Standardizing Knowledge Engineering Practices with a Reference Architecture

Bradley P. Allen[†] ✉ 🏠 ⬤
University of Amsterdam, The Netherlands

Filip Ilievski[†*] ✉ 🏠 ⬤
Vrije Universiteit, Amsterdam, The Netherlands

---- **Abstract** --

Knowledge engineering is the process of creating and maintaining knowledge-producing systems. Throughout the history of computer science and AI, knowledge engineering workflows have been widely used given the importance of high-quality knowledge for reliable intelligent agents. Meanwhile, the scope of knowledge engineering, as apparent from its target tasks and use cases, has been shifting, together with its paradigms such as expert systems, semantic web, and language modeling. The intended use cases and supported user requirements between these paradigms have not been analyzed globally, as new paradigms often satisfy prior pain points while possibly introducing new ones. The recent abstraction of systemic patterns into a boxology provides an opening for aligning the requirements and use cases of knowledge engineering with the systems, components, and software that can satisfy them best, however, this direction has not been explored to date. This paper proposes a vision of harmonizing the best practices in the field of knowledge engineering by leveraging the software engineering methodology of creating reference architectures. We describe how a reference architecture can be iteratively designed and implemented to associate user needs with recurring systemic patterns, building on top of existing knowledge engineering workflows and boxologies. We provide a six-step roadmap that can enable the development of such an architecture, consisting of scope definition, selection of information sources, architectural analysis, synthesis of an architecture based on the information source analysis, evaluation through instantiation, and, ultimately, instantiation into a concrete software architecture. We provide an initial design and outcome of the definition of architectural scope, selection of information sources, and analysis. As the remaining steps of design, evaluation, and instantiation of the architecture are largely use-case specific, we provide a detailed description of their procedures and point to relevant examples. We expect that following through on this vision will lead to well-grounded reference architectures for knowledge engineering, will advance the ongoing initiatives of organizing the neurosymbolic knowledge engineering space, and will build new links to the software architectures and data science communities.

[†] Equal contribution
[*] Corresponding author

Transactions on Graph Data and Knowledge, Vol. 2, Issue 1, Article No. 5, pp. 5:1–5:23
Transactions on Graph Data and Knowledge
TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Knowledge engineering (KE) is the collection of activities for eliciting, capturing, conceptualizing, and formalizing knowledge to be used in information systems [72]. KE includes two broader tasks of creating and maintaining knowledge [3]. Throughout the history of computer science and AI, KE workflows have been a critical component when building reliable intelligent agents across domains and tasks [53]. Indeed, it has been intuitive that developing trustworthy models for applications, from common sense to traffic, crime, and weather, requires well-understood knowledge processes [51]. Similarly, task solutions within these domains, including question answering, summarization, and forecasting, are expected to incorporate standardized KE procedures to be meaningfully applicable and compatible with humans [89].

The importance of knowledge production processes has yielded many notable architectures over the past decades, which aim to synthesize dominant patterns and best practices. As apparent from these architectures, the dominant paradigm of KE has been shifting through the history of AI, from its early days through the eras of expert systems and semantic web. Expert system workflows, like CommonKADS [77], enable the extraction of expert knowledge into knowledge bases based on lifecycle analysis and corresponding models. Many Semantic Web applications can be aligned to a general layered template, most famously the Semantic Web Layer Cake and its contemporary variants [31, 39, 9, 50]. Knowledge graph (KG) workflows [84] and comprehensive toolkits [43] aim to bridge the gap between knowledge bases and their applications, showing a larger emphasis on extensional and possibly less precise modeling of knowledge [79]. Knowledge graph engineering (KGE) emerged as a variant of KE geared towards capturing, representing, and utilizing complex information about entities, their relationships, and their underlying semantics [79, 33]. Researchers and domain experts have devised KGE workflows that are tailored to the needs of a variety of domains like biomedicine [54], library and information sciences [87], web democracy [90], commonsense knowledge [44], and publications [68]. Analogously, enterprise infrastructures, such as the Amazon Product Knowledge Graph [98], have been devised for commercial settings, without clear reference to a standardized workflow. The recent trends in KE, such as the consideration of large language models (LLMs) as knowledge artifacts [64] and the prominence of neurosymbolic systems [92, 72], bring a new perspective to KE. The role of LLMs in KE workflows is studied actively to understand the potential of LLMs to enhance, replace, or add KE components [34]. Meanwhile, recent work based on abstracting semantic web and machine learning systems (SWeMLS) [21] indicated the prominence of KE in neurosymbolic systems, with around a quarter of all SWeMLS patterns corresponding to a KE process.

The present landscape of KE methodologies and tools lacks a comprehensive framework of user needs and available paradigms, as each subsequent KE era does not necessarily include the benefits brought by its predecessors [3]. KE systems require a principled way of considering different user requirements, paradigms, and use cases, thus combining human, social, and technical factors [40]. To address this need, recent work has proposed the formalism of a *boxology*: a hierarchical taxonomy of systemic design patterns expressed in a graphical notation (cf. Figure 2 and the upper right hand corner of Figure 4) [92, 72]. Boxologies provide an opening for aligning the requirements and use cases of knowledge engineering with the systems, components, and software that can satisfy them best, however, this direction has not been explored to date. We see an urgency to understand the scope and purpose of KE in the latest evolving AI landscape, aiming to devise a general framework characterized by requirement-driven best practices. Such a framework should ideally support the perspectives of existing and emerging KE paradigms, enable a flexible definition and support for stakeholder requirements and priorities, and build on top of prior work on KE workflows and systemic patterns. Given the dynamic nature of KE, it should

also provide mechanisms to adapt to evolving requirements over time and across applications. It should provide a prescriptive framework that standardizes best practices while allowing them to be customized to specific circumstances.

This paper proposes a vision of harmonizing the field of knowledge engineering by leveraging the software engineering methodology of devising a reference architecture (RA), inspired by successful RAs designed and applied in domains such as the automotive sector, e-government, and service-oriented solutions [29]. RAs serve as a framework that standardizes a community of practice through a software engineering artifact, based on a survey of relevant stakeholders, their requirements, existing community-based workflows, and suitable evaluation. RAs are developed in a human-centric and iterative manner, which makes them particularly suitable for dynamic and frequently changing disciplines such as KE. They provide a common framework, while simultaneously enabling users to design specific RAs for their narrow use cases. The proposal to develop a methodology for devising RAs for KE is in line with the suggestions in [40]: we hypothesize that the development of RAs will make KE and related knowledge technologies accessible for outsiders and newcomers from disciplines such as software engineering and data science that have compatible goals.

We consider how RAs can be iteratively designed and implemented to associate user needs with recurring systemic patterns, building on top of existing KE workflows and boxologies. Section 2 provides an extended problem statement that motivates the need to consolidate KE practices by building on top of existing boxology patterns. Section 3 provides relevant background on existing RAs and common methodologies for their principled development, and reviews state-of-the-art architectures for KE. Section 4 details a six-step methodology to design and implement a human-centric reference architecture that standardizes KE practices, consisting of scope definition, selection of information sources, architectural analysis, synthesis of an architecture based on the information source analysis, evaluation through instantiation, and, ultimately, instantiation into a concrete software architecture. We describe an initial design of the architectural scope, information sources, and analysis, and prescribe the processes for the design, evaluation, and instantiation of the architecture, as the latter steps are highly use-case dependent. The paper is concluded in Section 5. We expect that following through on this vision will lead to well-grounded reference architectures for knowledge engineering, and will facilitate further links to the software architectures and data science communities.

2 Why a Reference Architecture Framework for KE?

The pursuit of a general reference architecture framework for KE is motivated in this section by three key factors. First, the KE paradigms have been shifting over time, each following one addressing pain points of the existing paradigms, but often failing to address other requirements. Second, KE users vary greatly, and while the user needs have been often discussed in the context of a specific application, a broad view of connecting users, their tasks, and their corresponding requirements is lacking. Third, the emerging boxology of neurosymbolic systems, with its recent link to knowledge engineering, provides a unique opportunity to exploit emerging patterns as components of a more comprehensive architecture.

2.1 Historiographic perspective: Consolidation of the KE paradigms

Prior research on KE is rich, spanning from the 1950s, through the expert systems era of the 1980s, the Semantic Web era, and the recent view of language modeling as a knowledge production process [70, 57, 24, 25, 77, 11, 64, 40, 43, 10, 36, 86, 2]. These different periods have approached KE following the contemporary technological, scientific, and societal focus. We provide a brief historiographic view on KE here, for an extended discussion we refer the reader to [3].

In the 1960s, researchers like Newell and Simon [57] were hopeful about the ability of goal-directed search with heuristics to perform practical general-purpose problem-solving. However, by the 1970s, it became evident that these systems were not easy to scale to complex applications. During the mid-1970s, Feigenbaum [24], influenced by Newell and Simon's work, maintained that focusing on specific domains was crucial for successful knowledge engineering. Knowledge engineers worked on the elicitation of domain-specific knowledge with high quality and expressivity, and domain-independence and scalability were often not prioritized. This period saw a surge in creating expert systems for decision support in businesses, but by the early 1990s, it was clear that these systems had limitations, being brittle and hard to maintain. Efforts to address these limitations included the development of structured methodologies for knowledge engineering during the late 1990s [77]. Feigenbaum [25] persisted in exploring the idea of domain-specific applications but suggested that future systems should be scalable, globally distributed, and interoperable. These ideas, ahead of their time, foreshadowed some aspects of what later became the World Wide Web.

In the era of the Semantic Web, Tim Berners-Lee [11] advocated the use of specific open standards (e.g., RDF and SPARQL) to encode knowledge in Web content, to improve access and discoverability of Web content, and to enable automated reasoning. However, adoption of Semantic Web technology was slow, ultimately leading researchers to seek ways to align these standards and principles more closely with general software industry norms and make them more developer-friendly. Recent efforts, particularly in commercial knowledge graphs developed by companies like Google and Amazon [98], have shown a shift towards custom architectures, often based on property graphs. This shift, while innovative, often sidesteps the interoperability and federation ideals of early visionaries like Feigenbaum and Berners-Lee. As a result, there's a growing need to refine what KE offers developers, focusing on comprehensive, scalable, customizable, and modular infrastructures that integrate with common data formats. KE should be domain-independent, supporting a wide range of use cases with user-friendly interfaces.

The rise of connectionist methods and graphical processing hardware in the 2010s has introduced new possibilities for knowledge production using large language models (LLMs). LLMs have been shown to be a means to provide robustness to missing schema and better generalization across domains and knowledge types. Two main perspectives have emerged regarding the relationship between LLMs and knowledge bases [2]. The first sees LLMs as standalone, queryable knowledge bases that can learn from unstructured text with minimal human intervention [64]. This method challenges traditional, labor-intensive KE processes, but raises concerns about accuracy, ethical use, interoperability, and curatability. The second, more cautious perspective views language models as components in a KE workflow, combining new and traditional methods [43]. This approach emphasizes accessibility, manual editing of extracted knowledge, and explanation of reasoning methods, addressing the limitations of earlier technologies. Both perspectives highlight the importance of sustainability and affordability in KE processes.

2.2   Social perspective: Systematic procedures for incorporating stakeholder tasks and needs

KE tasks can be roughly split into two main categories in terms of their goal: creating and maintaining knowledge artifacts. Here, the knowledge artifacts are typically knowledge graphs, ontologies, and taxonomies [72]. Representative KE tasks are shown in Table 1. These include tasks of creating an ontology or refining that ontology, for example, to include a newly discovered concept or relationship [58, 26, 83]. KE tasks include ingesting and transforming data from multiple data sources into a single artifact, or performing data integration between different schemas [45, 19]. Resulting KGs can be further refined to address issues such as inconsistencies of modeling, contradictions of factual information, outdated information, or missing/incomplete

Table 1 Representative user tasks and scenarios for knowledge engineering.

Task	Scenario
Ontology creation	A new domain is identified, for which an ontology needs to be created.
Ontology refinement	A new concept or relationship is identified in the domain, and the ontology needs to be modified to support it without disruptions.
Data ingest and transformation	Multiple data sources provide overlapping or complementary information. The system needs to transform and normalize this data to ensure consistency in the knowledge graph.
Data source integration	A new data source, in a previously unsupported schema, needs to be incorporated into the knowledge graph while ensuring data quality.
Anomaly detection	The system flags a potential inconsistency or contradiction in the knowledge graph, which needs to be resolved.
Knowledge graph completion	The system flags a missing or incomplete statement, which needs to be automatically inserted.
Human oversight of knowledge graph quality	A subject matter expert (SME) identifies a piece of outdated or incorrect information in the knowledge graph, which needs to be flagged to initiate a correction.
Human feedback	As SMEs interact with the system, they might have insights or suggestions based on their domain expertise, which needs to be supported and incorporated into the refinement process.

statements, to improve their quality [63]. Such issues may be raised by automated systems [78] as well as human subject matter experts (SMEs) [65]. Finally, humans interacting with the knowledge artifact may have further suggestions or feedback for refinement [43].

Commonly, KE procedures identify local requirements for an artifact, with an implicit assumption of the user profile. Meanwhile, user studies are increasingly present in KE research [1, 42], a trend that can be enriched by considering the natural plurality of users. While early KE might have been carried out by computer science practitioners, today it often includes domain experts interacting with knowledge directly, knowledge engineers building ontologies, knowledge editors fixing outdated information, data scientists developing knowledge completion systems, and business/organizational stakeholders that stress-test the available knowledge to understand its value [42]. Considering the example tasks in Table 1, we note that the tasks of ontology creation and data integration require expertise from knowledge engineers and subject matter experts. Refining of knowledge artifacts can be performed by knowledge engineers or data scientists, whereas providing human feedback and oversight requires subject matter experts. Many of these tasks may also benefit from the inputs from business and organizational stakeholders. While here we refer to *stakeholders* as humans that create and maintain knowledge engineering processes, there is an additional set of tasks and users that *use* the artifact resulting from the knowledge engineering process, such as data scientists developing AI prototypes that reason over knowledge and software developers that build knowledge-infused chatbots.

2.3 Component perspective: Preliminary source of architectural patterns

Driven by societal and technical needs for explainability, robustness, and collaboration, neurosymbolic AI has been growing in popularity recently, emerging as one of the key trends of AI research [46, 72]. Each neurosymbolic system combines neural, machine-learning components with

symbolic manipulation. Given the breadth of this definition, there have been attempts to organize neurosymbolic systems by abstracting their architectures using recurring patterns. Following the *boxology* approach [92, 72], data structures can be symbols or data, whereas algorithmic modules are either inductive (machine learning) or deductive (based on knowledge representation and reasoning formalisms). Then, each boxology pattern is a combination of alternating data structure and algorithmic module boxes. The initial boxology work identified 15 such patterns. 11 of these, together with 33 new patterns, were found in the systems systematically surveyed in [14]. The 44 patterns have been classified into a pattern typology based on their complexity, e.g., simple patterns have a single processing unit. Sample patterns from this boxology are illustrated in Figure 2, which we describe in more detail in Section 4.

The question emerges: what is the role of KE in NeSy AI systems? How often do NeSy architectures represent KE processes? An insight into these questions is provided by [72], who coined the term *neurosymbolic knowledge engineering* and analyzed the NeSy approaches that combine machine learning and semantic web components. While we see such component analysis as a step in the right direction of organizing neurosymbolic KE, we identify three challenges for state-of-the-art KE that are apparent from the boxology framework. *Challenge 1:* The patterns should be associated with user requirements, tasks, and application needs to enable their efficient and precise application. The present boxology patterns do not include this information, nor have the mechanism built in to include it in the future. *Challenge 2:* Mechanisms for aligning with ongoing trends and shifting requirements are lacking. These are needed as the set of boxology patterns and their specification (e.g., type constraints) are still largely in flux, as apparent from the large number of newly discovered patterns in [14], and given the lack of specification of how the boxology primitives align with popular NeSy processes (e.g., fine-tuning) and artifacts (e.g., knowledge graphs). *Challenge 3:* There is a lack of a standard for communicating the KE boxology patterns to non-knowledge engineers, including software engineers, data scientists, domain experts, and business stakeholders. While the abstraction of the boxology patterns makes a step towards facilitating broader comprehension, the patterns are still meant for experts.

3   Related Work

The previous section discussed three considerations that motivate the need for a reference architecture framework for the standardization of KE practices. This section summarizes definitions, practices, and methodologies associated with work on RAs, and describes how research in the areas of data, knowledge, and ontology engineering can contribute to the establishment of reference architectures for KE, towards the end of consolidating the three motivating perspectives.

3.1   Reference architectures

Definition and uses A reference architecture is a framework that aligns stakeholders' requirements with design patterns through a final architecture and a corresponding software system. As such, an RA serves as a generic architecture for a class of information systems within a software engineering community of practice [5]. RAs have several shared characteristics: they provide the highest level of abstraction, they heavily emphasize architectural qualities, their stakeholders are considered but absent from the architecture, they promote adherence to common standards, and are effective for system development and communication [7]. Notably, while architectures capture software structures, not every structure is architectural: architecture is an abstraction that should emphasize the attributes that are important to stakeholders [8]. For a comprehensive review of software RAs, we refer the reader to [29].

RAs are driven by two emerging trends [18]. First, an increasing complexity, scope, and size of the system of interest, its context, and the organizations creating the system. Second, increasing dynamics and integration, i.e., shorter time to market, more interoperability, rapid changes, and adaptations in the field. In [8], the authors identify thirteen uses for developing a central reference architecture. A key aspect of reference architectures, along with related types of architectural artifacts, is that they are key to the creation of a technology strategy that drives consensus across multiple groups of stakeholders with an enterprise engaged in software application development for business purposes. Other benefits include that RAs enable the system's quality attributes, enable early prediction of system qualities, encode fundamental design decisions, support the training of new team members, reduce system complexity, and facilitate reuse. Reference architectures provide a common lexicon and taxonomy, a common architectural vision, and modularization [18]. Notably, good architecture is necessary but not sufficient to ensure quality.

Many RAs have been proposed in the past decades, some of which have gained wide adoption in their domains. Well-known examples are AUTOSAR for automotive sector [81], CORBA for object integration through brokers [12], S3 for service-oriented solutions [6], EIRA for e-Government systems,[1] and NIST's Big Data Interoperability Framework [27]. We describe AUTOSAR in greater detail to provide an example of a typical RA. First introduced in 2003, AUTOSAR was developed as a cooperative effort between major automotive manufacturers, suppliers, and tool developers. The primary goal of AUTOSAR is to enable the development of highly modular, scalable, and reusable software components for automotive applications. By providing a common software infrastructure and standardized interfaces, AUTOSAR aims to reduce development costs, improve software quality, and facilitate the integration of software components from multiple suppliers.

Ironically, while the field of Semantic Web puts a lot of emphasis on developing artifacts like ontologies and knowledge graphs that enable common understanding between humans and machines, it has not caught up on the idea of developing architectures, such as AUTOSAR, that will provide a common framework in which different concerns can be expressed, negotiated, and resolved among stakeholders for large, complex knowledge systems [8].

Methodologies for creating RAs. A method to design a software architecture has been proposed by [56], consisting of five steps: establishing its scope, selecting and investigating information sources, performing an architectural analysis to identify architecturally significant requirements, carrying out synthesis of the reference architecture, and evaluating the architecture through surveys as well as its instantiation and use. Typical RAs for big data usually follow a three-step lifecycle consisting of data ingestion, transformation, and serving [7]. Their major architectural components can be roughly grouped into 1) big data management and storage, 2) data processing and application interfaces, and 3) big data infrastructure. Two types of requirements are commonly used to describe stakeholder needs for such software architectures: functional requirements (FRs) and quality attributes (QAs) [8]. *Functional requirements* typically describe what the system components are responsible for, i.e., they state what the system must do and how it must behave or react to runtime stimuli. They are satisfied by assigning an appropriate sequence of responsibilities throughout the architectural design. *Quality attribute* is "a measure or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders." Quality attributes must be characterized using one or more scenarios, and they must be unambiguous and testable.

---

[1] `https://joinup.ec.europa.eu/collection/european-interoperability-reference-architecture-eira/about`

An example of the application of RAs to knowledge engineering is the work of Ocaño et al. on an RA for integrating artificial intelligence and knowledge bases to support journalists and newsrooms [61]. They apply a methodology similar to that of [56], taking domain-specific requirements for the effective support of journalistic activities, defining a reference architecture, and then implementing a prototype instantiation of that architecture. This architecture provides a crucial example of what a realization of an RA for KE would look like for a particular domain. This paper provides a streamlined procedure for instantiating other procedures based on a general RA framework. In the case of general RAs for KE, prior work has devised a set of QAs and FRs [3] based on a historiographic analysis. The present paper considers how this effort can be advanced to result in a general-purpose RA framework for KE.

3.2   Methodologies and workflows for knowledge engineering

Reference architectures can serve as a framework that shapes and optimizes knowledge engineering workflows, ensuring they are efficient, scalable, and compliant with best practices and standards; conversely, knowledge engineering methodologies and workflows can drive the definition of RAs by providing structured approaches to requirements specification and providing specific choices of technologies that constrain the design of a reference architecture.

Knowledge engineering.   From the earliest days of the expert systems era there was a recognition that KE needed a principled methodology [38], but the first complete realization of such a methodology came in the 1990s with the development of KADS [94] and subsequently CommonKADS [77]. CommonKADS is a methodology for the extraction of expert knowledge into knowledge bases based on lifecycle and corresponding models. CommonKADS has been applied to a variety of domains, from e-governance [96] to multi-agent scenarios [41]. The models formalized by CommonKADS are complemented by MIKE's [4] formalization of the execution of the model, and the Protege [30] software for collaborative knowledge production and maintenance. The primary focus of this work is on aspects of task selection, knowledge modeling, and knowledge elicitation, and relatively little attention was paid to architectural aspects and deployment in modern Web-based applications and services, except for the linked data community's emphasis on the use of W3C linked data stack and standards [39]. More recently, the growth of Semantic Web applications has resulted in research into semantic patterns [28] and boxologies that organize systems using abstract components [92]. While these boxologies originally aimed to capture purely automated processes, there have been attempts to include human agents, either as process initiators [91] or following the human-in-the-loop paradigm [95]. With the emergence of knowledge graphs, recent work has devised corresponding workflows for particular domains like the Library and Information Studies (LIS) [87] community, e-commerce applications like the Amazon Product KG [98], and generic workflows for the biomedical domain [55]. Finally, there have been attempts to identify common patterns in knowledge graph workflows [84] and design toolkits [43] that implement these patterns as reusable pipelines of commands.

Ontology engineering.   A specific area of focus within knowledge engineering is ontology engineering (OE) [32]. The Semantic Web era is characterized by a strong focus on the manual development of ontologies [58] and their publishing on the Web using linked data principles, with a strong focus on interoperability, reuse, and integration [66]. Methodologies for ontology engineering developed over the past thirty years include METHONTOLOGY [26], Kendall and McGuinness's Ontology Development 101 [48], and NeOn [83]. As with the KE methodologies described in the previous section, OE methodologies are concerned with the organizational structures and workflows associated with ontology design, knowledge representation (e.g. the modeling of spatio-temporal

modeling [35, 23]), and ontology matching [62]. There has been limited work in understanding the relationship between data governance [49] and ontology engineering; while both disciplines overlap in their concerns for structuring and managing data, the integration of data governance principles into ontology engineering workflows remains a less explored area. OE and KE workflows are abstracted through the neurosymbolic boxology patterns [72], which enables them to be represented in an overarching architecture that is composed of those patterns.

Data engineering. Data engineering (DE) itself has provided a wide range of best practices and workflows in common use across the industry. Standard architectural models of data processing systems include data warehouses [16], data lakes [71], and distributed data processing platforms such as Apache Spark [74]. Recent work on *DataOps* [22] as an adaptation of DevOps principles and best practices to the design and operation of data processing workflows has established many concepts towards the end of ensuring that data ingestion and integration are smooth, continuous, and error-free. These principles include the monitoring of the quality of data to prevent poor quality or inconsistent data from compromising data integrity of the knowledge graph; data versioning, supporting the ability to revert to previous states of the data or understand changes over time; and designing workflows such that the system can scale accordingly without a compromise in performance [7, 82]. All of these techniques can inform the design of architectures for knowledge engineering. The Andreessen Horowitz reference architecture [13] for emerging data infrastructure and platforms is a snapshot of the current industry stack and trends that subsume most current uses of data within an enterprise. This architecture includes several high-level elements, such as sources, ingestion and transport, storage, query and processing, transformation, and analysis and output. It is noteworthy that, while this architecture has been adapted for artificial intelligence and machine learning workflows, it does not refer at all to knowledge graphs or Semantic Web concepts or products, especially given the care it takes to address specific use cases related to machine learning.

4   A six-step roadmap to an RA for KE

By using a requirements-driven approach [8, 5, 18], RA methodologies, informed by recent work on DE, KE, and OE methodologies and workflows, can support the consolidation of different perspectives and paradigms under a single umbrella (*challenge 1*). RAs provide a suitable approach for technological alignment by first identifying, consolidating, and prioritizing user needs, followed by formalizing these needs into functional requirements and quality attributes, and, finally, following an iterative development and evaluation of architectures that satisfy these requirements best. Addressing *challenge 2*, using a requirements-driven iterative design, RAs are developed to suit current technological trends and to be dynamically adapted in the future when the underlying requirements shift significantly. In other words, RAs are designed to be representative of the current technological trends and are flexible to be enhanced over time to suit further developments that are likely to occur in a dynamic field such as KE. As mainstream software engineering artifacts, RAs can facilitate smoother adoption of KE by software engineers and computer/data scientists (*challenge 3*). An RA is a mechanism for meeting practitioners in such fields halfway and enabling a bridge for seamless integration and collaboration between these fields and KE.

We propose that RAs for KE should be designed by applying the mainstream software engineering techniques described in the previous section. We adopt the methodology proposed by Nakagawa et al. [56], consisting of five steps: scope identification (including extraction of requirements), selecting and investigating information sources, architectural analysis, synthesizing
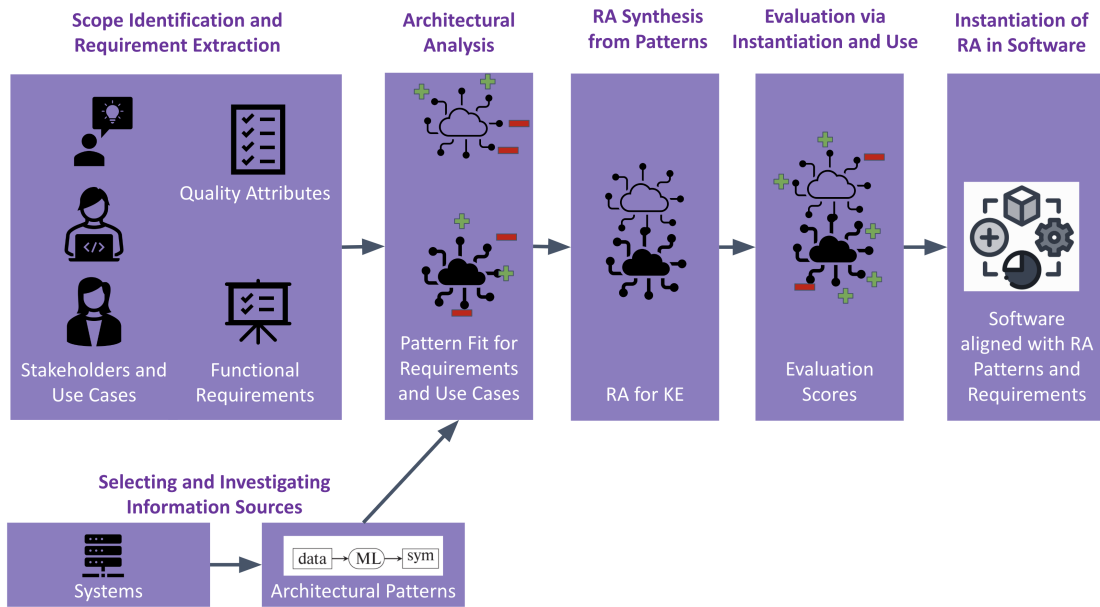
▪ **Figure 1** Pipeline for devising an RA for KE. First, we identify the scope by defining stakeholders and use cases, ultimately resulting in a set of quality attributes and functional requirements [3]. Second, we select and investigate information sources, according to the SWeMLS corpus of neurosymbolic systems and patterns for KE [21, 72]. Third, we connect these components through architectural analysis, yielding information about the fit of various patterns for requirements and use cases. Based on these insights, the fourth step synthesizes an RA from these patterns. Fifth, the RA is evaluated through instantiation and use using a standard software architecture methodology. Finally, the RA is instantiated into software.

an RA, and evaluating the RA through instantiation and use. We include an additional step of instantiating the RA in software, resulting in a six-step procedure. We see the last three steps as iterative steps, which can be modified given the shifting stakeholder requirements, the modular design of the architecture, and the dynamic nature of the underlying technology for the software implementation. The methodology for devising an RA for KE is summarized in Figure 1.

4.1   Scope identification and extraction of requirements

We take inspiration from the software engineering practice of using reference architectures as consolidation mechanisms. On the one hand, the RA framework needs to cover the **use cases** that fall under the task of KE. According to our definition and following [72], KE is a knowledge process that includes knowledge creation (e.g., ontology creation, data ingest) and refinement (e.g., ontology refinement, knowledge graph completion, anomaly detection). The scope of the RA framework should enable machine, human, and joint machine-human knowledge processes [89]. The set of tasks that fall within the scope of KE are listed in Table 1, together with a typical scenario and a question that an RA should be designed to solve. For example, the knowledge graph refinement task can be illustrated with a system flagging a potential inconsistency or contradiction. A question for an RA is how it can facilitate the resolution of such quality challenges.

On the other hand, the RA framework must identify and support the **requirements** of the relevant stakeholders. In software architecture development [18, 85, 8], requirements serve as a common denominator to align the needs of the stakeholders and the technical patterns. While stakeholders may include both knowledge engineers and beneficiaries of KE (e.g., data

scientists building applications), we focus on the requirements of knowledge engineers, i.e., users that perform the aforementioned knowledge graph creation and refinement tasks. As is common in software engineering [8], the requirements can be translated into two categories: functional requirements and quality attributes. In recent work [3], we devised a set of 23 quality attributes and 8 functional requirements for KE, based on a historiographic analysis of the field of KE (similar to Subsection 2.1). We show an excerpt of five quality attributes and five functional requirements in Table 2. These requirements are manually selected to show diverse representative FRs and QAs. Their role is to illustrate to the reader what FRs and QAs for KE (may) look like. An example of QA is modularity, namely, a requirement that the components of the KE workflow enable selective composition for supporting particular use cases. An example of an FR is the import of common data formats, including mainstream semantic web and software sources, as well as serializations. While we consider [3] to provide an initial set of FRs and QAs, we note that the review of papers in this prior work is not based on a systematic selection. The work on analyzing Semantic Web and Machine Learning Systems (SWeMLS) identifies three other requirements: maturity, transparency, and auditability, based on a systematically collected set of papers [72]. Critical future work is to explore how to automatically derive FRs and QAs from such a systematically collected set of papers, based on formally defined requirements. Moreover, given a particular, more narrow scope, the users are expected to define a subset of high-priority requirements that will guide the construction of their RA.

4.2 Selection and investigation of information sources

A systematic analysis of the NeSy landscape, aiming to characterize SWeMLS published between 2010 and 2020, resulted in a corpus of 476 system papers [14]. In this work, each of the papers was annotated with bibliographic information (authors, institutions, publication year, and venue), domain of application, task solved, input/output system architecture, characteristics of the machine learning and the semantic web modules, and levels of maturity, transparency, and provenance. The system components are aligned to the boxology for neurosymbolic systems [92]. In total, 44 patterns were discovered, classified into a typology of six types according to their shapes. Some example boxology patterns from the SWeMLS corpus are shown in Figure 2. The F2 pattern (short for *fusion-2*) is described in [93] as a simple fusion design pattern that takes both symbolic (s) and unstructured data (d) as inputs and produces symbolic data (s) as output using a model M. The F2 pattern corresponds to two specific systems, one being a geological text document classifier [69], and the other an application that classifies heterogeneous web content to create symbolic data extending an enterprise knowledge graph [80].

The data from the study by [21] is made available as a knowledge graph. The ontology of this knowledge graph is centered around the class `System`, which belongs to one `Pattern` and has N `System Component` values. Using SPARQL queries against the SWeMLS knowledge graph, we identified a subset of 139 papers as KE-related, consisting of papers whose systems perform `Graph creation` or `Graph extension` tasks, and produce `Symbol` as the final output. In doing so, we followed the procedure described in [72]. We use this set of 139 KE papers in the rest of our methodology, given the systematic approach to collecting them, their rich annotation, and their alignment with the NeSy boxology components. This procedure illustrates how the selection of information sources can be achieved - in practice, RA developers may decide to focus on a different set of sources, e.g., covering a larger set of papers or a particular subarea of KE for better representativeness to their envisioned use cases.

Table 2 Example QAs and FRs for KE from [3], extended with evaluation criteria. We refer to each requirement with "should" signifying a uniform level of importance. The priority scale of the requirements can be further distinguished according to the specific use case requirements.

requirement	description	evaluation criteria
interoperability [25]	the knowledge produced by the KE process should be easy to share across sites and applications	compatibility with different data formats and standards; ease of integration with other systems; number of supported interfaces/APIs
curatability [10]	the KE process should support human curation of automatically extracted and/or inferred knowledge	effectiveness of human curation interfaces; balance between automation and human oversight; quality control measures for curated knowledge
scalability [25]	the KE process should scale economically with the amount of knowledge produced (measured in terms of rules, triples, nodes, edges, etc.)	performance under increasing amounts of knowledge (e.g., response times, throughput); cost-effectiveness at different scales; system behavior under concurrent user loads
modularity [43]	the components of the knowledge engineering process should be selectively composable to suit a specific use case	independence and interchangeability of system components; ability to integrate or detach modules based on need; impact of module changes on overall system performance
customizability [43]	the components of the KE process should be modifiable to support specific use cases	ease and extent of system modifications; number of customizable components; user feedback on customization features
supports semantic web standards [11]	the KE process should support the use of W3C semantic web standards	use of standard knowledge representation (e.g., RDF, property graphs), serializations (e.g. Turtle, JSON-LD) and query languages (e.g., SPARQL, Cypher), evaluated by ontology quality metrics, pitfall scanning
imports common data formats [43]	the KE process should support the import of data and/or knowledge from data sources	use of standard serializations (e.g., CSV, JSON, Parquet), evaluated by ontology quality metrics, parsing error rate
exports common data formats [43]	the produced knowledge should be exportable to software industry-standard data delivery mechanisms	use of software industry-standard data storage mechanisms (e.g., relational databases, RDF data dumps, search engine indexes) and integration standards (e.g., serialized data dumps, publish/subscribe messaging, REST APIs), evaluated by time to deploy, storage and compute costs
provides user-friendly interfaces [43]	the knowledge produced by the KE process should be accessible and applicable by end users	industry-standard user experience (e.g., command line interfaces, visual editors and browsers, reporting and analytics dashboards) measured by time to complete tasks, user satisfaction surveys
supports heterogeneous query [36]	the knowledge produced by the KE process should be searchable using multiple query languages	use of industry-standard query languages (e.g., SQL, Cypher, SPARQL) and query execution strategies (e.g., federated query, centralized query, find-and-follow), with developer experience measured by time to complete tasks, user satisfaction surveys
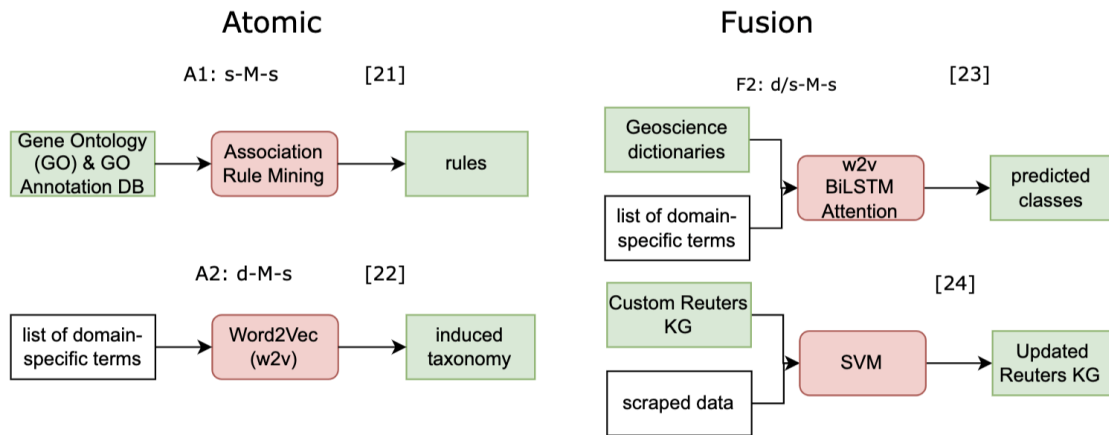
Figure 2 Simple neurosymbolic system design patterns from the SWeMLS KG, as shown in [93]. The F2 design pattern, appearing on the right of the figure, is a simple fusion that takes both symbolic (s) and unstructured data (d) as inputs and produces symbolic data (s) as output using a model M.
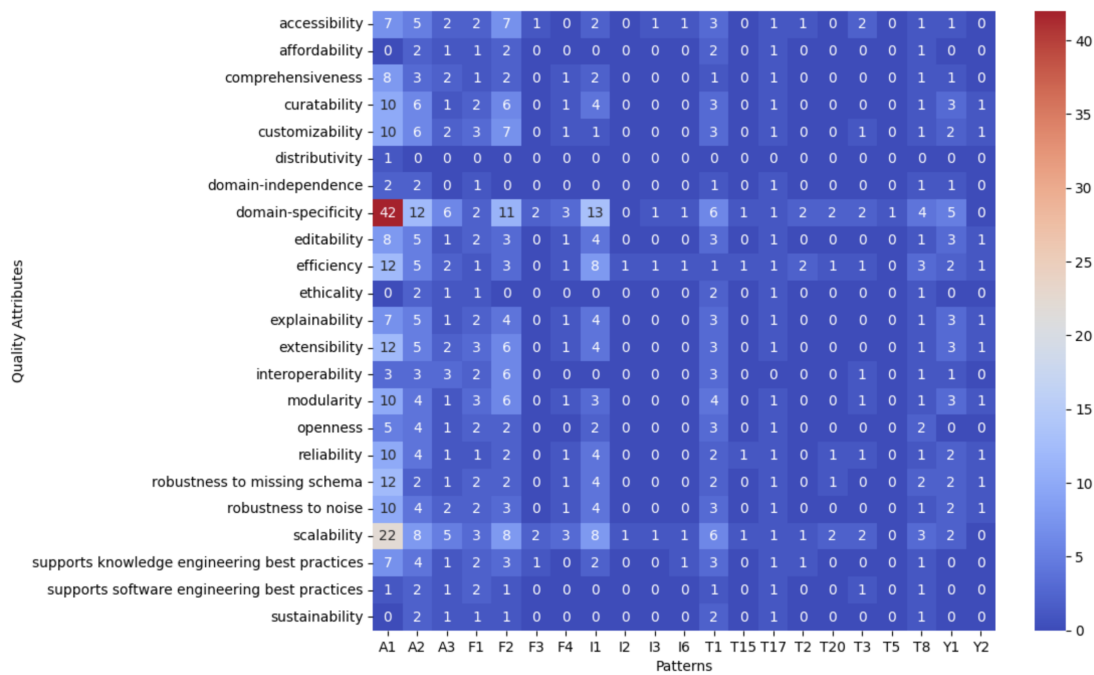
Figure 3 Preliminary analysis of the relationships between quality attributes for KE identified in [3] and the KE design patterns from [72] that are associated with knowledge graph creation and extension. The number in each cell is the count of occurrences of the quality attributes assigned to papers by the zero-shot text classifier that describes systems with the given pattern.

4.3 Architectural analysis

The next step is to perform a preliminary analysis of the extent to which quality attributes for KE are supported within specific SWeMLS patterns. We illustrate this analysis over the SWeMLS KG, where papers describe a system, and each system is associated with a specific pattern. To

establish a connection between quality attributes and patterns, we utilize the SerpApi Google Scholar API to obtain snippets from the abstracts of each of the 139 papers described in the previous section.[2] We then construct a zero-shot text classifier using prompt programming of ChatGPT [75] that, given an article's snippet and title, assigns one or more quality attributes to each paper. Here, we used the 23 QAs identified in the first step. We then aggregate the quality attributes for each paper's system's pattern across all of the papers and patterns. This allows us to derive a **matrix relating quality attributes to patterns**, as shown in Figure 3. From this initial analysis, we find that the A1, A2, F2, and T1 patterns cover the most quality attributes. Namely, A1 covers 20 of the 23 QAs, except for affordability, ethicality, and sustainability; A2 and T1 only lack the QA of distributivity; and F2 covers 20 QAs lacking only distributivity, domain independence, and ethicality. We find these insights to be largely intuitive, as many systems belong to patterns such as A1 and F2. Among the quality attributes, we note that most patterns capture domain-specificity and scalability, whereas ethicality and distributivity are rare. This indicates the tendency of neurosymbolic KE systems to focus on scalability and domain-specificity, whereas aspects such as sustainability, ethicality, and distributivity are gradually gaining momentum but are not yet a primary consideration for most systems.

We emphasize that the corpus used for our analysis is not comprehensive and that the specific analytical methodology followed in this paper may exhibit classification bias. Thus, the significance of this analysis is mainly to show an illustration of how architectural patterns and quality attributes can be linked together. This provides us with a means to determine, given the quality attributes and functional requirements from the scope identification and requirements extraction steps, which pattern(s) are candidates for RA synthesis. We leave it to future work to further tune this procedure, generalize it to a larger dataset, and devise a more robust classification engine. Finally, we note that an analogous procedure can be followed for aligning functional requirements with boxology patterns.

4.4 RA synthesis from patterns

4.4.1 Procedure

Given the architectural analysis of the patterns from the boxology and from other prominent workflows, the construction of the RA follows as a natural synthesis step. Namely, the RA consolidates the discovered pattern(s) with consideration for their adequacy for addressing the use cases and the derived requirements. The benefit of this synthesis is that it prescribes a global view of how a given pattern addressed the stakeholder needs, how the different patterns fit together if a complex pattern is being composed of simpler patterns, and how the architectural pattern(s) can be technically realized; all of that, while aligning with state-of-the-art workflows as reported in the literature.

How would this synthesis of patterns into an RA be realized in practice? As the synthesis is highly dependent on the high-priority requirements of the RA stakeholders, it is impossible to prescribe a one-size-fits-all architecture. Instead, we describe the procedure of how an RA would be synthesized for a specific use case, illustrated in Figure 4. With the prioritized QAs (from Step 1) as a guide, components (from Step 2) that address these attributes (using the analysis from Step 3) will be identified and integrated into the architecture. Practically, an initial design meeting would be scheduled to review the existing workflows and patterns concerning the requirements. During this meeting, a candidate RA will be crafted, following high-level architectural principles

---

[2] `https://serpapi.com/google-scholar-api`, accessed: 2024-01-05.
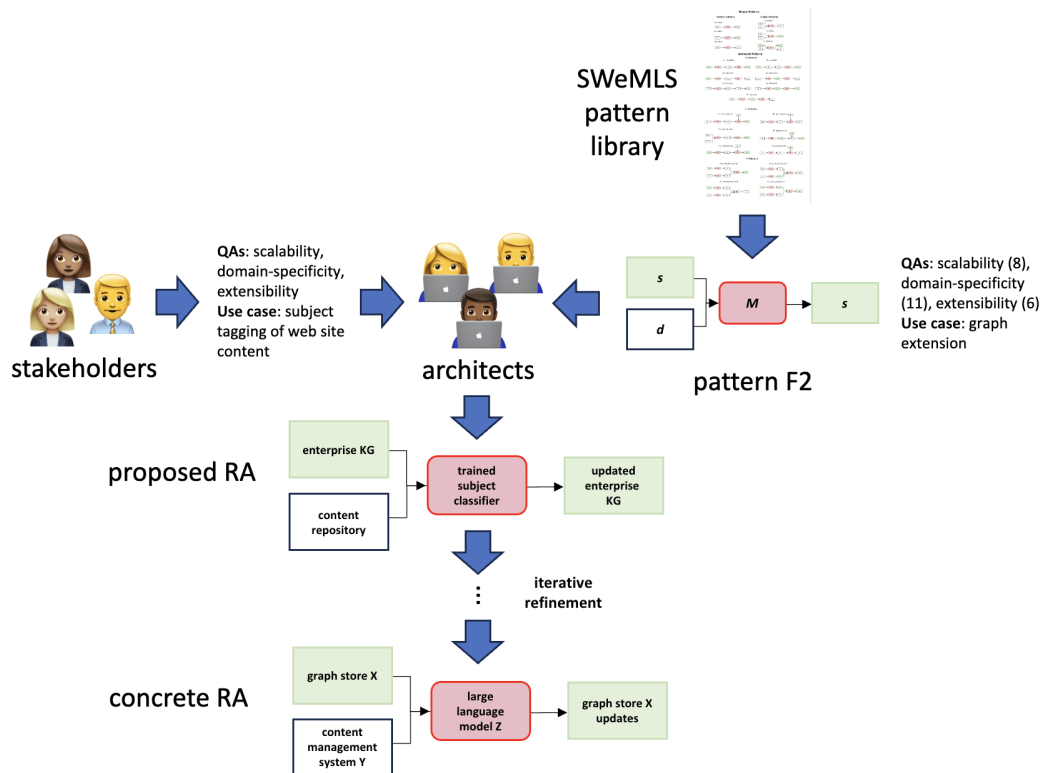
Figure 4 An example of RA synthesis. Stakeholders have identified a set of QAs (scalability, domain-specificity, and extensibility) and a specific use case (graph extension of an enterprise KG). The team of architects has taken this as input, selected an adequate pattern F2 (fusion 2) based on its support for the indicated QAs and use case, and synthesized a proposed RA that uses a trained subject classifier to perform graph extension based on the KG and data from a content repository. After a process of iterative refinement, choices are made about specific technologies to use, and a concrete RA is proposed.

and approaches. A core team of architects is then essential to conduct a collaborative design session. In this session, the RA's architecture, its components, and their interactions are laid out, creating a platform for real-time discussions and potential modifications. During these discussions, the SWeMLS knowledge graph can provide information about potential alternative technologies for the components. To refine the design further, a series of subsequent sessions can be organized that invite a broader set of participants. The feedback gathered from these sessions can be used to iterate and enhance the design.

Following a similar procedure, an example RA for KE in the domain of newsrooms and journalism has been provided by [61]. A critical future work is to apply our process to other use cases with potentially different requirements.

4.4.2 Hypothetical scenario

We proceed to illustrate this process with a hypothetical **scenario** (Figure 4). Business stockholders at a large enterprise have identified that there is a need to improve the discoverability of content on a corporate website. The company has a repository of content, including product information, articles, blog posts, case studies, and user guides. However, users often struggle to find the content they need, leading to frustration, reduced engagement, and potentially lost sales opportunities. The company recognizes that better content recommendations and more intuitive navigation can significantly enhance the user experience and drive business outcomes.

Based on their understanding of industry best practices, the stakeholders determine that a solution that performs subject tagging of content using a domain-specific ontology will support the discoverability of content for tasks their users are attempting to accomplish. They identify several **quality attributes** that they want to ensure such a solution addresses:

- Scalability: The solution should handle a large and growing volume of content and user interactions.
- Domain-specificity: The solution should provide subject tagging from a domain-specific taxonomy of vocabulary terms and definitions.
- Extensibility: The solution should be extensible as new content and subjects become available.

These QAs, along with additional FRs, are presented to a team of architects. The architects review the **design patterns** captured in the SWeMLS KG to determine which of the identified design patterns most closely address these QAs and requirements. The knowledge graph supports the review process by surfacing relevant papers, case studies, and benchmarks for similar systems and use cases. Querying the KG, the architects identify the F2 design pattern as describing systems that match the stakeholder QAs and the use case. Both of the systems corresponding to F2 are similar to the stakeholder use case, and both provide evidence that the pattern can address the specified QAs. Based on this review, the **F2** design pattern is selected.

The architects then specify how the requirements and use case can be addressed by instantiating the components in the F2 design pattern into a **proposed RA**, as follows:

- The input symbolic representation (s) is an enterprise KG that captures the semantics of the content repository and application domain, including the domain subject taxonomy. The KG should capture key entities like products, articles, and customer segments, along with their relationships.
- The input data (d) is the content on the corporate website.
- The model (M) component is a combination of ML and NLP technologies that given the KG and content, classifies the content according to the domain-specific subject taxonomy.
- The output symbolic representation (s) are relations to be added to the knowledge graph to link content on the website to relevant concepts in the vocabulary.

Given these decisions, the architects then proceed to make additional choices for what specific technologies are to be used to implement each component into a **concrete RA**:

- Knowledge graph (s): this could be stored in a labeled property graph database, an RDF triple store, or a relational database with a graph-friendly schema.
- Content repository (d): given the existing website, the input data may reside in various enterprise systems like content management systems or customer-customer relationship management systems.
- Model (M): the model is responsible for producing multi-class subject classifications from the input data and KG, and updating the KG with this new knowledge. Suitable approaches could include hybrid models that combine text, user interactions, and graph structure, e.g., using transformer architectures like BERT or pre-trained language models like GPT-4.

The architects additionally consider factors such as the volume and variety of input data, the complexity of the topic taxonomy, explainability requirements, and the team's AI/ML skills in making their decisions about how to instantiate the RA, including the following considerations:

- The iterative nature of the F2 pattern, where the output enhances the KG, can support continuous improvement of recommendations as new content is incorporated.
- The use of a KG as the core representation to aid explainability, as the relationships between content and topics can be traced and visualized, potentially helping content managers optimize the content strategy and troubleshoot issues.

- The separation of concerns in the F2 design pattern, with dedicated components for data ingestion, model training, and KG management, promotes scalability and performance, as each component can be independently optimized and scaled based on the workload.

The architects then go through a final process of determining the **final proposed architecture**, potentially including:

- Assessing the current state of the KG and identifying gaps in topic coverage
- Inventorying available data sources and evaluating their relevance and quality
- Experimenting with different modeling approaches and comparing their accuracy, scalability, and interpretability
- Validate model outputs with subject matter experts and through user testing

The final proposed architecture is then documented to support the evaluation process described in the next phase of the process. Once the reference architecture has been defined, it can be **stored** in the SWeMLS KG. This allows the RA to be shared and reused in other content classification applications within the enterprise. Some examples of how elements of the reference architecture definition can be mapped into the SWeMLS knowledge graph are:

- The overall RA for content recommendation can be represented as an instance of the `swemls:System` class. The specific pattern it implements (F2) can be indicated using the `swemls:hasCorrespondingPattern` property.
- The business problem of improving content discoverability on the corporate website can be described using the `swemls:Task` class.
- The various data sources used to build and enhance the KG, such as content metadata, user interaction logs, and external taxonomies, can be captured using the `swemls:Data` class.
- The KG serving as the core symbolic representation can be modeled as an instance of the `swemls:SemanticWebResource` class. The specific KG technology used can be specified using the `swTechnology` property.
- The machine learning model used to learn topic classifications can be represented using the `swemls:Model` class.
- The process of training the machine learning models using the input data and KG can be represented using the `swemls:ProcessingEngine` class.
- The specific tools, libraries, and frameworks used to implement the RA components can be captured as instances of the relevant classes, including `swemls:Data`, `swemls:Model`, and `swemls:SemanticWebResource`.

By mapping the RA to the SWeMLS ontology in this way, we establish a structured and semantically rich representation of the architectural knowledge that can be a resource in the evaluation process described in the next section. The ontology classes, properties, and relationships provide a standardized vocabulary to describe the various aspects of the RA, from the business goals and QAs to the technical components and best practices. This consistent representation facilitates comparison, integration, and reasoning across different RAs and domain applications.

4.5 RA evaluation through instantiation and use

Once specified, architectures synthesized in this manner from design patterns can then be evaluated through a lightweight version of the Architecture Tradeoff Analysis Method (ATAM). ATAM [47] is a risk-mitigation process used to identify architectural risks that have implications in fulfilling quality attributes. As originally proposed by CMU's Software Engineering Institute, this process involved a multi-day face-to-face gathering of stakeholders and architects. A lightweight ATAM process [73] is a streamlined version of the traditional ATAM, focusing on a shorter,

more rapid timeframe and often less resource-intensive evaluation of architectural decisions. The use of web-conferencing and real-time collaborative document editors allows this process to be conducted remotely, increasing the ability to gather a large and diverse group of stakeholders. First, we will identify and recruit a set of stakeholders for an ATAM session. On the day of the session, the stakeholders will be presented with an agenda for the session that defines the scope of the evaluation and presents the RA, identifying architectural approaches used. This will be followed by a presentation of user scenarios relative to evaluating the RA. The user scenarios for knowledge creation and maintenance processes should include capabilities for data integration from multiple structured sources [20], data quality checks [67], entity resolution [17], ontology merging and alignment [15, 62], query optimization [37], and natural language processing [76] (cf. Table 1). Moreover, the production processes should be automated to enable efficient updates and maintenance of the knowledge artifact [59]. In cases where the KE involves the use of personally identifiable information or other sensitive data for knowledge elicitation or training ML components, there is the danger of leakage of sensitive information; in addition, ML components themselves can inadvertently leak data under adversarial attack [60]. Therefore, the production process should incorporate mechanisms for security and privacy, as well as access control mechanisms to ensure that the data stays secure and that only authorized users have access. It is worth observing that many of these issues have been explored to date in the more generic context of data engineering and data science architectures and platforms. Once the stakeholders have considered the various scenarios, they can proceed to collaboratively analyze the scenarios, identifying risks and trade-offs, and gathering feedback, focusing on potential refinements and architectural alternatives. The stakeholders will then document risks, trade-offs, architectural decisions, and the reasons for them, finishing by summarizing the final consensus RA.

4.6   RA instantiation in a concrete software architecture

Given a consensus RA, we can proceed to finalize a comprehensive architectural blueprint. The RA does not provide absolute recommendations on such choices, assuming that those are stakeholder need-dependent. It does, however, prescribe an association between different requirements, architectural patterns, and adequate implementations. For each component, the range of options for its instantiation using existing software packages or through bespoke development will be identified. Here, we are inspired by recent toolkits for knowledge graphs, like KGTK [43], which connect knowledge engineering operations by defining a universal interface format and abstracting the implementation of each component from the user. The implementation of each component relies on thorough research and consideration of the best existing tool or implementation that can be wrapped, i.e., that the software can provide an interface to. For instance, one could provide an interface to Pytorch-Biggraph [52] for knowledge completion, Shape expressions (Shex) tools [88] for using constraints to evaluate quality, and RLTK [97] for record linkage across knowledge artifacts. However, as KGTK and similar toolkits are built based on an implicit set of use cases and user requirements, further investigation is required to assess whether they will align with emerging architectural contributions like the set of boxology patterns.

A strategic approach to instantiating an RA involves a phased implementation. Each phase should predominantly focus on one specific component. During this development phase, constant testing and evaluation of each component will be performed to ensure the component aligns with the predefined QAs and specific scenarios. After the conclusion of each phase, feedback will be gathered from all involved stakeholders. This iterative process will ensure the architecture remains relevant and effective, as necessary revisions based on the feedback can be made. The resulting implementation will be open-sourced and thoroughly documented in a publicly accessible code repository. After the entire process is complete, the system's efficacy will be tested in pilot

trials facilitated by the stakeholders. During these trials, relevant data on system performance and quality assessment will be collected to ensure the architecture's robustness and efficiency. Notably, the implemented RA would serve as a comprehensive framework that enables decisions on technology for representation, integration, and quality assurance, among others, to be made based on high-priority requirements. While the implementation is meant to be prescriptive and enable efficient KE by profiles with various backgrounds, goals, and levels of expertise, we acknowledge that these recommendations should be considered relative to the stakeholders' needs.

5 Conclusions

Knowledge engineering, as a process of creating and maintaining knowledge artifacts, has remained relevant throughout the history of AI. In light of the heterogeneous requirements and KE use cases, on the one hand, and the emergence of architectural components and partial workflows, on the other hand, this paper makes a case for developing reference architectures for KE. Following software engineering practices, an RA would provide an organizational principle for isolated systemic patterns, thus providing a key contribution to this ongoing work that enables the knowledge engineering field to be systematized. A reference architecture consolidates the patterns while simultaneously considering its scope, defined through a set of use cases and their corresponding requirements, distilled as quality attributes and functional requirements. The synthesis of the architecture is an iterative process, inspired by success stories of reference architectures for service-oriented design, e-government, and the automotive sector. A key aspect of the development is its evaluation through instantiation and use with representative users for representative KE tasks. As a final step, the reference architecture components need to be instantiated into software, thus closing the cycle between user needs and existing technological capabilities.

While this paper outlines a roadmap for devising comprehensive and requirement-grounded RAs for KE, its realization in practice is partial at present. We present a broad definition of scope through a definition of representative tasks and distillation of 23 quality attributes and 8 functional requirements, which could be narrowed down given a specific use case. We take the recently identified collection of system patterns for neurosymbolic KE as information sources, providing initial components that can be used to construct an RA. We present an architectural analysis, as a direct mapping between QAs and the identified architectural patterns, detecting requirements with various levels of support. The steps of synthesizing an RA from patterns, evaluating the RA through instantiation and use, and instantiating the RA into software are presented as prescriptive, consolidating best practices and methodologies from software engineering through step-by-step processes, because these steps are highly dependent on the specific use cases. Each of these steps requires the dedicated effort of iterative design, development, implementation, and evaluation of an RA, which we plan to pursue as the next steps for representative subsets of tasks and domains. We believe that the presented methodology for devising RAs for KE provides an important extension of emerging work that systematizes KE methods, by providing a mechanism to associate architectural patterns with user requirements and identify potential gaps. We invite the broader community of interested researchers and developers to join us in these discussions and complement our future efforts in consolidating KE practices.

References

1 David Abián, Albert Meroño-Peñuela, and Elena Simperl. An analysis of content gaps versus user needs in the wikidata knowledge graph. In *International Semantic Web Confer-* *ence*, pages 354–374. Springer, 2022. `doi:10.1007/978-3-031-19433-7_21`.

2 Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona Diab, and Marjan Ghazvininejad. A review on language models as knowledge bases. *arXiv*

preprint arXiv:2204.06031, 2022. `doi:10.48550/arXiv.2204.06031`.

3 Bradley P. Allen, Filip Ilievski, and Saurav Joshi. Identifying and consolidating knowledge engineering requirements. *arXiv preprint arXiv:2306.15124*, 2023. `doi:10.48550/arXiv.2306.15124`.

4 Jürgen Angele, Dieter Fensel, Dieter Landes, and Rudi Studer. Developing knowledge-based systems with mike. *domain modelling for interactive systems design*, pages 9–38, 1998.

5 Samuil Angelov, Paul Grefen, and Danny Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 141–150. IEEE, 2009. `doi:10.1109/WICSA.2009.5290800`.

6 Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. S3: A service-oriented reference architecture. *IT professional*, 9(3):10–17, 2007. `doi:10.1109/MITP.2007.53`.

7 Pouya Ataei and Alan Litchfield. The state of big data reference architectures: A systematic literature review. *IEEE Access*, 2022. `doi:10.1109/ACCESS.2022.3217557`.

8 Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. SEI Series in Software Engineering. Addison-Wesley Professional, fourth edition, 2022.

9 Wouter Beek, Laurens Rietveld, Stefan Schlobach, and Frank van Harmelen. Lod laundromat: Why the semantic web needs centralization (even if we don't like it). *IEEE Internet Computing*, 20(2):78–81, 2016. `doi:10.1109/MIC.2016.43`.

10 Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pages 610–623, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3442188.3445922`.

11 Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

12 Juergen Boldt. The common object request broker: Architecture and specification. Specification formal/97-02-25, Object Management Group, jul 1995. URL: `http://www.omg.org/cgi-bin/doc?formal/97-02-25`.

13 Matt Bornstein, Jennifer Li, and Casado. Martin. Emerging architectures for modern data infrastructure, 2020. `https://future.com/emerging-architectures-modern-data-infrastructure/`.

14 Anna Breit, Laura Waltersdorfer, Fajar J Ekaputra, Marta Sabou, Andreas Ekelhart, Andreea Iana, Heiko Paulheim, Jan Portisch, Artem Revenko, Annette ten Teije, et al. Combining machine learning and semantic web: A systematic mapping study. *ACM Computing Surveys*, 2023. `doi:10.1145/3586163`.

15 Niladri Chatterjee, Neha Kaushik, Deepali Gupta, and Ramneek Bhatia. Ontology merging: A practical perspective. In *Information and Communication Technology for Intelligent Systems (ICTIS 2017)-Volume 2 2*, pages 136–145. Springer, 2018.

16 Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997. `doi:10.1145/248603.248616`.

17 Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)*, 53(6):1–42, 2020. `doi:10.1145/3418896`.

18 Robert Cloutier, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole, and Mary Bone. The concept of reference architectures. *Systems Engineering*, 13(1):14–27, 2010. `doi:10.1002/sys.20129`.

19 Xin Luna Dong and Divesh Srivastava. Schema alignment. In *Big Data Integration*, pages 31–61. Springer, 2015.

20 Fajar Ekaputra, Marta Sabou, Estefanía Serral Asensio, Elmar Kiesling, and Stefan Biffl. Ontology-based data integration in multidisciplinary engineering environments: A review. *Open Journal of Information Systems*, 4(1):1–26, 2017. URL: `https://www.ronpub.com/ojis/OJIS_2017v4i1n01_Ekaputra.html`.

21 Fajar J Ekaputra, Majlinda Llugiqi, Marta Sabou, Andreas Ekelhart, Heiko Paulheim, Anna Breit, Artem Revenko, Laura Waltersdorfer, Kheir Eddine Farfar, and Sören Auer. Describing and organizing semantic web and machine learning systems in the swemls-kg. In *European Semantic Web Conference*, pages 372–389. Springer, 2023. `doi:10.1007/978-3-031-33455-9_22`.

22 Julian Ereth. Dataops-towards a definition. *LWDA*, 2191:104–112, 2018. URL: `https://ceur-ws.org/Vol-2191/paper13.pdf`.

23 Vadim Ermolayev, Sotiris Batsakis, Natalya Keberle, Olga Tatarintseva, and Grigoris Antoniou. Ontologies of time: Review and trends. *International Journal of Computer Science & Applications*, 11(3), 2014. URL: `http://www.tmrfindia.org/ijcsa/v11i34.pdf`.

24 Edward A Feigenbaum. The art of artificial intelligence: Themes and case studies of knowledge engineering. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, volume 2. Boston, 1977. URL: `http://ijcai.org/Proceedings/77-2/Papers/092.pdf`.

25 Edward A. Feigenbaum. A personal view of expert systems: Looking back and looking ahead. *Expert Systems with Applications*, 5(3):193–201, 1992. Special Issue: The World Congress on Expert System. `doi:10.1016/0957-4174(92)90004-C`.

26 Mariano Fernández-López, Asuncion Gomez-Perez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. *Engineering Workshop on Ontological Engineering (AAAI97)*, mar 1997.

27 DRAFT NIST Big Data Interoperability Framework. Draft nist big data interoperability framework: Volume 6, reference architecture. *NIST Special Publication*, 1500:6, 2015.

28 Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009. `doi:10.1007/978-3-540-92673-3_10`.

29 Lina Garcés, Silverio Martínez-Fernández, Lucas Oliveira, Pedro Valle, Claudia Ayala, Xavier Franch, and Elisa Yumi Nakagawa. Three decades of software reference architectures: A systematic mapping study. *Journal of Systems and Software*, 179:111004, 2021. `doi:10.1016/j.jss.2021.111004`.

30 John H Gennari, Mark A Musen, Ray W Fergerson, William E Grosso, Monica Crubézy, Henrik Eriksson, Natalya F Noy, and Samson W Tu. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1):89–123, 2003. `doi:10.1016/S1071-5819(02)00127-1`.

31 Randy Goebel, Sandra Zilles, Christoph Ringlstetter, Andreas Dengel, and Gunnar Aastrand Grimnes. What is the role of the semantic layer cake for guiding the use of knowledge representation and machine learning in the development of the semantic web? In *AAAI Spring Symposium: Symbiotic Relationships between Semantic Web and Knowledge Engineering*, pages 45–50, 2008. URL: `http://www.aaai.org/Library/Symposia/Spring/2008/ss08-07-006.php`.

32 Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media, 2006.

33 Paul Groth, Elena Simperl, Marieke van Erp, and Denny Vrandečić. Knowledge graphs and their role in the knowledge engineering of the 21st century (dagstuhl seminar 22372). *Dagstuhl Reports*, 12(9), 2023. `doi:10.4230/DagRep.12.9.60`.

34 Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023. URL: `http://papers.nips.cc/paper_files/paper/2023/hash/f9f54762cbb4fe4dbffdd4f792c31221-Abstract-Conference.html`.

35 Anaïs Guillem, Antoine Gros, Kévin Réby, Violette Abergel, and Livio De Luca. Rcc8 for cidoc crm: semantic modeling of mereological and topological spatial relations in notre-dame de paris. In *SWODCH'23: International Workshop on Semantic Web and Ontology Design for Cultural Heritage*, 2023. URL: `https://ceur-ws.org/Vol-3540/paper2.pdf`.

36 Olaf Hartig. Reflections on Linked Data Querying and other Related Topics. `https://olafhartig.de/slides/Slides-DKG-SWSA-Talk.pdf`, 2022. Accessed: 2022-03-17.

37 Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings 8*, pages 293–309. Springer, 2009. `doi:10.1007/978-3-642-04930-9_19`.

38 Frederick Hayes-Roth, Donald A Waterman, and Douglas B Lenat. *Building expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1983.

39 James A Hendler. Tonight's dessert: Semantic web layer cakes. In *European Semantic Web Conference*, pages 1–1. Springer, 2009. `doi:10.1007/978-3-642-02121-3_1`.

40 Aidan Hogan. The semantic web: Two decades on. *Semantic Web*, 11(1):169–185, 2020. `doi:10.3233/SW-190387`.

41 Carlos A Iglesias, Mercedes Garijo, José C González, and Juan R Velasco. Analysis and design of multiagent systems using mas-commonkads. In *Intelligent Agents IV Agent Theories, Architectures, and Languages: 4th International Workshop, ATAL'97 Providence, Rhode Island, USA, July 24–26, 1997 Proceedings 4*, pages 313–327. Springer, 1998. `doi:10.1007/BFb0026768`.

42 Ana Iglesias-Molina, Kian Ahrabian, Filip Ilievski, Jay Pujara, and Oscar Corcho. Comparison of knowledge graph representations for user consumption scenarios. In *International Semantic Web Conference (ISWC) Research Track*, 2023. `doi:10.1007/978-3-031-47240-4_15`.

43 Filip Ilievski, Daniel Garijo, Hans Chalupsky, Naren Teja Divvala, Yixiang Yao, Craig Rogers, Rongpeng Li, Jun Liu, Amandeep Singh, Daniel Schwabe, and Pedro Szekely. Kgtk: a toolkit for large knowledge graph manipulation and analysis. In *International Semantic Web Conference*, pages 278–293. Springer, 2020. `doi:10.1007/978-3-030-62466-8_18`.

44 Filip Ilievski, Pedro Szekely, and Bin Zhang. Cskg: The commonsense knowledge graph. In *Extended Semantic Web Conference (ESWC)*, 2021. `doi:10.1007/978-3-030-77385-4_41`.

45 Prateek Jain, Pascal Hitzler, Amit P Sheth, Kunal Verma, and Peter Z Yeh. Ontology alignment for linked open data. In *International semantic web conference*, pages 402–417. Springer, 2010. `doi:10.1007/978-3-642-17746-0_26`.

46 Henry Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):93–104, 2022. `doi:10.1609/aimag.v43i1.19122`.

47 Rick Kazman, Mark Klein, and Paul Clements. *ATAM: Method for architecture evaluation*. Carnegie Mellon University, Software Engineering Institute Pittsburgh, PA, 2000.

48 Elisa F Kendall and Deborah L McGuinness. *Ontology engineering*. Morgan & Claypool Publishers, 2019. `doi:10.2200/S00834ED1V01Y201802WBE018`.

49 Vijay Khatri and Carol V Brown. Designing data governance. *Communications of the ACM*, 53(1):148–152, 2010. `doi:10.1145/1629175.1629210`.

50 Gongjin Lan, Ting Liu, Xu Wang, Xueli Pan, and Zhisheng Huang. A semantic web technology index. *Scientific reports*, 12(1):3672, 2022.

51 Doug Lenat and Gary Marcus. Getting from generative ai to trustworthy ai: What llms might learn from cyc. *arXiv preprint arXiv:2308.04445*, 2023. `doi:10.48550/arXiv.2308.04445`.

52   Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019. URL: https://proceedings.mlsys.org/book/282.pdf.

53   Bo Li, Peng Qi, Bo Liu, Shuai Di, Jingen Liu, Jiquan Pei, Jinfeng Yi, and Bowen Zhou. Trustworthy ai: From principles to practices. *ACM Computing Surveys*, 55(9):1–46, 2023. doi:10.1145/3555803.

54   Sebastian Lobentanzer, Patrick Aloy, Jan Baumbach, Balazs Bohar, Vincent J Carey, Pornpimol Charoentong, Katharina Danhauser, Tunca Doğan, Johann Dreo, Ian Dunham, et al. Democratizing knowledge representation with biocypher. *Nature Biotechnology*, pages 1–4, 2023.

55   Sebastian Lobentanzer, Patrick Aloy, Jan Baumbach, Balazs Bohar, Vincent J Carey, Pornpimol Charoentong, Katharina Danhauser, Tunca Doğan, Johann Dreo, Ian Dunham, et al. Democratizing knowledge representation with biocypher. *Nature Biotechnology*, 41(8):1056–1059, 2023.

56   Elisa Y Nakagawa, Fabiano C Ferrari, Mariela MF Sasaki, and José C Maldonado. An aspect-oriented reference architecture for software engineering environments. *Journal of Systems and Software*, 84(10):1670–1684, 2011. doi:10.1016/j.jss.2011.04.052.

57   Allen Newell, John Calman Shaw, and Herbert A Simon. Elements of a theory of human problem solving. *Psychological review*, 65(3):151, 1958.

58   Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

59   Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019. doi:10.1145/3329781.3332266.

60   Marc Gallofré Ocaña, Tareq Al-Moslmi, and A. Opdahl. Data privacy in journalistic knowledge platforms. In *International Conference on Information and Knowledge Management*, 2020. URL: https://ceur-ws.org/Vol-2699/paper44.pdf.

61   Marc Gallofré Ocaña and Andreas L Opdahl. A software reference architecture for journalistic knowledge platforms. *Knowledge-Based Systems*, 276:110750, 2023. doi:10.1016/j.knosys.2023.110750.

62   Lorena Otero-Cerdeira, Francisco J Rodríguez-Martínez, and Alma Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015. doi:10.1016/j.eswa.2014.08.032.

63   Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017. doi:10.3233/SW-160218.

64   Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, nov 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-1250.

65   Alessandro Piscopo and Elena Simperl. Who models the world? collaborative ontology creation and user roles in wikidata. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–18, 2018. doi:10.1145/3274410.

66   María Poveda-Villalón, Alba Fernández-Izquierdo, Mariano Fernández-López, and Raúl García-Castro. Lot: An industrial oriented ontology engineering framework. *Engineering Applications of Artificial Intelligence*, 111:104755, 2022. doi:10.1016/j.engappai.2022.104755.

67   Alun Preece. Evaluating verification and validation methods in knowledge engineering. In *Industrial knowledge management: A micro-level approach*, pages 91–104. Springer, 2001.

68   Jason Priem, Heather Piwowar, and Richard Orr. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint arXiv:2205.01833*, 2022. doi:10.48550/arXiv.2205.01833.

69   Qinjun Qiu, Zhong Xie, Liang Wu, and Liufeng Tao. Dictionary-based automated information extraction from geological documents using a deep learning algorithm. *Earth and Space Science*, 7(3):e2019EA000993, 2020.

70   F.P. Ramsey. Knowledge. In *F.P. Ramsey: Philosophical Papers*, pages 110–111. Cambridge University Press, 1929.

71   Franck Ravat and Yan Zhao. Data lakes: Trends and perspectives. In *Database and Expert Systems Applications: 30th International Conference, DEXA 2019, Linz, Austria, August 26–29, 2019, Proceedings, Part I 30*, pages 304–313. Springer, 2019. doi:10.1007/978-3-030-27615-7_23.

72   Marta Sabou, Majlinda Llugiqi, Fajar J Ekaputra, Laura Waltersdorfer, and Stefani Tsaneva. Knowledge engineering in the age of neurosymbolic systems. *Neurosymbolic AI Journal (under review)*, 2024.

73   Mahdi Sahlabadi, Ravie Chandren Muniyandi, Zarina Shukur, and Faizan Qamar. Lightweight software architecture evaluation for industry: A comprehensive review. *Sensors*, 22(3):1252, 2022. doi:10.3390/s22031252.

74   Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016. doi:10.1007/s41060-016-0027-9.

75   Michael Schade. How ChatGPT and Our Language Models Are Developed, 2023. URL: https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-language-models-are-developed.

76   Phillip Schneider, Tim Schopf, Juraj Vladika, Mikhail Galkin, Elena Paslaru Bontas Simperl, and Florian Matthes. A decade of knowledge graphs in natural language processing: A survey. In *AACL*, 2022. URL: https://api.semanticscholar.org/CorpusID:252683270.

77 August Th Schreiber, Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Nigel Shadbolt, Robert de Hoog, Walter Van de Velde, and Bob Wielinga. *Knowledge engineering and management: the CommonKADS methodology*. MIT press, 2000.

78 Kartik Shenoy, Filip Ilievski, Daniel Garijo, Daniel Schwabe, and Pedro Szekely. A study of the quality of wikidata. *Journal of Web Semantics*, 2021. `doi:10.1016/j.websem.2021.100679`.

79 Umutcan Simsek, Elias Kärle, Kevin Angele, Elwin Huaman, Juliette Opdenplatz, Dennis Sommer, Jürgen Umbrich, and Dieter Fensel. A knowledge graph perspective on knowledge engineering. *SN Computer Science*, 4(1):16, 2022. `doi:10.1007/s42979-022-01429-x`.

80 Dezhao Song, Frank Schilder, Shai Hertz, Giuseppe Saltini, Charese Smiley, Phani Nivarthi, Oren Hazai, Dudi Landau, Mike Zaharkin, Tom Zielund, et al. Building and querying an enterprise knowledge graph. *IEEE Transactions on Services Computing*, 12(3):356–369, 2017. `doi:10.1109/TSC.2017.2711600`.

81 Miroslaw Staron and Miroslaw Staron. Autosar (automotive open system architecture). *Automotive Software Architectures: An Introduction*, pages 97–136, 2021.

82 Monika Steidl, Michael Felderer, and Rudolf Ramler. The pipeline for the continuous development of artificial intelligence models—current state of research and practice. *Journal of Systems and Software*, 199:111615, 2023. `doi:10.1016/j.jss.2023.111615`.

83 Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. The neon methodology for ontology engineering. In *Ontology engineering in a networked world*, pages 9–34. Springer, 2011. `doi:10.1007/978-3-642-24794-1_2`.

84 Gytė Tamašauskaitė and Paul Groth. Defining a knowledge graph development process through a systematic review. *ACM Transactions on Software Engineering and Methodology*, 2022. `doi:10.1145/3522586`.

85 Richard N Taylor, Nenad Medvidović, and Eric M Dashofy. *Software architecture: foundations, theory, and practice*. John Wiley & Sons, Inc., 2010.

86 WDQS Search Team. WDQS Backend Alternatives: The Process, Details and Results. `https://www.wikidata.org/wiki/File:WDQS_Backend_Alternatives_working_paper.pdf`, 2022. Accessed: 2022-08-15.

87 Karim Tharani. Much more than a mere technology: A systematic review of wikidata in libraries. *The Journal of Academic Librarianship*, 47(2):102326, 2021.

88 Katherine Thornton, Harold Solbrig, Gregory S Stupp, Jose Emilio Labra Gayo, Daniel Mietchen, Eric Prud'Hommeaux, and Andra Waagmeester. Using shape expressions (shex) to share rdf data models and to guide curation with rigorous validation. In *The Semantic Web: 16th International Conference, ESWC 2019, Portorož,*

Slovenia, June 2–6, 2019, Proceedings 16, pages 606–620. Springer, 2019. `doi:10.1007/978-3-030-21348-0_39`.

89 Ilaria Tiddi, Victor De Boer, Stefan Schlobach, and André Meyer-Vitali. Knowledge engineering for hybrid intelligence. In *Proceedings of the 12th Knowledge Capture Conference 2023*, pages 75–82, 2023. `doi:10.1145/3587259.3627541`.

90 Riccardo Tommasini, Filip Ilievski, and Thilini Wijesiriwardene. The internet meme knowledge graph. In *ESWC*, 2023. `doi:10.1007/978-3-031-33455-9_21`.

91 Michael van Bekkum, Maaike de Boer, Frank van Harmelen, André Meyer-Vitali, and Annette ten Teije. Modular design patterns for hybrid learning and reasoning systems: a taxonomy, patterns and use cases. *Applied Intelligence*, 51(9):6528–6546, 2021. `doi:10.1007/s10489-021-02394-3`.

92 Frank Van Harmelen and Annette Ten Teije. A boxology of design patterns for hybrid learning and reasoning systems. *Journal of Web Engineering*, 18(1-3):97–123, 2019. `doi:10.13052/jwe1540-9589.18133`.

93 Laura Waltersdorfer, Anna Breit, Fajar J Ekaputra, Marta Sabou, Andreas Ekelhart, Andreea Iana, Heiko Paulheim, Jan Portisch, Artem Revenko, Annette ten Teije, et al. Semantic web machine learning systems: An analysis of system patterns. In *Compendium of Neurosymbolic Artificial Intelligence*, pages 77–99. IOS Press, 2023. `doi:10.3233/FAIA230136`.

94 Bob J Wielinga, A Th Schreiber, and Jost A Breuker. Kads: A modelling approach to knowledge engineering. *Knowledge acquisition*, 4(1):5–53, 1992.

95 Hans Friedrich Witschel, Charuta Pande, Andreas Martin, Emanuele Laurenzi, and Knut Hinkelmann. Visualization of patterns for hybrid learning and reasoning with human involvement. In *New Trends in Business Information Systems and Technology: Digital Innovation and Digital Business Transformation*, pages 193–204. Springer, 2020.

96 Dong Yang, Lixin Tong, Yan Ye, and Hongwei Wu. Applying commonkads and semantic web technologies to ontology-based e-government knowledge systems. In *The Semantic Web–ASWC 2006: First Asian Semantic Web Conference, Beijing, China, September 3-7, 2006. Proceedings 1*, pages 336–342. Springer, 2006. `doi:10.1007/11836025_34`.

97 Yixiang Yao, Pedro Szekely, and Jay Pujara. Extensible and scalable entity resolution for financial datasets using rltk. In *Proceedings of the 5th Workshop on Data Science for Macro-modeling with Financial and Economic Datasets*, pages 1–1, 2019. `doi:10.1145/3336499.3338008`.

98 Nasser Zalmout, Chenwei Zhang, Xian Li, Yan Liang, and Xin Luna Dong. All you need to know to build a product knowledge graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 4090–4091, 2021. `doi:10.1145/3447548.3470825`.