Knowledge Graph Consumption Layers

Lars Heling[1,*], Munir Abobaker[1], Richard Detlefs[1], Hassan El Hajj[1], Isaiah Onando Mulang'[1] and Felix Sasaki[1]

[1]SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany

Abstract

Knowledge graphs have become fundamental infrastructure for organizing and integrating heterogeneous data across domains. However, different applications require tailored views over the same underlying graph. Current RDF-based approaches lack a standardized mechanism to represent and manage such derived, consumer-facing artifacts while maintaining provenance and schema contracts. In this paper, we introduce *Knowledge Graph Consumption Layers*, a lightweight architectural pattern that addresses this gap. A consumption layer is an RDF dataset comprising a *content graph*, which is derived from a source knowledge graph via a configurable transformation function, and a *metadata graph* that captures provenance, transformation specifications, schema references, and versioning information. Multiple consumption layers can coexist over the same source graph, each optimized for specific consumption scenarios. By organizing layers into *consumption layer collections* that share a common schema, we decouple consumer applications from source-graph evolution by maintaining a stable consumption interface. We formalize consumption layers using RDF named graphs and graph transformation functions, and illustrate their applicability through a real-world use case of API endpoint retrieval for LLM-based agents at SAP. The use case combines two consumption layer collections—description enrichment and graph clustering with community reports—to enable effective retrieval. Our approach complements existing semantic layer patterns by treating consumer-facing artifacts as first-class RDF datasets with explicit transformation semantics and reusable schema contracts.

Keywords

Knowledge Graph, RDF, Data Management, Transformation, Consumption

1. Introduction

Knowledge graphs (KGs) have become a core infrastructure component for organizing, integrating, and exploiting heterogeneous data in both open-data ecosystems and industrial settings [1]. As KGs grow in size and complexity, the challenge is to make their content more *accessible and useful* to a wide range of downstream consumers. Different applications, such as entity retrieval, graph analytics, retrieval-augmented generation (RAG) pipelines, or AI agent tooling, require different views over the same underlying graph. However, there is no native mechanism within the RDF data model to represent and manage such derived, consumer-facing artifacts.

In this paper, we introduce *Knowledge Graph Consumption Layers*, a lightweight architectural pattern for RDF knowledge graphs that addresses this gap. A consumption layer is an RDF dataset consisting of a *content graph* (derived from a source KG via a transformation function) and a *metadata graph* that records provenance, transformation parameters, schema references, and further information. Multiple consumption layers can coexist over the same source graph, each tailored to a specific application. In addition, layers can be organized into *consumption layer collections* that provide interface contracts independent of the underlying transformation. The main contributions of this paper are:

- A formal definition of consumption layers as RDF datasets comprising of a content graph and a metadata graph, together with the notion of graph transformation functions that range from SPARQL CONSTRUCT queries to custom algorithms;

DMKG: 2nd International Workshop on Data Management for Knowledge Graphs, October 25, 2026, Bari, Italy
*Corresponding author.
lars.heling@sap.com (L. Heling); munir.abobaker@sap.com (M. Abobaker); richard.detlefs@sap.com (R. Detlefs); hassan.el.hajj@sap.com (H. E. Hajj); mulang.onando@sap.com (I. O. Mulang'); felix.sasaki@sap.com (F. Sasaki)
0000–0001–9668–8935 (L. Heling); 0009-0007-7448-9258 (R. Detlefs); 0000–0001–6931–7709 (H. E. Hajj); 0000-0002-0554-0511 (I. O. Mulang'); 0000–0002–0985–7399 (F. Sasaki)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-09

CEUR-WS.org/Vol-4322/paper_8.pdf

- The concept of *consumption layer collections*, which group layers under a shared schema to decouple consumer applications from source-graph evolution;
- A concrete use case example demonstrating the applicability of the pattern combining consumption layers for entity retrieval and graph clustering.

This paper is structured as follows: Section 2 discusses related work, Section 3 introduces the formal model for consumption layers, Section 4 presents use case examples, and Section 5 concludes the paper.

2. Related Work

Existing approaches for consumer-facing views over knowledge graphs (KGs) can be grouped into ontology-based data access (OBDA), semantic abstraction, graph summarization, and consumption-oriented embedding layers. OBDA frameworks [2, 3, 4], expose virtual RDF views over relational sources by representing mappings as SQL triple-views and compiling OWL 2 QL axioms into SQL queries, thereby allowing users to query a conceptual schema without seeing the physical data structures. Similarly, enterprise semantic layers and ontologies, as described in Graphwise's semantic layer model [5], provide business-friendly abstractions that translate heterogeneous data into stable domain concepts and relationships for downstream applications. RDF graph summarization techniques construct compact quotient or extractive summaries that can stand in for the original graph in specific analytic tasks, reducing complexity while preserving query-relevant information [6, 1].

In parallel, standards such as `PROV-O` and Dublin Core Terms (`dcterms`) provide vocabularies for modeling provenance, derivation activities, and lifecycle metadata of derived artifacts, while SHACL and ShEx offer shape-based schema contracts that data and views can be validated against [7, 8, 9, 10]. Further, recent work on embeddings and semantic search over knowledge graphs, including graph-embedding empowered entity retrieval and RAG-oriented pipelines that index node and pattern embeddings in vector stores, has highlighted the importance of dedicated retrieval layers that encode entities, relations, and textual context for AI models [11, 12, 13]. Finally, knowledge graph-based API discovery for AI agents shows that representing APIs and their capabilities as graph entities substantially improves agent tooling and discovery workflows [14, 15].

Our notion of knowledge graph consumption layers complements these lines of work by treating such consumer-facing artifacts as first-class RDF datasets consisting of content and metadata graphs, tightly coupling transformation functions, provenance, and reusable schemas into purpose-bound, interchangeable collections over a source knowledge graph.

3. Knowledge Graph Consumption Layers

We propose a novel data management approach called *Consumption Layers* that aims to facilitate consuming knowledge graphs by abstracting away some of the complexity of an underlying knowledge graph to the consumer and providing more consumable and stable artifacts. We introduce consumption layers as a design pattern for RDF knowledge graphs that separates source data from consumer-facing views. Consumption layers therefore provide means to create, manage, and publish extensions to a *source* knowledge graph that can serve various consumption and usage purposes. These layers are therefore only loosely coupled to the source graph's content.

We focus on knowledge graphs represented in the Resource Description Framework (RDF). We assume familiarity with the core RDF data model—triples (s, p, o) where subjects, predicates, and objects are drawn from disjoint sets of IRIs ($\mathcal{I}$), blank nodes ($\mathcal{B}$), and literals ($\mathcal{L}$); RDF graphs as sets of triples; and RDF datasets as collections of named graphs (n, G) with graph names $n \in \mathcal{I}$ [16]. We build directly on the RDF dataset model to define consumption layers.

Definition 1 (Consumption Layer). *A* consumption layer *is an annotated RDF dataset of the form*

$$CL = \{G, \mathcal{T}_\theta, (n_{CG}, CG), (n_{MG}, MG)\}$$

where G is the source graph, $\mathcal{T}_\theta$ *a* transformation function, *CG is a* content graph *identified by* n_{CG} *and MG is a* metadata graph *identified by* n_{MG}.

In a consumption layer, the content graph *CG* contains the content for the consumer. The content is derived from the source knowledge graph *G* by a transformation function $\mathcal{T}$.

Definition 2 (Graph Transformation Function)**.** *An* RDF graph transformation *is a function:*

$$\mathcal{T}_\theta : \mathcal{G} \to \mathcal{G}$$

where $\theta \in \Theta$ *is a parameter that configures the transformation. Given an input graph* $G \in \mathcal{G}$*, the transformation produces:*

$$G' = \mathcal{T}_\theta(G).$$

In a consumption layer $CL = \{G, \mathcal{T}_\theta, (n_{CG}, CG), (n_{MG}, MG)\}$, the content graph is derived from a source graph *G* via a transformation function: $CG = \mathcal{T}_\theta(G)$. For example, a simple way to express a transformation are SPARQL Construct queries. Let *Q* be a SPARQL Construct query. The transformation is defined as:

$$\mathcal{T}_Q(G) = [\![G]\!]_Q$$

where $[\![G]\!]_Q$ is the evaluation of the query *Q* over graph *G*[1] [17]. In a consumption layer, the output graph of the transformation function G' corresponds to the content graph *CG*. While Definition 2 permits any function $\mathcal{T}_\theta : \mathcal{G} \to \mathcal{G}$, in practice we distinguish between *deterministic* and *non-deterministic* transformation functions based on their reproducibility guarantees. A transformation is *deterministic* if, for a fixed source graph and parameters, repeated invocations yields the same content graph[2]. Query-defined transformations, such as SPARQL Construct queries, are typically deterministic. In contrast, *non-deterministic* transformations incorporate stochastic components that may produce different outputs on repeated runs. For example, clustering algorithms (cf. Section 4) are inherently non-deterministic and even with recorded parameters, exact reproduction may not be possible.

This distinction has practical implications for consumption layer management. The metadata graph plays different roles depending on determinism: for deterministic transformations, it enables *reproducibility* by recording the transformation specification; for non-deterministic transformations, it provides *traceability* via PROV-O, documenting what was done and with what inputs, without guaranteeing identical re-execution results.

In order to contextualize the content graph as well as for the purpose of managing consumption layer, the metadata graph *MG* of the consumption layer provides data about the transformation function, such as the transformation function and its parameters, as well as references to the original graph *G*. The metadata can be described using well-known ontologies, such as PROV-O [7] and `dcterms` [8].

Definition 3 (Metadata Graph Requirements)**.** *The metadata graph MG of a consumption layer contains at least:*

- *Provenance information linking CG to source graph G (e.g., using PROV-O),*
- *Transformation specification* $\mathcal{T}_\theta$ *and parameters* θ*,*
- *A Schema reference S which defines the schema of the content graph,*
- *Creation timestamp and versioning information (e.g., using* `dcterms`*).*

The representation of transformation functions in the metadata graph varies by complexity: simple transformations such as SPARQL Construct queries can be embedded directly as string literals, while algorithmic transformations (e.g., clustering, link prediction) reference algorithm names, parameters, and software library and version. LLM-based transformations record model identifiers, prompts, and parameters (such as temperature).

[1] For the sake of simplicity, we abuse notation and assume that $[\![G]\!]_Q$ yields a set of RDF triples rather than solution mappings.
[2] We consider two RDF graphs equal if they are isomorphic, accounting for blank node renaming.

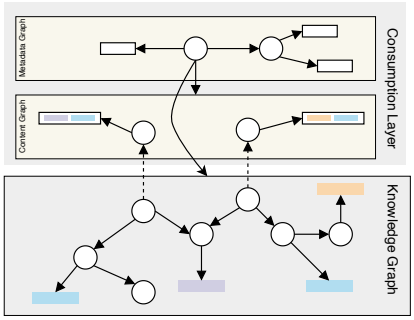


Figure 1: Simplified Description Layer: Nodes in the consumption layer are labeled with aggregated descriptions from the source KG. Dashed arrows link source graph entities to consumption layer annotations.

A key aspect of consumption layers is the re-usability of the content graph schemas. The schema reference S in the metadata graph defines the schema to which the content graph adheres to. As a result, the content graphs of multiple consumption layers can use the same schema independent of the transformation function. We can therefore organize consumption layers into *collections*.

Definition 4 (Consumption Layer Collection)**.** *Given a schema S, we denote a* Consumption Layer Collection *for schema S as:*

$$\mathcal{CL}^S = \{CL_1^S, \ldots, CL_n^S\}$$

with $CL_i^S = \{G_i, \mathcal{T}_{\theta,i}, (n_{CG_i^S}, CG_i^S), (n_{MG_i^S}, MG_i^S)\}$ *a consumption layer and content graph* CG_i^S adheres *to schema S.*

We leave the precise semantics of schema adherence implementation-dependent, requiring only that a decidable validation function $validates(CG, S) \rightarrow \{true, false\}$ exists. In practice, SHACL [18] or ShEx [9] shapes can be used to define the schema of a content graph. In this way, all content graphs from a consumption layer collection conform to the same SHACL or ShEx shapes. Note that $\mathcal{T}_{\theta,i}$ may differ for the consumption layers in the collection to allow for different transformation functions and the source graph G_i is also not fixed by the collection to support graph evolution in the same collection.

While it is possible to create any kind of consumption layer, in practice it is usually sufficient to define a limited set of consumption layer collections. Using consumption layer collections provides several key advantages:

- **Stable consumption interface:** Consumers rely on a fixed schema (in terms of the content graph structure) regardless of the underlying transformation function or layer creator. This allows the source knowledge graph to evolve independently and only the transformation function needs adjustment while the content graph schema remains unchanged.
- **Modular experimentation:** Alternative transformation functions can be developed and tested without modifying consumer applications. For example, different clustering algorithms can be compared for optimal performance by simply pointing to different consumption layers within the same collection.
- **Selective publication:** Consumption layers enable controlled knowledge graph publication. Depending on the consumer's needs, only the consumption layer may be published rather than the entire source graph, reducing complexity and potentially hiding sensitive information from the underlying knowledge graph.

4. Use Case Example

To demonstrate the practical applicability of consumption layers, we present a real-world use case from SAP involving API endpoint retrieval for LLM-based agents. A source knowledge graph represents

API services derived from API specifications, containing endpoints, schemas, and parameters. The core challenge for LLM agents is identifying the correct endpoint from thousands of candidates for a given natural language query. Yet, individual node descriptions in the source graph are too concise to provide sufficient semantic signal for effective retrieval. We address this through two consumption layer collections: (i) the *Description Layer Collection* enriches endpoints with aggregated context and (ii) the *Cluster Layer Collection* to organize endpoints into topical communities for effective filtering.

Description Layer Collection The Description Layer Collection annotates API endpoints with enriched descriptions derived from their graph neighborhood as sketched in Figure 1. Each layer conforms to a shared SHACL schema defining the `cons:hasTag` annotation pattern. The transformation function $\mathcal{T}$ is typically a SPARQL CONSTRUCT query that aggregates related content (Listing 1).

Listing 1: Simplified CONSTRUCT query for a Description Layer.

```
CONSTRUCT { ?endpoint cons:hasTag [
                cons:description ?enrichedDescription ;
                a cons:RetrievalDescriptionTag ] }
{ # Aggregate relevant content for ?endpoint
BIND(CONCAT(?endpointLabel, " ", ?schemaDesc, ...) AS ?enrichedDescription) }
```

Multiple layers can coexist with different aggregation strategies. For example, Layer 1 concatenates schema descriptions and parameter names, Layer 2 includes property data types, Layer 3 uses an LLM to rewrite content into paragraphs. Because all conform to the same schema, consuming applications switch between or combine layers by changing the named graph reference. The metadata graph records each layer's transformation (SPARQL query or LLM prompt) as a PROV-O activity, enabling reproducibility/traceability and provenance.

Cluster Layer Collection The Cluster Layer Collection organizes endpoints into topical communities to enable two-stage retrieval: match queries against cluster reports, then retrieve endpoints only from relevant clusters. Each layer's transformation $\mathcal{T}$ consists of $\mathcal{T}_{\text{cluster}}$ (partitioning entities) and $\mathcal{T}_{\text{report}}$ (generating textual summaries per cluster). The content graph materializes clusters with their reports and tags source entities with cluster membership as exemplified in Listing 2.

Listing 2: Example cluster content graph with community report and annotations.

```
:Cluster_07 a cons:Cluster ;
    rdfs:label "Sales document processing"@en ;
    cons:report "Endpoints for sales orders, items, and billing..."@en ;
    cons:prominentEntity :SalesOrderAPI, :BillingDocumentAPI ;
    cons:size 142 .

:SalesOrderAPI cons:hasTag [ a cons:ClusterTag ; cons:cluster :Cluster_07 ] .
```

Multiple layers can coexist with different algorithms—e.g., Layer A uses Louvain clustering with extractive summarization, Layer B uses Leiden with LLM-generated reports, Layer C uses expert-curated clusters. The metadata graph records the algorithm, hyperparameters, and report generator via PROV-O.

The two collections work synergistically: queries first match cluster reports to select relevant clusters, then match enriched descriptions within those clusters to rank endpoints. Both layers can be swapped independently—testing Layer 1 descriptions with Layer A clustering, then Layer 2 with Layer B—without modifying the retrieval pipeline, demonstrating how consumption layer collections enable modular experimentation while maintaining stable interfaces.

Further Layer Collections Beyond retrieval discussed in the previous section, consumption layers support diverse analytical tasks over the same source graph. A *Link Prediction Layer Collection* predicts

relationships between services frequently used together in business processes, enabling developers to discover complementary APIs when building integrations. The transformation function $\mathcal{T}$ applies link prediction with θ parameterizing the embedding model or co-occurrence threshold derived from process execution logs. The content graph materializes predicted service–service edges annotated with co-usage scores and process context, while the metadata graph records model provenance, training data sources, and prediction confidence via PROV-O. Multiple layers can coexist—Layer 1 using graph neural network embeddings over service dependencies, Layer 2 deriving patterns from process mining over execution traces, Layer 3 incorporating expert annotations—each conforming to the same co-usage schema, allowing applications to switch prediction strategies without code changes.

5. Conclusion

In this paper, we introduced *Knowledge Graph Consumption Layers*, a lightweight architectural pattern for RDF knowledge graphs that separates source data from consumer-facing derived artifacts. We formalized consumption layers as RDF datasets comprising content and metadata graphs, and introduced consumption layer collections that group layers under shared schemas to decouple consumer applications from source-graph evolution. We demonstrated the pattern through a real-world API endpoint retrieval use case at SAP, where Description Layers and Cluster Layers work synergistically to enable effective retrieval for LLM-based agents. Future work includes tooling support for authoring and managing consumption layers, systematic evaluation of query performance and maintainability in large-scale deployments, incremental update strategies for evolving source graphs, and exploring consumption layers as a governance mechanism for controlled KG publication. In addition we consider to enhance the approach to enable capturing of common patterns in transformation components, control flows like branching and iteration, error handling or static validation.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude Opus 4 for Grammar and spell checking and to paraphrase and reword. The author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] X. Wang, G. Cheng, A survey on extractive knowledge graph summarization: applications, approaches, evaluation, and future directions, in: Proceedings of the 33rd IJCAI, 2024. URL: https://doi.org/10.24963/ijcai.2024/916. doi:10.24963/ijcai.2024/916.

[2] J. F. Sequeda, M. Arenas, D. P. Miranker, Ontology-based data access using views, RR'12, Springer-Verlag, Berlin, Heidelberg, 2012, p. 262–265. doi:10.1007/978-3-642-33203-6_29.

[3] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. Zakharyaschev, Ontology-based data access: a survey, in: Proceedings of the 27th IJCAI, AAAI Press, 2018, p. 5511–5519.

[4] D. Calvanese, B. Cogrel, E. G. Kalayci, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Obda with the ontop framework., in: SEBD, 2015, pp. 296–303.

[5] What Is a Semantic Layer? — graphwise.ai, https://graphwise.ai/fundamentals/what-is-a-semantic-layer/, 2026. [Accessed 09-07-2026].

[6] H. Kondylakis, D. Kotzinos, I. Manolescu, Rdf graph summarization: principles, techniques and applications, in: International Conference on Extending Database Technology, 2019. URL: https://api.semanticscholar.org/CorpusID:81985923.

[7] T. Lebo, S. Sahoo, D. McGuinness, PROV-O: The PROV Ontology, W3C Recommendation, W3C, 2013. URL: http://www.w3.org/TR/2013/REC-prov-o-20130430/, latest version available at http://www.w3.org/TR/prov-o/.

[8] Dublin Core Metadata Initiative, DCMI Metadata Terms, https://www.dublincore.org/specifications/dublin-core/dcmi-terms/, 2020. Accessed: 2026-07-13.

[9] E. Prud'hommeaux, I. Boneva, J. E. L. Gayo, G. Kellogg, Shape Expressions Language 2.0, https://shex.io/shex-semantics/, 2017. W3C Community Group Draft Report. Accessed: 2026-07-13.

[10] K. Thornton, H. Solbrig, G. S. Stupp, J. E. Labra Gayo, D. Mietchen, E. Prud'hommeaux, A. Waagmeester, Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation, in: Proceedings of ESWC 2019, Springer-Verlag, Berlin, Heidelberg, 2019, p. 606–620. doi:10.1007/978-3-030-21348-0_39.

[11] Y. Sun, K. Sun, Y. E. Xu, X. Yang, X. L. Dong, N. Tang, L. Chen, KERAG: Knowledge-enhanced retrieval-augmented generation for advanced question answering, in: Findings of the ACL: EMNLP'25, Suzhou, China, 2025. doi:10.18653/v1/2025.findings-emnlp.329.

[12] R. Chen, Retrieval-augmented generation with knowledge graphs: A survey, in: Submitted to Computer Science Undergradaute Conference 2025 @ XJTU, 2025. URL: https://openreview.net/forum?id=ZikTuGY28C, under review.

[13] M. Galkin, J. Wu, E. G. Denis, W. L. Hamilton, Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs, CoRR abs/2106.12144 (2021). URL: https://arxiv.org/abs/2106.12144. arXiv:2106.12144.

[14] M. Lanthaler, C. Gütl, Hydra: A vocabulary for hypermedia-driven web apis, in: LDOW, 2013. URL: https://api.semanticscholar.org/CorpusID:17074932.

[15] I. O. Mulang', J. Thaller, T. Trivedi, L. Heling, F. Sasaki, Representing agentic tools in knowledge graphs for structure-aware tool discovery under tool overload, in: Joint Workshop on GenAIKNORA, 2026. URL: https://openreview.net/forum?id=7MVoH9Y3mi.

[16] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[17] M. Schmidt, M. Meier, G. Lausen, Foundations of SPARQL query optimization, in: L. Segoufin (Ed.), Proceedings of Database Theory - ICDT, ACM, 2010, pp. 4–33. doi:10.1145/1804669.1804675.

[18] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017. URL: https://www.w3.org/TR/shacl/, accessed: 2026-07-13.