Governance Architecture for Knowledge Graph Construction Using LLM Agents

John C. Strassner[1]

[1]*Futurewei Technologies, 2560 N First St, San Jose, CA 95131*

Abstract

This paper proposes a governance architecture for knowledge graph construction using large language model agents. Semantic agents extract, align, and review candidate knowledge; trusted services control authorization, provenance, lifecycle transitions, schema evolution, approval, and publication. A typestate-inspired contract distinguishes six states from Proposed to Published and binds approval to immutable change sets and governing versions. The paper specifies transition conditions, a trust boundary, and a conditional publication safety argument. A worked example distinguishes grounding, validation, and semantic review. The contribution integrates governance controls architecturally; implementation, formal verification, and empirical evaluation remain future work.

Keywords

knowledge graph construction, large language model agents, multi-agent systems, neuro-symbolic AI, AI governance, provenance, ontology evolution, typestate

1. Introduction

Large language models (LLMs) support knowledge graph (KG) construction through extraction, alignment, conflict resolution, and schema induction. KARMA uses specialized agents for KG enrichment, while AutoSchemaKG combines extraction with dynamic schema induction [1, 2]. These capabilities motivate a separate operational question: under what authority, evidence, and lifecycle conditions may a generated change affect a production graph?

Functional specialization does not itself establish authorization. A plausible proposal may lack evidence, violate policy, depend on obsolete schema assumptions, or be approved by an actor with conflicting responsibilities. Norm-governed agent systems and complete mediation provide foundations for explicit control [3, 4].

This paper proposes a neuro-symbolic governance architecture separating semantic proposal generation from authoritative state changes. Its contributions are: (1) a KG-specific contract linking authority, evidence, and artifact lifecycle; (2) explicit transition conditions and a conditional publication safety argument; and (3) a worked example of agents interacting with trusted enforcement services. This is an architectural specification, not an implemented library or an empirical demonstration of security or performance.

2. Related Work

2.1. Construction and Runtime Enforcement

We compare three approaches selected for direct relevance to complementary concerns: multi-agent KG enrichment, dynamic schema induction, and agent runtime enforcement. This is a focused architectural comparison, not a systematic review.

Additional contact: strazpdj@gmail.com

Proceedings of the 5th International Workshop on Natural Language Processing for Knowledge Graph Construction (NLP4KGC 2026), co-located with the 25th International Semantic Web Conference (ISWC 2026), Bari, Italy, October 26, 2026

jstrassn@futurewei.com (J. C. Strassner)
https://www.futurewei.com/ (J. C. Strassner)
0000-0002-7887-4782 (J. C. Strassner)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



published 2026-11-19

CEUR-WS.org/Vol-4335/paper1.pdf

Table 1
Focused architectural comparison.

Approach	Described capability	Additional contract proposed here
KARMA	Specialized agents, schema alignment, conflict resolution, and scored integration.	Separate assessment from approval authority and version-bound publication.
AutoSchemaKG	Knowledge extraction and dynamic schema induction.	Impact assessment, approval, and controlled schema activation.
AgentSpec	Programmable runtime enforcement of agent constraints.	KG artifact states, provenance obligations, and transactional publication.
This paper	Architectural integration of these governance concerns.	Explicit lifecycle and trust assumptions; implementation and evaluation outstanding.

KARMA coordinates agents for extraction, schema alignment, conflict resolution, and evaluation; its described integration decision aggregates assessment scores [1]. AutoSchemaKG extracts knowledge and induces schemas from source corpora [2]. AgentSpec supplies runtime constraints using triggers, predicates, and enforcement actions [5]. Runtime enforcement is therefore an existing capability, not a novelty claimed here.

Table 1 identifies the additional contract proposed here. The distinctions concern the cited designs; they do not establish that other systems cannot be extended with equivalent controls.

2.2. Governance Foundations

The architecture builds on norm-governed computational societies [3], complete mediation [4], and typestate [6]. PROV-O supplies provenance vocabulary [7]; SHACL validates RDF graphs against specified shapes [8]. Ontology evolution also requires attention to semantic effects and dependent knowledge [9]. Provenance representation does not authenticate recorded claims, and constraint validation does not establish source truthfulness. The contribution is an explicit KG-update contract integrating authority, evidence, lifecycle, schema activation, and publication.

3. Problem Statement

The target system constructs a KG from policy, standards, governance, or technical documents. Candidate updates include assertions, mappings, and schema changes. Acceptance requires evidence preservation, separation of review from authorization, and prevention of stale or unauthorized production changes.

The research question is: how can semantic agents contribute knowledge while independently enforced rules control its admission to the production graph? Each publication must identify its exact approved change set, supporting evidence and decisions, and governing policy, ontology, and graph versions. This process-safety requirement does not imply that accepted assertions or governing policies are correct.

4. Proposed Architecture

The architecture separates probabilistic semantic processing from trusted authorization, validation, lifecycle management, and publication (Figure 1). Services are logical components; they need not be separate processes or depend on a particular agent framework.

4.1. Semantic Agents and Trusted Services

The Reader segments sources; the Extractor proposes assertions and evidence; the Alignment and Fusion Agent proposes mappings and resolves candidate equivalences and conflicts; the conditional Schema

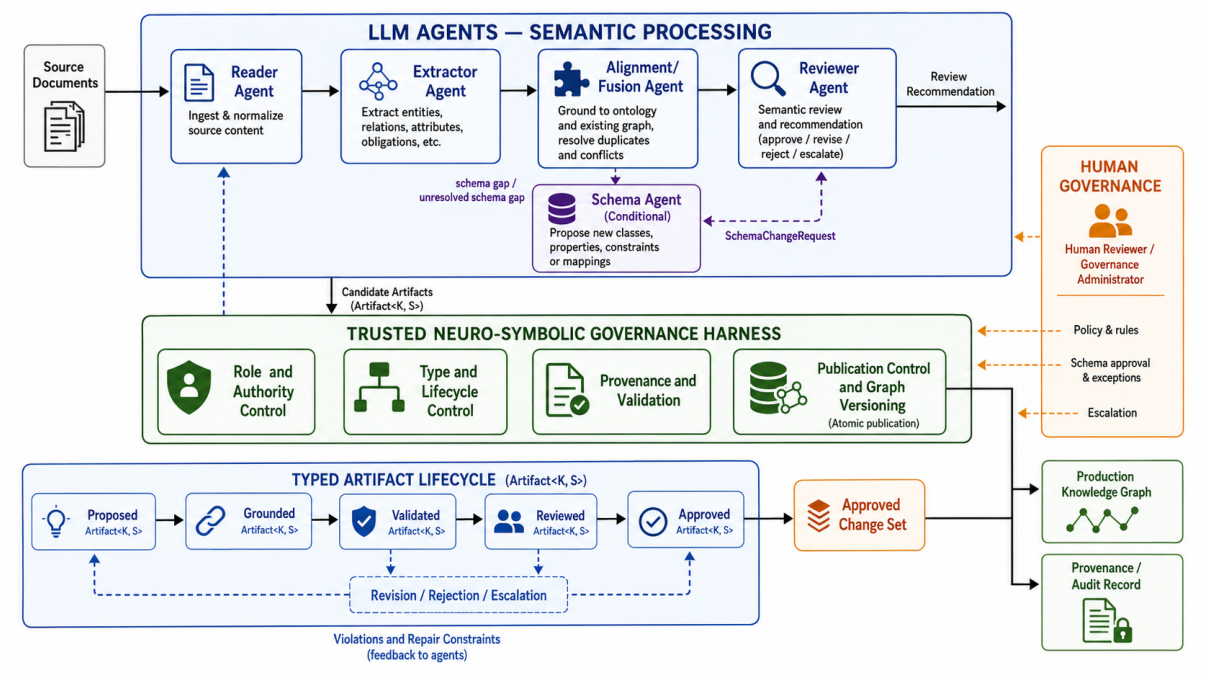


Figure 1: Typed neuro-symbolic governance architecture. Agents propose and review; trusted services enforce transitions and publication. Published denotes successful atomic publication at the right. Dashed paths indicate feedback and governance interactions, not permission to bypass transition guards.

Agent proposes vocabulary changes; and the Reviewer assesses semantic fidelity and recommends a disposition. These roles do not correspond one-to-one with lifecycle states.

Review is not approval authorization. Policy identifies an approval authority: a human or an explicitly authorized non-LLM service. The lifecycle service verifies that authority's decision and other prerequisites before recording approval. The publication controller alone commits production changes. Semantic agents cannot assign roles, modify authoritative lifecycle records, or write to production.

4.2. Trust Boundary and Enforcement Contract

An attacker may control sources, retrieved material, and semantic agents, including their proposals, identity claims, recommendations, and tool requests. The model covers injection, fabricated evidence, impersonation, approval replay, and publication bypass attempts. It assumes integrity of trusted services, their credentials, and protected stores. Compromise of this trusted computing base and denial of service are outside the conditional guarantee.

The trusted base comprises identity and role services, the lifecycle store, policy engine, validators, provenance recorder, version managers, and publication controller. An implementation requires:

1. **Authenticated authority.** The service boundary verifies caller credentials and resolves permissions from a protected registry; agent-supplied role fields convey no authority.
2. **Protected state.** Access-controlled stores hold immutable revisions and authoritative lifecycle records. Requests identify expected revision and state. Guard evaluation, state updates, and audit recording are atomic; failed or indeterminate guards do not advance state.
3. **Recorded evidence.** Trusted ingestion retains source versions and content digests. Provenance links source spans, revisions, processing identities, model identifiers, prompt and tool versions, retrieved context where applicable, and validation, review, and approval records. Integrity protection covers records and references; PROV-O supplies their representation [7]. Digests detect changes, not truthfulness or authorship.
4. **Exclusive publication.** Agents receive no production write credentials. Production interfaces accept only controller writes. Administrative, import, and recovery paths must satisfy equivalent

controls to fall within the guarantee.

These are implementation requirements, not deployment claims. Authenticated service calls, transactional storage, a policy engine, SHACL, and protected audit records provide one implementation approach.

4.3. Lifecycle and Transition Conditions

A candidate revision has type $\mathrm{Artifact}\langle K, S\rangle$, where K is its kind and S its state. Following typestate, operations depend on state [6]. The principal lifecycle for candidate knowledge updates is:

$$\mathsf{Proposed} \to \mathsf{Grounded} \to \mathsf{Validated} \to \mathsf{Reviewed} \to \mathsf{Approved} \to \mathsf{Published}.$$

Supporting validation, review, and approval records have kind-specific rules; they do not recursively traverse this lifecycle. Publication operates on an immutable change-set manifest identifying all member revisions and dependencies. Approval of that manifest requires admissible member revisions; aggregate validation checks interactions among members as well as with the target graph.

Proposed. Authenticated submission registers an immutable candidate revision with origin and initial evidence references.

Grounded. Required spans resolve to retained sources and required ontology and graph mappings are recorded. Unresolved mappings block advancement unless the grounding policy explicitly permits them. Grounding records evidence and referents; it does not certify interpretation.

Validated. Configured type, provenance, ontology, graph, and domain constraints pass against recorded versions. SHACL may implement RDF checks [8]; passing them alone does not establish semantic fidelity.

Reviewed. Required semantic assessments and dispositions are recorded for the exact revision. Recording a review does not imply a favorable outcome.

Approved. An authorized decision binds the exact change set. Required reviews are favorable, separation-of-duty predicates hold, evidence and validation remain applicable, and current policy permits approval.

Published. The controller atomically commits after checking contents, non-revoked approval, current authorization, and expected policy, ontology, and graph versions. The transaction durably links the commit to its approval and resulting graph version.

Every transition requires authenticated operation authority, the prescribed prior state, and stage-specific evidence. Provenance accumulates; early states do not require future review or approval records. Unlisted forward transitions are denied. At minimum, proposer and semantic reviewer identities differ, and the proposer cannot authorize its own change set; policy may impose stricter separation.

Authorized rejection closes an unpublished revision. Escalation blocks advancement until authorized resolution and guard re-evaluation. Revision creates a new Proposed revision; prior validation and approval do not automatically transfer. Supersession and withdrawal require separately governed change sets preserving history.

Schema changes additionally require impact and compatibility assessment, designated approval, and a migration or deprecation plan [9]. Only trusted version management activates an approved schema. Dependent updates are revalidated against that version before publication.

4.4. Conditional Publication Safety

Let C identify an immutable change set and $V = (v_p, v_o, v_g)$ its expected policy, ontology, and graph versions. $\mathrm{Eligible}(C, V)$ requires Approved state, a current approval bound to C and V, applicable validation, complete required provenance, satisfied separation of duties, and no blocking rejection, escalation, or revocation. The guard is:

$$\mathrm{MayCommit}(a, C, V) \iff \mathrm{Publisher}(a) \wedge \mathrm{Eligible}(C, V) \wedge \mathrm{Current}(V).$$

Here a is the authenticated publication service. Guard evaluation and mutation use one serializable transaction or equivalent atomic version checking across all governing stores. A version mismatch blocks publication for renewed assessment; it does not silently rebase approval.

The conditional invariant is that every production mutation after initialization is attributable to a change set satisfying this guard at commit time. By induction over operations, non-publication operations cannot mutate production; publication can do so only after the guard succeeds atomically. This argument assumes exclusive write access, protected authoritative state, and correct enforcement. It is not a machine-checked proof or evidence of implementation security.

Neuro-symbolic feedback returns structured violations and repair constraints to agents. Humans govern policies, exceptional decisions, and high-impact schema approvals.

5. Illustrative Use Case

Consider the synthetic provision: "Operators of category-A systems must retain access logs for 90 days, except for systems covered by exemption E." This illustrates the contract; it is not an observed execution or a statement of law.

Proposed. The Extractor creates x_1, representing operator, system category, obligation, and duration. It cites document d_1, version 1, span s_7, but omits the exemption.

Grounded. The span resolves to retained text, and terms map to ontology identifiers. These links establish inspectable grounding, not faithful interpretation.

Validated. Suppose configured shapes require actor, control, applicability category, and typed duration but do not detect omitted exceptions. Then x_1 passes: structural conformance does not establish source-relative completeness.

Reviewed. The Reviewer compares x_1 with the full passage and requests revision for the missing exemption. The harness records the unfavorable assessment and prevents approval. A new x_2 includes the exemption and undergoes grounding, validation, and review anew.

Approved. After favorable review, the designated authority authorizes a change set containing x_2. The lifecycle service checks separation of duties and binds approval to the manifest and governing versions.

Published. The controller checks approval and current versions and commits atomically. A changed ontology or graph blocks publication for reassessment. Publishing x_1 with x_2's approval fails because contents do not match.

The example distinguishes grounding, constraint validation, semantic assessment, and authorization. Recommendations supply decision evidence; they do not confer publication permission.

6. Discussion, Limitations, and Future Work

Provenance, validation, and policy enforcement do not guarantee correctness. Distinct agent identities do not establish epistemic independence when models or sources are shared.

Policy and ontology changes require expert impact assessment, testing, approval, and versioned activation. Pending updates need reassessment after governing versions change. Dependency records can support selective revalidation; uncertain impact requires broader checks. Published knowledge may require governed supersession or withdrawal. Historical approval does not establish current admissibility.

Identity management, transactions, provenance, and recovery incur costs; checks may block legitimate updates. Overhead and security effectiveness remain unmeasured.

Future work will implement the contract and formalize concurrency, revocation, and recovery. Evaluation will use identical semantic models and inputs across single-agent and prompt-governed multi-agent baselines. Measures include extraction quality, provenance coverage, unauthorized commits, erroneous rejection of independently adjudicated legitimate changes, repair effort, latency, and API cost. Adversarial tests cover injection, fabricated evidence, impersonation, approval replay, version changes, and write-path bypass.

7. Conclusion

This architecture binds KG publication to authorized change sets, evidence, and governing versions. Its explicit transitions and protected write boundary support a conditional safety argument, pending implementation, formal verification, and empirical evaluation.

Declaration on Generative AI. OpenAI's ChatGPT assisted manuscript and bibliography preparation; ChatGPT Images rendered the author-designed Figure 1. The author reviewed and edited the material and takes full responsibility for the contribution.

References

[1] Y. Lu, J. Wang, KARMA: Leveraging multi-agent LLMs for automated knowledge graph enrichment, arXiv preprint arXiv:2502.06472, 2025. URL: https://arxiv.org/abs/2502.06472.

[2] J. Bai, W. Fan, Q. Hu, Q. Zong, C. Li, H. T. Tsang, H. Luo, Y. Yim, H. Huang, X. Zhou, F. Qin, T. Zheng, X. Peng, X. Yao, H. Yang, L. Wu, Y. Ji, G. Zhang, R. Chen, Y. Song, AutoSchemaKG: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora, arXiv preprint arXiv:2505.23628, 2025. URL: https://arxiv.org/abs/2505.23628.

[3] A. Artikis, M. Sergot, J. Pitt, Specifying norm-governed computational societies, ACM Transactions on Computational Logic 10 (2009) 1:1–1:42. doi:10.1145/1459010.1459011.

[4] J. H. Saltzer, M. D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63 (1975) 1278–1308. doi:10.1109/PROC.1975.9939.

[5] H. Wang, C. M. Poskitt, J. Sun, AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents, arXiv preprint arXiv:2503.18666, 2025. URL: https://arxiv.org/abs/2503.18666.

[6] R. E. Strom, S. Yemini, Typestate: A programming language concept for enhancing software reliability, IEEE Transactions on Software Engineering SE-12 (1986) 157–171. doi:10.1109/TSE.1986.6312929.

[7] T. Lebo, S. Sahoo, D. McGuinness, PROV-O: The PROV ontology, W3C Recommendation, 2013. URL: https://www.w3.org/TR/prov-o/.

[8] H. Knublauch, D. Kontokostas, Shapes constraint language (SHACL), W3C Recommendation, 2017. URL: https://www.w3.org/TR/shacl/.

[9] N. F. Noy, M. Klein, Ontology evolution: Not the same as schema evolution, Knowledge and Information Systems 6 (2004) 428–440. doi:10.1007/s10115-003-0137-2.