against an explicit authority object:

```
authority
  issuer
  jurisdiction
  scope
  delegation
  version
  effective_period
  constraints
  evidence_requirements
  rules
  status
  integrity
```

The authority becomes something that can be:

- identified;
- versioned;
- delegated;
- activated;
- revoked;
- composed;
- evaluated;
- inspected;
- transferred between systems;
- and eventually represented independently of the software implementing it.

This leads to a broader proposition:

Institutional authority can become a first-class object in the software stack.

The significance of this becomes clearer when software begins acting across organisational boundaries.

A future autonomous system may not merely consume an institution's API.

It may consume the institution's **computable authority**.

The distinction is important.

An API exposes what a system can technically do.

A Computable Authority artifact expresses what the institution authorizes that system to do.

4. Why Autonomous Agents Change the Problem

Traditional automation generally follows a relatively constrained execution path.

An organisation specifies:

If X, perform Y.

The software implements that workflow.

An autonomous agent can behave differently.

It may:

- interpret objectives;
- retrieve information;
- formulate plans;
- select tools;
- call APIs;
- initiate transactions;
- communicate with external parties;
- modify records;
- select among multiple possible actions;
- and delegate tasks to other systems.

NIST's 2026 work on software and AI-agent identity and authorization explicitly identifies the growing scale and autonomy of agent actions and the resulting need to apply identity, authentication, and authorization principles to agentic architectures.

Identity and authorization remain essential.

But they do not completely answer the institutional question.

Knowing:

"This is Agent A."

is different from knowing:

"Agent A is authorized to perform this particular action, under this institutional rule, at this point in time, within this scope, using this evidence."

The difference matters because autonomous systems can operate at a speed and scale at which human adjudication of each individual action becomes impossible.

The execution boundary therefore becomes a governance boundary.

Where the machine decides whether an action proceeds is where the institution's authority must ultimately become operational.

This leads to a first-principles observation:

If human institutions govern machines through authority expressed only in forms machines cannot directly evaluate, the institution remains dependent on an intermediary translation layer at precisely the point where autonomous action occurs.

Computable Authority proposes making that boundary computational.

5. What Computable Authority Is — and Is Not

Computable Authority is deliberately narrower than a theory of computational law.

It is not

- a replacement for legislation or regulation;
- a claim that code becomes law;
- a substitute for legal interpretation;
- an autonomous legal interpreter;
- a replacement for courts or regulators;
- a mechanism for deciding what the law ought to mean;
- a claim that all legal rules can or should become deterministic.

It is

A computational representation of the **institutional authority boundary within which software is permitted to act**.

The authoritative legal text remains authoritative.

Interpretive, discretionary, and evaluative elements remain visible rather than being silently transformed into deterministic instructions. This is consistent with the OECD's current consultation, which explicitly states that Law as Code does not change the law or its interpretation and that discretionary elements should not be silently converted into deterministic rules.

The distinction is therefore:

Law determines what the institution has established. Computable Authority determines how that established authority is made operationally

enforceable at the machine execution boundary.

6. Compilation Is Not Execution

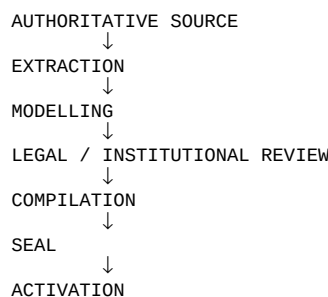
This distinction is fundamental to the architecture.

A language model, compiler, legal expert, policy team, or other transformation process may participate in turning authoritative source material into a computational authority artifact.

That process is **compilation**.

Execution is something different.

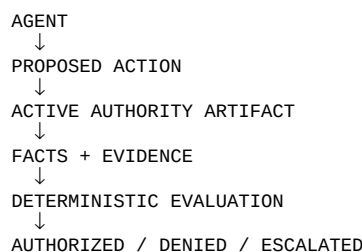
Compilation



Compilation is therefore a governance process.

It is where interpretation, modelling choices, conflict resolution, delegation, versioning, and approval occur.

Once an authority artifact is activated, execution becomes a separate process:



This separation matters because the component that **constructs or interprets authority** does not need to be the component that **enforces authority**.

A language model may assist with compilation.

It should not therefore become the runtime authority.

The runtime should evaluate the already-approved authority artifact.

This produces an architectural boundary:

Probabilistic systems may participate in the construction of authority. Deterministic systems enforce the authority that has been approved.

That separation is one of the principal design principles implemented in NOMOS.

7. The Reference Architecture

The proposed architecture consists of six layers.

Layer 1 — Authoritative Sources

These include:

- statutes;
- regulations;
- binding legal instruments;
- institutional policies;
- delegations;
- standards where they have applicable authority;
- other formally recognised sources of governing rules.

The authoritative source remains the source of legal authority.

Layer 2 — Computational Representation

The source is represented computationally through:

- rules;
- conditions;
- exceptions;
- constraints;
- temporal relationships;
- dependencies;
- responsibilities;
- discretionary elements;
- evidence requirements.

This is where existing Law-as-Code approaches provide important foundations.

Layer 3 — Computable Authority Artifact

The computational representation is packaged into an explicit authority object containing, as applicable:

- issuer;
- jurisdiction;
- scope;
- delegation;
- effective period;
- version;
- dependencies;
- constraints;
- evidence requirements;

- lifecycle state;
- integrity information;
- rules and decision semantics.

This layer answers:

What authority is this machine operating under?

Layer 4 — Runtime Authorization

An autonomous system presents:

- agent identity;
- proposed action;
- relevant context;
- supplied evidence;
- target resource;
- applicable authority.

The authority artifact determines whether the requested action falls within its scope.

Layer 5 — Deterministic Decision

The runtime returns an explicit outcome.

For example:

```
AUTHORIZED
DENIED
ESCALATED
```

The vocabulary can vary between implementations, but the important property is that the result is produced by the declared execution semantics of the authority artifact rather than by an unconstrained language-model response.

Layer 6 — Institutional Record

The system records:

- authority artifact;
- authority version;
- agent identity;
- requested action;
- relevant inputs;
- evidence;
- rules evaluated;
- decision;
- timestamp;
- execution identity.

The record makes the operation attributable and inspectable.

8. NOMOS as a Working Implementation

Computable Authority is not presented here only as a conceptual proposal.

The **NOMOS Protocol** is a working implementation of the architecture.

NOMOS separates the construction of authority from its runtime enforcement.

A `.nomos` artifact is a JSON-based governance artifact. It is not executable code in the conventional sense. It contains structured authority and rule information that a deterministic runtime evaluates against proposed actions.

The current format uses JSON canonicalisation under RFC 8785 and cryptographic sealing. The media-type registration request describes the `.nomos` format as a governance artifact evaluated by a deterministic, deny-by-default runtime and identifies the subtype `application/vnd.nomos+json`. The IANA review record shows the request was approved by the designated expert following revision of the registration.

The artifact model includes, among other elements:

- issuing organisation;
- jurisdiction;
- authority metadata;
- rules;
- constraints;
- governance controls;
- provenance;
- versioning;
- lifecycle state;
- integrity information.

The significance of the artifact is not its file extension.

The significance is that **institutional authority has been made portable as a computational object**.

A `.nomos` file can therefore be understood as a package containing the authority required for a machine to determine whether an action is within an institutionally defined boundary.

This opens a deeper possibility.

Today, software consumes APIs exposed by institutions.

In a future architecture, software may also consume **authority artifacts issued by institutions**.

The institution's authority would no longer exist only as documents interpreted separately by every application.

A defined portion of that authority could become a deployable computational object.

9. The Model Should Not Be the Final Authority

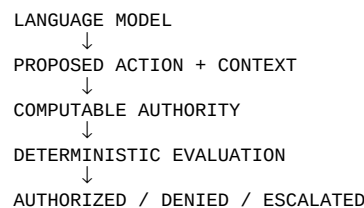
The architecture draws a hard line between **reasoning** and **authorization**.

A language model can:

- interpret a request;
- retrieve relevant material;
- propose an action;
- identify potentially applicable rules;
- classify evidence;
- explain a decision;
- assist with compilation.

But the final authorization boundary should not depend on unconstrained probabilistic generation.

The architecture is therefore:



This is not an argument against language models.

It is an argument for assigning them the correct architectural role.

In NOMOS, rule extraction can use probabilistic systems. The resulting rules are subject to validation and institutional review before becoming sealed authority.

Once sealed, runtime evaluation is separated from the model.

The model may help construct the authority.

It does not decide whether the authority permits the action.

That distinction becomes increasingly important as agents acquire greater operational autonomy.

10. Ambiguity and Discretion

A serious architecture cannot assume that all law is deterministic.

Legal systems contain:

- interpretation;

- discretion;
- evaluative standards;
- exceptions;
- conflicting provisions;
- incomplete evidence;
- unresolved authority;
- institutional judgment.

The correct response is not to pretend these elements do not exist.

They should become explicit computational states.

For example:

Deterministic authority

```
IF A AND B
THEN AUTHORIZED
```

Conditional authority

```
IF A AND B
UNLESS C
THEN ESCALATED
```

Discretionary authority

```
IF A
THEN HUMAN AUTHORITY REQUIRED
```

Unresolved authority

```
IF AUTHORITY CANNOT BE ESTABLISHED
THEN ESCALATED
```

This is particularly important because a machine should not convert the absence of a deterministic answer into a fabricated deterministic answer.

The existence of an **ESCALATE** state is therefore not a failure of the system.

It is an explicit representation of the boundary between computational authority and institutional judgment.

The architecture should make it possible for an institution to say:

"The machine is not authorized to decide this."

That statement is itself a form of computable authority.

11. Authority, Evidence and Time

Authority cannot be separated entirely from context.

A runtime authorization decision may depend upon:

- who is acting;
- what action is proposed;
- which authority applies;
- which version is active;
- when the action occurs;

- what evidence is available;
- what constraints apply;
- what delegation exists.

The result is therefore not simply:

RULE → OUTCOME

but:

```
AUTHORITY
+
AGENT
+
ACTION
+
CONTEXT
+
EVIDENCE
+
TIME
```

↓

AUTHORIZATION DECISION

This matters particularly for autonomous agents because they operate dynamically.

A permission that existed yesterday may no longer exist today.

A policy may have been superseded.

A delegation may have expired.

An agent may have changed role.

A required piece of evidence may be missing.

A system therefore needs more than access credentials.

It needs a representation of the authority conditions under which those credentials may legitimately be exercised.

12. Provenance and Independent Verification

Computable Authority also requires a clear answer to a fundamental institutional question:

What authority governed this machine action?

For a consequential automated action, a system should be able to establish:

- who acted;
- what action was requested;
- under what authority;
- which version was active;
- what evidence was supplied;
- which rules were evaluated;
- what decision resulted;
- when the decision occurred.

These records provide accountability.

Cryptographic integrity can provide an additional property: the ability to establish that an artifact or record has not been altered after issuance.

NOMOS uses cryptographic sealing for authority artifacts and structured execution records for runtime decisions.

These mechanisms are not intended to establish the legal validity of an artifact by themselves.

A cryptographic signature cannot turn an invalid rule into valid law.

Instead, cryptography can establish integrity and provenance around a representation that an institution has already authorised.

This distinction is essential.

Cryptography does not create authority. It protects the representation of authority after authority has been established.

That is why independent verification is a consequence of the architecture rather than its central thesis.

The central thesis remains that authority must become computationally expressible if machines are to be governed at machine speed.

13. Relationship to Existing Law-as-Code and Agent Work

Computable Authority is not proposed as a replacement for existing approaches.

It occupies a complementary position.

OECD / Rules as Code

The OECD's work has established the importance of producing machine-consumable representations of authoritative rules and has now opened a consultation to develop a shared international reference framework for Law as Code.

Contribution of Computable Authority:

The runtime authority boundary between encoded law and autonomous execution.

Akoma Ntoso

Akoma Ntoso provides a mature standard for structured, machine-readable legal and parliamentary documents.

Contribution of Computable Authority:

A structured legal document is not necessarily an operational authority object for an autonomous runtime.

LegalRuleML

LegalRuleML provides a sophisticated representation of legal norms, rules, reasoning, temporality, jurisdiction, and defeasibility.

Contribution of Computable Authority:

A representation of a legal norm does not necessarily establish which autonomous system may act on it, under which delegation, and within what institutional execution boundary.

Catala

Catala demonstrates that statutory provisions can be represented in executable form with high-assurance techniques and is particularly focused on faithful implementation of socio-fiscal legislation.

Contribution of Computable Authority:

Execution semantics and institutional authority semantics are related but distinct. A faithfully compiled program does not by itself answer who is authorized to invoke it, under what delegation, or under which institutional version.

Agent Identity and Authorization

NIST's current work addresses the identity and authorization foundations required for increasingly autonomous software and AI agents.

Contribution of Computable Authority:

Agent identity establishes who or what is acting.

Computable Authority establishes the institutional boundary within which that actor is permitted to act.

These are complementary layers.

14. Objections

"Isn't this just access control?"

Traditional access control generally asks whether an identity has permission to access a resource.

Computable Authority asks a broader institutional question:

Is this particular action authorised under this authority, this version, this context, this evidence boundary, and this delegation?

An agent can possess valid credentials and still be denied because the governing authority does not permit the proposed action.

Computable Authority therefore complements identity and access management.

"Isn't this just policy-as-code?"

Policy-as-code technologies can certainly provide part of the implementation.

But the proposal concerns the **authority lifecycle surrounding executable policy**:

- issuance;
- delegation;
- versioning;
- activation;
- scope;
- evidence;
- constraints;
- revocation;
- composition;
- execution.

Policy-as-code primarily addresses how policy logic is expressed and evaluated.

Computable Authority addresses what makes that executable policy an operational representation of institutional authority.

"What if the encoded authority is wrong?"

This is the most important objection.

Computable Authority does not eliminate the possibility that an institution has encoded its authority incorrectly.

Instead, it makes the compilation process explicit.

A robust implementation should therefore include:

- source provenance;
- traceability to authoritative text;
- institutional review;
- conformance testing;
- version control;
- activation controls;
- integrity protection;
- independent inspection;
- clear lifecycle management.

The objective is not to claim that computational representation cannot contain error.

The objective is to prevent errors in authority representation from becoming invisible software behaviour.

"Code cannot capture legal interpretation."

Agreed.

Computable Authority should not attempt to eliminate legal interpretation.

Where a decision depends on interpretation or discretion that cannot safely be represented computationally, the machine should be able to return:

ESCALATED

rather than inventing a deterministic interpretation.

The ability to represent the boundary of machine authority is itself an important governance capability.

"Why not simply keep a human in the loop?"

Because human presence does not solve a machine-speed governance problem if the human must adjudicate every action.

The human should establish and approve the authority boundary.

The machine should then enforce that boundary at runtime.

Human intervention remains necessary where the authority itself requires judgment, discretion, or escalation.

The objective is therefore not:

humans versus machines.

It is:

human authority expressed in a form machines can faithfully enforce.

15. The Governance Lifecycle

A Computable Authority system should treat authority as having a lifecycle.



Each stage answers a different question.

Source

What authoritative material establishes the rule?

Extract

What computationally relevant provisions exist?

Model

How are those provisions represented?

Review

Has the representation been accepted by the institution?

Compile

What executable authority results?

Seal

Which exact version has been approved?

Publish

Where can authorised systems obtain it?

Assign

Which agents or systems may operate under it?

Verify

Is the authority active and applicable?

Execute

Does the proposed action fall within the authority?

Audit

What occurred under that authority?

Revise

What has changed, and what new version should replace it?

Revision should not necessarily mean rebuilding every downstream representation independently.

Where authorities form hierarchies, a specialised institutional authority could inherit from or extend a shared upstream authority through explicit overlays, additions, removals, and overrides.

Similarly, operational outcomes can become evidence for institutional review.

If repeated execution reveals that an authority produces unexpected or undesirable outcomes, that evidence should feed into a human-controlled revision process.

The system should not rewrite its own authority.

It should surface evidence that the institution may wish to reconsider its authority.

16. Toward Institutions as Computable Objects

The deeper implication of this architecture deserves explicit consideration.

Today, an institution's authority is distributed across:

- legislation;
- regulations;
- policies;
- delegations;
- organisational structures;
- procedures;
- databases;
- access-control systems;
- software;
- human interpretation.

Autonomous systems increasingly require those authority relationships to be available computationally.

This suggests a future in which an institution may expose not only an API, but a **computable representation of its operational authority**.

Such an object could express:

```
WHO IS THE INSTITUTION?
WHAT JURISDICTION DOES IT HAVE?
WHAT AUTHORITY DOES IT HOLD?
WHAT HAS IT DELEGATED?
TO WHOM?
UNDER WHAT CONDITIONS?
FOR HOW LONG?
WHAT ACTIONS ARE PERMITTED?
WHAT ACTIONS ARE PROHIBITED?
WHEN IS HUMAN JUDGMENT REQUIRED?
WHAT VERSION IS CURRENT?
```

The .nomos artifact is an early implementation of this idea.

The longer-term proposition is more ambitious:

Institutional authority itself may become a native object in the software stack.

A machine could therefore consume an institution's authority in much the same way that software today consumes an API or a cryptographic identity.

The institution would not become "a file" in any legal sense.

Rather, a formally defined portion of its operational authority could acquire a machine-native representation that software can consume, evaluate, and enforce.

This is the conceptual territory that Computable Authority opens.

17. What a Shared Law-as-Code Framework Should Consider

The OECD consultation seeks to develop common principles, structures, and requirements for trustworthy and interoperable Law as Code infrastructure.

This paper therefore proposes that a future reference framework consider not only how authoritative law is encoded, but how that representation interfaces with autonomous execution.

In particular, a shared framework should consider:

1. Separation of source, compilation and execution

The authoritative source, its computational representation, and its runtime enforcement should be distinguishable.

2. Explicit authority metadata

Machine-executable law should be capable of carrying or referencing issuer, jurisdiction, scope, effective period, version, delegation and applicable constraints.

3. Explicit treatment of discretion

Interpretive and discretionary elements should remain identifiable and should not silently become deterministic machine decisions.

4. Versioned authority

A consequential machine action should be attributable to the exact version of the computational authority that governed it.

5. Agent binding

The framework should distinguish between the identity of an executing agent and the authority under which that agent is acting.

6. Evidence boundaries

Where an authority depends on facts or evidence, the machine representation should make those requirements explicit.

7. Runtime outcomes

The framework should support explicit outcomes for situations where authority is absent, satisfied, or unresolved.

8. Independent implementation

The representation should not require reliance on a single vendor or proprietary runtime.

9. Conformance

Implementations should be testable against a shared structural and semantic specification.

10. Authority lifecycle

Issuance, review, activation, delegation, revocation, versioning, and retirement should be treated as first-class lifecycle functions.

11. Separation of authority from execution infrastructure

The authority artifact should be portable enough that the institution does not have to surrender control of its authority representation to the software vendor executing it.

12. Composability

Authority should be capable of composition where institutions legitimately inherit, specialise, delegate, or constrain authority from higher-level frameworks.

These requirements do not prescribe a single technical implementation.

They describe properties that a trustworthy ecosystem should be able to support.

18. Why This Matters Now

The technical components required for Computable Authority already exist.

Cryptographic signatures exist.

Version control exists.

Machine-readable rules exist.

Policy engines exist.

Agent identity exists.

Audit systems exist.

The missing element is architectural integration.

The urgency comes from the changing nature of software.

When automation consisted primarily of deterministic workflows designed by humans, institutional authority could remain largely implicit.

The software was effectively an instrument of the organisation.

As autonomous agents increasingly interpret objectives, choose actions, and operate across systems, that assumption becomes weaker.

The machine is no longer merely executing a fixed workflow.

It is operating inside a space of possible actions.

The institution therefore needs to define that space computationally.

This does not require eliminating humans.

It requires moving humans to the level at which they are most effective:

establishing, approving, delegating, revising and withdrawing authority.

Machines can then operate within those boundaries at machine speed.

This is not a case for removing institutional judgment.

It is a case for making institutional judgment **operationally enforceable** once it has been exercised.

19. Closing Thesis

The transition to autonomous AI requires more than machine-readable law.

It requires **machine-computable authority**.

Law as Code addresses the computational representation of authoritative rules.

Agent identity addresses who or what is acting.

Access control addresses permissions over resources.

Policy engines address the evaluation of conditions.

Computable Authority connects these capabilities to the institutional question:

Under whose authority is this machine acting, and what is that authority permitting it to do?

The answer cannot remain entirely in documents that must be interpreted afresh at every point of execution.

If autonomous software is expected to operate at machine speed, then institutional authority must eventually be capable of operating at machine speed as well.

The human institution should establish the boundary.

The computational authority should represent the boundary.

The runtime should enforce the boundary.

And where the authority cannot be safely determined, the machine should stop and return the decision to the institution.

The deeper proposition is therefore not simply that law should become code.

It is that **authority itself must become a computational primitive for the autonomous software era**.

The .nomos artifact is one implementation of that proposition.

The broader question for Law as Code is whether the digital state of the future will merely make law executable — or whether it will also make the authority governing autonomous execution itself computationally explicit.

That is the problem this paper proposes the Law-as-Code community should begin addressing.

References

1. Mohun, J. and Roberts, A. (2020), *Cracking the Code: Rulemaking for Humans and Machines*, OECD Working Papers on Public Governance No. 42, OECD Publishing.
2. OECD (2026), *Consultation on the Digital Provision of Law: Towards a Shared Reference Framework for Law as Code*, Public Consultation, 2026.
3. OASIS LegalDocumentML Technical Committee, *Akoma Ntoso Version 1.0*, OASIS Standard.
4. OASIS LegalRuleML Technical Committee, *LegalRuleML Core Specification Version 1.0*, OASIS Standard, 30 August 2021.
5. Catala Team / Inria, *The Catala Domain-Specific Programming Language*. Catala provides a domain-specific language for deriving executable implementations from legislative texts with high-assurance properties.
6. NIST National Cybersecurity Center of Excellence (2026), *Accelerating the Adoption of Software and AI Agent Identity and Authorization — Concept Paper*.
7. IETF / IANA Media Types, registration record for application/vnd.nomos+json, IANA request #1456754, August 2026.
8. RFC 8785, *JSON Canonicalization Scheme (JCS)*.
9. NOMOS Protocol, *NOMOS-SPEC-001: Governance Artifact Format*, including subsequent protocol extensions.

Author

Allan Sendagi Founder, SafeHaven AI / NOMOS Protocol August 2026

This paper is submitted as an independent contribution to the OECD consultation. It does not represent the position of the OECD or any government authority and does not claim endorsement by any authority referenced herein.