







Exploring Enterprise Knowledge Graphs: A Use Case in Software Engineering

Marta Sabou¹ , Fajar J. Ekaputra¹ , Tudor Ionescu², Juergen Musil¹ ,
Daniel Schall², Kevin Haller¹ , Armin Friedl¹, and Stefan Biffl¹

¹ Technical University of Vienna, Vienna, Austria

{marta.sabou,fajar.ekaputra,juergen.musil,kevin.haller,armin.friedl,
stefan.biffl}@tuwien.ac.at

² Siemens AG Vienna, Vienna, Austria

{tudor.ionescu,daniel.schall}@siemens.com

Abstract. When reusing software architectural knowledge, such as design patterns or design decisions, software architects need support for exploring architectural knowledge collections, e.g., for finding related items. While semantic-based architectural knowledge management tools are limited to supporting lookup-based tasks through faceted search and fall short of enabling exploration, semantic-based exploratory search systems primarily focus on web-scale knowledge graphs without having been adapted to enterprise-scale knowledge graphs (EKG). We investigate how and to what extent exploratory search can be supported on EKGs of architectural knowledge. We propose an approach for building exploratory search systems on EKGs and demonstrate its use within Siemens, which resulted in the STAR system used in practice by 200–300 software architects. We found that the EKG’s ontology allows making previously implicit organisational knowledge explicit and this knowledge informs the design of suitable relatedness metrics to support exploration. Yet, the performance of these metrics heavily depends on the characteristics of the EKG’s data. Therefore both statistical and user-based evaluations can be used to select the right metric before system implementation.

Keywords: Software engineering · Software architectural knowledge
Enterprise knowledge graph · Exploratory search

1 Introduction

In the area of software engineering, software documentation in general [9] and software architectural knowledge (AK) in particular [3] are important assets throughout the software engineering life-cycle. AK encompasses software design patterns, architecture descriptions, reference architecture models (i.e., the outcome of the software architecture design process) as well as the design decisions taken by the software architects during the design process itself [13]. AK has considerable value as it enables reusing already validated design solutions. Therefore,

its management is crucial as it affects the effectiveness of architecture decision making processes that steer the evolution of a software platform.

Therefore, researchers investigate for more than a decade approaches and systems to support software engineers and architects with AK management [3]. Some of these approaches are based on Semantic Web techniques, with ontology-based semantic data enabling intuitive, graphical navigation of AK [4, 14] or acting as backbones for wiki-based systems offering browsing, faceted-search and querying capabilities [5–7, 10, 15, 22]. Some ontology enabled faceted search systems were shown to improve the effectiveness and efficiency of AK retrieval [5, 15]. Yet, recent studies indicate that architects still mostly rely on ad-hoc decisions to address architectural quality attributes (e.g., scalability) and that they have limited awareness of available alternatives when making architectural decisions to address quality attributes [1, 2]. Therefore, a challenge is providing search mechanisms that go beyond information lookup, such as achievable with faceted-search, and support software architects in *exploring the space of available AK elements, especially plausible alternatives*. This challenge also exists at Siemens AG, a large organization with a key focus on software engineering. Siemens software architects need support to intuitively search for and explore architectural knowledge but are currently poorly supported in this task by legacy technologies which rely primarily on databases and wikis.

A promising approach to support software architects who need to sieve through and make sense of AK repositories is *exploratory search* which enables open-ended, weakly-defined information seeking tasks such as learning and sense-making [23]. Previous evidence shows that semantic structures can support exploration, e.g., by improving search performance and decreasing frustration levels [8]. And indeed, several semantics-based exploratory search systems were proposed [16]. Typically, these systems support exploration by presenting new knowledge derived from the underlying semantic structure *algorithmically* (e.g., through relatedness and similarity metrics) and they primarily focus on enabling the exploration of large knowledge graphs (e.g., DBpedia) [16].

Although enterprises increasingly create their own *Enterprise Knowledge Graphs (EKG)* to represent expert knowledge [19], adapting exploratory search to such graphs has received limited attention. Expert knowledge has been characterised as domain specific and constrained in its scope [20] as well as highly structured, detailed and interconnected [11]. Therefore we define EKGs as (1) *highly specific*, often encoding a single, narrowly defined domain; (2) comparatively *small* (compared to e.g., general knowledge graphs such as DBpedia) and (3) *deeply structured with detailed interconnections* between their concepts.

Our goal is to implement an exploratory search system of architectural knowledge at Siemens. Therefore, we investigate the following research questions:

RQ1: How to implement exploratory search on EKGs? What is a suitable approach? What main steps need to be followed?

RQ2: How to identify suitable relatedness metrics? Our hypothesis is that relatedness metrics supporting exploration could be derived based on

domain knowledge from the EKG’s ontology. We investigate methods to select the most suitable metric that minimise the need for expensive user studies.

RQ3: Are explanations derived from the EKG helpful? Based on [17], our hypothesis is that relatedness explanations which can be derived based on domain-specific relatedness metrics are helpful to support exploration.

We investigate these research questions in the context of Siemens. Since previous studies have positively evaluated semantics-based faceted-search AK management systems [5, 15], we designed, implemented and deployed the Semantic Search for Architectural Knowledge (STAR) prototype, a system that enables faceted search of AK. Additionally, we extended this system with *exploratory search* that takes advantage of the underlying EKG. STAR is novel in the AK management system landscape because it goes beyond faceted search based on the ontology structure, such as in [5–7, 10, 15, 22], and introduces exploratory search by recommending related AK, derived from the underlying EKG by semantic relatedness metrics. From the perspective of exploratory search research, we propose an approach for implementing this paradigm on an EKG and report on its concrete use in practice. STAR has been integrated into the current use case setting in a way that augments rather than replaces legacy solutions, and, as a result, is being used for accessing AK by cca. 200–300 Siemens software architects. Although we report work in the context of Siemens, our approach and lessons learned are of interest to other companies that need exploratory AK management, or, more broadly, enterprises with an EKG which could enable exploratory search.

Next, we describe the Siemens use case (Sect. 2) and present our approach to implement exploratory search (Sect. 3) as well as its main steps: the creation of the STAR EKG (Sect. 4), the definition and evaluation of relatedness metrics (Sects. 5, 6) and the actual system implementation (Sect. 7). We overview related work (Sect. 8) and conclude with lessons learned, benefits and challenges (Sect. 9).

2 Use Case

Siemens AG is the largest manufacturing and electronics company in Europe. Its central research and development unit, Siemens Corporate Technology, shapes innovation activities in the company and provides solutions to the company’s Business Units, e.g., through the creation of software by some 4800 software engineers across the globe.

A portion of this workforce (i.e., software and system architects, technical project managers, senior developers, research group leaders) focuses on designing software architecture. On a daily basis, these employees take several architectural design decisions [13] carefully selecting the most appropriate architectural elements (e.g., design patterns and tactics) to employ and weighing the effects of these elements on the desired architectural qualities of the created system (e.g., scalability, reliability, accessibility). In their decisions, they also consider

specific characteristics and requirements of the domain for which the software is designed (e.g., cloud computing, smart cities, cyber-physical systems).

A key task in designing software architecture is finding and reusing AK. For example, an architect wants to design a system that has the *qualities* of being *configurable* and *expressive*. He knows that the *façade* pattern enables designing systems with these characteristics, but would like to know about other patterns with similar effects, which he can consider for reuse. This reuse of already validated architectural solutions not only makes the design process faster but also ensures a better outcome, i.e., a well-designed software product. To enable such knowledge reuse, architectural knowledge is documented and during the years, this activity has led to several collections of semi-structured documents describing AK elements and their use within Siemens projects. The corpus of our use case contained about 600 AK entities (e.g., design patterns and tactics).

Siemens employs a wide range of advanced methods and platforms to manage AK, relying primarily on wiki-based systems and on databases (see Sect. 7). The search functionalities of these approaches rely primarily on organizational metadata such as the project where the AK was applied or the group/person that developed it. It is *difficult to search for AK based on meaningful features* such as e.g., its effect on architectural qualities or the (fine-grained) relations between architectural elements. There is therefore a need for sophisticated AK search capabilities to intuitively search and easily find AK. Going beyond simply organizing and accessing AK, the search system should support discovering new knowledge in a serendipitous fashion, i.e., learning something relevant but unexpected. One instance of serendipitous learning is finding related AK elements: for example, two design patterns might be considered as related when they have comparable effects on architectural qualities. We implement an exploratory search system in this context as described next.

3 Implementing Exploratory Search: Approach

Our approach for implementing exploratory search on an EKG, assumes a green-field context where the EKG itself needs to be created. Recently, knowledge graphs were defined as “*a set of interconnected typed entities and their attributes*”, where an ontology defines the vocabulary of the graph [19]. Therefore, our approach covers both ontology and EKG construction among other steps:

Create ontology to capture the relevant domain knowledge. While any of the existing ontology creation methodologies can be followed, in an enterprise setting at least the following three stages apply: **1. Scoping** clarifies the purpose of the system as well as the use cases and competency questions to be supported; **2. Definition** creates the first version of the ontology taking into account organization internal and external data sources; **3. Refinement** of the ontology with respect to the use case, data to be represented in the EKG and actual usage during system implementation.

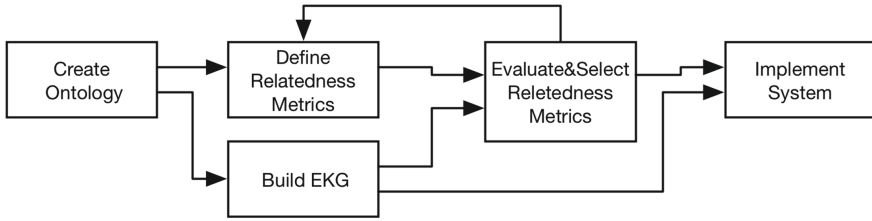


Fig. 1. STAR approach for creating an exploratory search system.

Build EKG. To create the EKG entities based on the defined ontology, several methods can be used to transfer relevant data into a semantic representation.

Define relatedness metrics. Taking the view of exploratory search as recommending related entities, at this step suitable relatedness metrics are define, for example, by relying on domain specific heuristics.

Evaluate and select relatedness metrics. Before implementing relatedness metrics in a concrete exploratory search system, it is vital to test these. A combination of statistical comparison techniques and user studies is recommended. Evaluation results can be taken into account to refine the metrics.

Implement system with the selected metric(s).

The next sections describe the application of the approach within our use case.

4 Creating the STAR EKG of Architectural Knowledge

Creating the STAR EKG entailed ontology construction and its population. To construct the STAR ontology, in an initial **scoping** workshop system purpose, use cases, and competency questions were clarified. To create an ontology that could play the role of a common denominator on architectural knowledge terminology within the company, it was decided to build on widely adopted AK models and extend these with domain and organizational specific concepts. The main standard for documenting software architectures is the ISO/IEC/IEEE 42010:2010 [12]. It comprises the meta models used to derive standard-conform viewpoints, architecture frameworks and architecture description languages.

During ontology **definition**, the project team, including a knowledge engineer and a software engineering researcher, identified a list of relevant concepts based on: (1) the datamodel of the legacy database for storing AK; (2) ISO/IEC/IEEE 42010 and (3) other AK ontologies proposed in the literature [7, 22]. Next, the relations that hold between these concepts were identified, thus making explicit the information encoded implicitly in the database schema. To **refine** the ontology, several iterations were performed with Siemens partners to validate the relevance and usefulness of these concepts for the use case. As part of these iterations, the AK corpus was transformed into the ontology in order to check coverage. A stakeholder workshop closed the development process, where

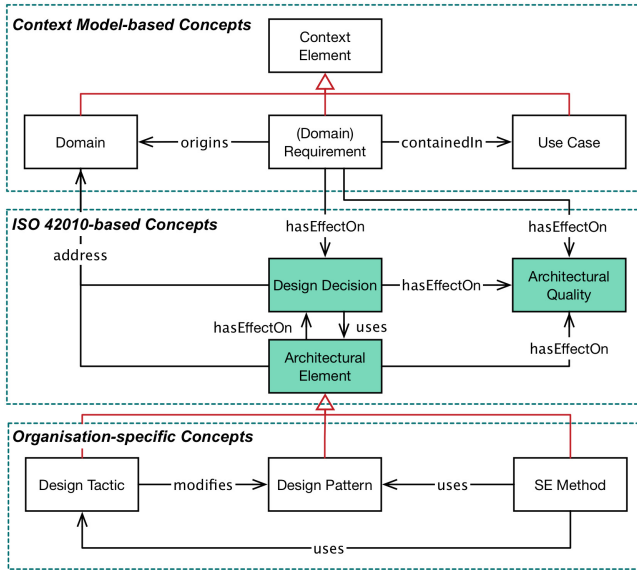


Fig. 2. STAR ontology: main concepts and their relations.

the first draft version of the ontology was approved. Smaller, implementation specific adaptations happened during the ontology's use in system development.

The STAR EKG consists of the STAR ontology¹ (Fig. 2) with 20 concepts, 30 properties and around 1000 entities that are instances of the ontology classes and were extracted from legacy databases as discussed in Sect. 7. Because design decisions play a key role in the work of software architects, these are modeled as a core concept in the STAR ontology, i.e., *DesignDecision* (90 entities). They have effects on *ArchitecturalQualities* (95), which refer to desired characteristics of the designed system such as accuracy, accessibility, autonomy or usability. Design decisions rely for their realization on (i.e., use) *ArchitectureElements* (AE). These three concepts and their relations emerge as a core modeling pattern in the software architecture domain. Types of architectural elements interesting for the context of Siemens are *DesignPatterns* (301), *DesignTactics* (137) and *SEMethods* (43). The design process and its elements are affected by and address a set of *ContextElements*. Relevant for Siemens are context elements such as *Domain* (107), *UseCase* and *Requirement*. Finally, a set of concepts have been introduced to capture the provenance of the information (such as the *Author* that added it) or relevance for a certain *Project* (18), *ProjectRole* (e.g., test architect, requirements engineer) or *ProjectPhase* (e.g., design, testing).

¹ The STAR ontology is available at: <http://data.ifs.tuwien.ac.at/star/>.

5 Defining Relatedness Metrics for Exploratory Search

We achieve exploratory search by recommending related architectural elements (e.g., design patterns) based on the STAR EKG. One benefit of the ontological representation of the core concepts and relations of interest to the company was that it provided intuitions about heuristics to design metrics for computing the relatedness of two AEs. We designed two such metrics, as discussed next.

AEs have either negative or positive effects on architectural qualities. For example, the *façade* design pattern has a positive effect on *flexibility*, but negatively affects *accessibility*. Therefore, one domain heuristic is that AEs are related if they have similar effects on AQs. This heuristic lead to the Rel_{AQ} metric which is directly proportionate with the number of AQs addressed by both elements ($Q(ae_1, ae_2)$) and with the number of those AQs which they affect in the same way ($Q_a(ae_1, ae_2)$). Rel_{AQ} is indirectly proportionate with the number of qualities affected in different ways by two elements ($Q_{da}(ae_1, ae_2)$).

$$Rel_{AQ}(ae_1, ae_2) = Q(ae_1, ae_2) * \frac{|Q_a(ae_1, ae_2)| + 1}{|Q_{da}(ae_1, ae_2)| + 1}$$

Our second metric (Rel_{DOM}) relies on information theory to capture the relation between architectural elements and the domains that they address. For each domain (e.g., “smart cities”), its information content is computed ($IC(d)$). A domain addressed by many elements has a lower IC than a domain addressed only by a few elements. Rel_{DOM} is the average information content of the domains that both architectural elements address ($D(ae_1, ae_2)$).

$$IC(d) = -\log \frac{\text{count}(ae_i.\text{address}.d)}{\sum_{i,j} \text{count}(ae_i.\text{address}.d_j)}; Rel_{DOM}(ae_1, ae_2) = \frac{\sum_{d_i \in D} IC(d_i)}{|D(ae_1, ae_2)|}$$

Generating explanations. For Rel_{AQ} , explanations of AE relatedness list the AQs on which the two AEs have the same effect and those AQs which are affected differently. For Rel_{DOM} , we list the domains that both elements address.

6 Evaluation of Relatedness Metrics

After designing the relatedness metrics we perform an evaluation to select the metric to be integrated into the exploratory search system. Our approach is to first perform a comparative evaluation of the overall behaviour of the metrics over the entire EKG, and based on the findings to select the most promising metric (Sect. 6.1) which is then evaluated in a user study (Sect. 6.2).

6.1 Comparative Evaluation of Relatedness Metrics

We use statistical methods to compare the overall behaviour of the two metrics over the entire EKG. The box-plots and the relative frequency histograms displayed in Fig. 3 show that Rel_{AQ} leads to a variety of values and has a good discriminative power among related architectural elements while Rel_{DOM} returns similar values for most element pairs, thus only weakly supporting relatedness

identification. Indeed, although over 100 domains are defined in the EKG, 68% of the architectural elements address only the domain “abstract” and therefore the applicability of the information content inspired metric on this data is not optimal. Therefore, our user evaluation focuses on Rel_{AQ} alone.

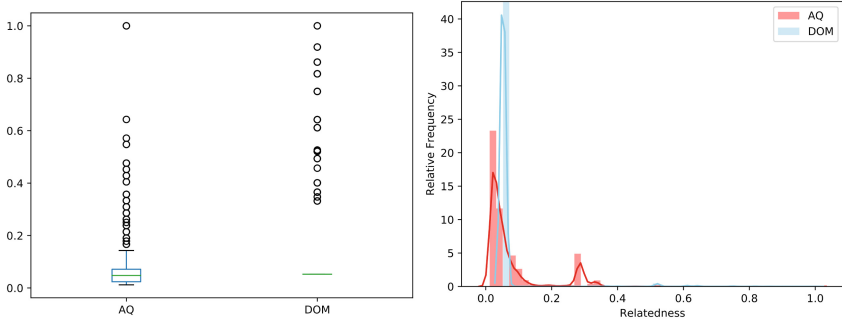


Fig. 3. Comparison of the overall behaviour of the relatedness metrics.

6.2 User Based Evaluation

The task of determining architectural element relatedness received limited attention in the literature so far. Therefore, the main goals of the user based evaluations were: (Q1) to assess how difficult this task is for human raters; (Q2) to determine the quality of ratings derived by the relatedness metric Rel_{AQ} and (Q3) to investigate whether explanations derived from the EKG support users in understanding relatedness among architectural elements.

Experimental Data. We focus the evaluation on design patterns as the largest group of architectural elements in our collection. The relatedness metric supports an exploratory search interface, in which only patterns related above a certain threshold are shown. Our interest therefore is to evaluate how well the metric works for highly related pattern pairs. We computed the weighted Rel_{AQ} values for all patterns in the dataset and chose five patterns for which relatedness pairs ranked highest (e.g., façade, proxy). For each pattern, we selected the top 5 related patterns recommended by Rel_{AQ} . The evaluation dataset therefore consisted in 25 pattern pairs and the explanations for deriving their relatedness (e.g., the qualities on which the patterns had/did not have the same effect).

The study population consisted in 8 participants (6 from Siemens, 2 from the university), all with a software engineering background and with education ranging from undergraduate (2), to graduate (1), doctoral (2) and post-doc (3) levels. Industrial experience with software architecture ranged from 1 year (2), to 5–7 years (4) and 10 years (2). In terms of experience with software design patterns the population included 2 experts, 5 advanced and 1 intermediate participant. Participants were divided in two groups, Gr1 and Gr2.

Study Task. Participants were shown pattern pairs, their descriptions and (for half of the dataset) an explanation of how their relatedness was derived.

They were asked to rank the pairs relatedness on a 5-point scale (1 completely unrelated; 5 very related) considering a broad notion of relatedness that one would expect in the context of an exploratory search system. Gr1 evaluated the first half of the pair set without explanation and the second half with explanation, while Gr2 did the opposite. A post-study survey collected background information and opinions about the study task as qualitative data. Based on the collected data the following *conclusions* were drawn.

Q1: Difficulty of Pattern Relatedness Judgement. Based on the data collected through the survey to the question *How challenging was the task of comparing the patterns?*, most participants considered the task of average difficulty (avg:3.25 on a scale from 1-very difficult to 5-very easy). Main issues encountered were: (1) the understanding of the patterns was hampered when the quality of pattern descriptions was suboptimal (e.g., too short, too generic); (2) it was difficult to judge how patterns relate without considering a concrete context such as a use case; (3) generally, the notion of pattern relatedness was perceived as challenging to quantify especially when patterns shared some common characteristics but differed in others (e.g., when patterns had: different intentions but similar realization/different realization but similar intention). Interpreting the actual rankings made by the participants in terms of Fleiss Kappa, we obtained an overall agreement of 34%, with a fixed marginal Kappa of 0.16. This shows only a slight agreement among human raters and suggests that the task is more challenging than perceived by our participants in the qualitative questionnaire.

Q2: Quality of Relatedness Ratings. In the post-study survey, participants rated the plausibility of the reviewed pattern pairs as moderate, avg. 2.15 on a scale from 1 (not plausible) to 4 (very plausible). This perception correlates with the interpretation of the experimental data. We considered as successful recommendation all pairs that had an average rating score of 3 or above derived from the 8 raters. We obtained an overall precision value of 52%, a reasonable result considering the inherent difficulty of the task. Looking at pair sets returned for each of the 5 patterns, precision at three (Prec@3) had an average value of 60% across the five pair sets corresponding to the seed patterns.

Q3. Usefulness of Relatedness Explanations. Based on the survey results, explanations of pattern relatedness as currently provided were considered moderately helpful, avg. 2 on a scale from 1 (not helpful) to 4 (very helpful). Main factors that lowered the usefulness of the explanations were: (1) too many architectural qualities were shown, thus increasing confusion; (2) AQs were not sufficiently clear, an explanatory sentence would have helped; (3) some AQs were perceived as not specific enough to be meaningful (e.g., maintainability can include both error handling and complexity management). As a result of these findings, explanations are not included in the current user interface.

Useful suggestions were collected for improving explanations, most of them (e.g., (2), (4) in the following list) revealing the need for more sophisticated semantic analysis on more detailed semantic annotations. Suggestions included: (1) provide more context information; (2) use more selective qualities that describe relations more appropriately; (3) reduce the number of the displayed

AQs; (4) explanations should be more detailed and based on a multitude of criteria (e.g., pattern domain, functionality, the problems solved); (5) recommend more refined relations between patterns (e.g., complements, refines, occurs with).

7 System Implementation

We implemented a faceted and exploratory search solution by *augmenting the legacy search solution* (top-half of Fig. 4) already in place at Siemens. In that solution, architectural knowledge was *acquired* through crawling mechanisms from both organization external and internal sources (e.g., project and domain-specific Wikis and databases of AK). As part of the acquisition phase, AK elements were enriched with information about experiences with their use in Siemens specific projects/domains and with their effect on system qualities. These metadata elements were added as key-value pairs and stored in a NoSQL database (i.e., MongoDB) as part of the *synthesis* stage of the system. A *keyword-based search interface* retrieves the architectural elements relevant for a given keyword and displays these in a search-engine style. The search interface proved helpful for users (i.e., software architects and engineers) to find relevant information about particular AKs and their application at Siemens. However, the solution lacked more detailed information about the relation between architectural elements and therefore provided limited support to explore the AK collection in more depth.

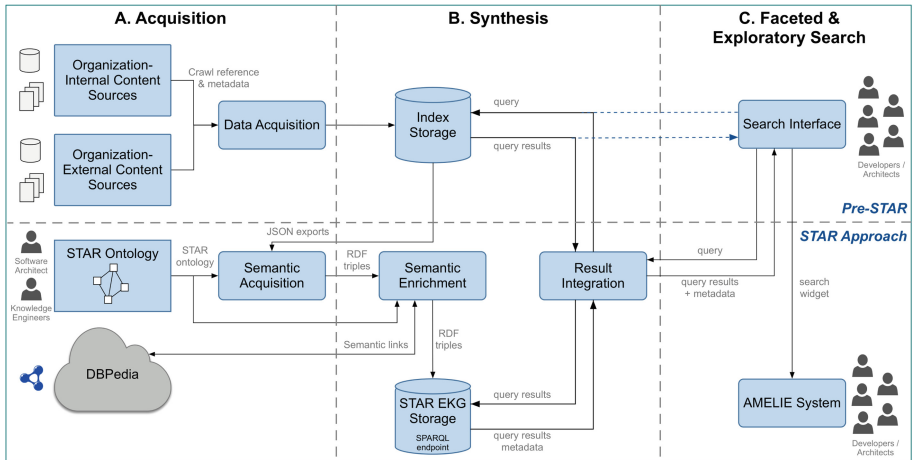


Fig. 4. System overview without (top-half) and with the STAR approach (lower-half).

We extended the legacy search solution with components for acquiring, storing and exploring semantic information (lower-half of Fig. 4). *Semantic Acquisition* focuses on ontology population. Ontology instances were created from the

information stored in MongoDB in a two-step process: (1) GSON and Apache Jena libraries were used to convert the JSON export from MongoDB into a temporary RDF file reflecting the databases key-value pair structure; (2) SPARQL construct queries were developed to transform the temporary RDF data into instances of the STAR ontology, thus creating the STAR EKG. The *Semantic Enrichment* component includes modules for data linking and computing additional semantic links between architectural elements. As some of the design patterns are not company specific, 66 were linked to the equivalent pages in DBpedia by defining linkage rules based on the *rdfs : label* values through the SILK framework. Furthermore, during semantic enrichment an “is related to” relation is added between the architectural element pairs related above a threshold according to *Rel_{AQ}*. The resulting EKG is stored in a Sesame repository.

The screenshot displays the STAR search system interface. At the top, there is a Siemens logo and a navigation menu with links for CT Collaboration Platform, Contact, Help, Browse, and Administration. Below the navigation, the page title is "Semantic Search for Architectural Knowledge" with sub-links for Attachments (0) and Comments (0). A search bar contains the term "rollback" and a "Search" button. Below the search bar, there is a search history path: "Search history > scalability > polling > performance > rollback". The search results are categorized under "All (2)" and "Design Patterns (2)". The first result is "Rollback (Design Patterns)" with a URL: "https://workspace.cee.siemens.com/content/00000102/Wiki/Rollback.aspx?masterview=nwacollabprint". A "show less" button is visible. The description of the result states: "A rollback is an operation which returns to a previously stored state. The underlying assumption is that (1) checkpoints are defined where the systems state is stored and (2) the system is able to restore to these checkpoints (recovery). Consequently once an error occurs the systems state is reverted to prior checkpoint [Hanmer 2007]pp 154-155. Relations Error Handling [Hanmer 2007]pp 154-155." Below the description, there is a "Rollback" section with a list of related terms and their counts: "has an effect on" (13), "is related to" (6), "is supported by" (2), "is used in" (1), "is added by" (2), "is relevant for" (1), "architectural quality" (13), "design pattern" (6), "design tactic" (2), "domain" (1), "author" (2), "project" (1), "performance" (13), "reliability" (6), "autonomy" (2), "fault tolerance" (2), "resilience" (2), "availability" (2), "dependability" (2), "survivability" (2), "maturity" (2), "robustness" (2), "recoverability" (2), "viability" (2), "safety" (2), "3-category logging" (2), "acknowledgement" (2), "checkpoint" (2), "checksum" (2), "failover" (2), "error handling" (2), "limit access" (2), "abstract" (2), "ionescu tudor" (2), "ct sad-sra" (2), "wiki" (2). A "Feedback" link is located at the bottom right of the result box. Below the result box, there is a "Return to Reference Point (Design Patterns)" section with a URL: "https://workspace.cee.siemens.com/content/00000102/Wiki/Return%20to%20Reference%20Point.aspx?masterview=nwacollabprint". The description states: "Return to ReferencePoint is used when an error occurs that can be recovered but for which the recovery does not provide appropriate Rollback or Rollforward. Execution is resumed by returning to a specific known state. That place might not have been in the execution path that led to the error but i...". A "show more" button is visible at the end of the description.

Fig. 5. STAR search system interface.

The *search mechanism* extends the search workflow of the legacy system. For a given search term, the MongoDB keyword search is used to locate all relevant resources. These results are used by the *Result Integration* component to retrieve relevant metadata from the EKG thus enhancing the MongoDB results

with additional semantic information. The *Search Interface* component contains tabs for all results as well as tabs dedicated to results of key types (e.g., Design Patterns, Architectural Qualities) as shown in Fig. 5. The detailed information about one search result (e.g., the “rollback” design pattern) contains the full-text description of that result. An extension of this legacy interface consists in displaying the semantic information relevant for a result and allowing its inspection through faceted-search. In this interface, the first column lists all relevant semantic relations for the current entity; the second column shows all semantic entity types relevant to the current entity (e.g., “architecture quality”) and how many instances of each type are semantically related to the result; the third column contains a collection of all entities (of all types) connected to the current result through semantic relations (i.e., a semantic entity-cloud).

The interface supports two modes of result inspection. Firstly, users get an overview of semantically relevant entities through the semantic entity cloud and can directly access those that they know and are interested in. Secondly, inspection can also be performed at schema element level. Clicking on a relation in the first column (e.g., “has an effect on”), will update the second column to contain only those entity types that are connected through this relation (e.g., “architectural quality”) and reduce the entities in the third column to the entities of the current type and related to the result with the selected semantic relation (i.e., all architectural qualities on which rollback has an effect on). Similarly, by selecting a concept in column two, only the relations that exist between that concept and the result are maintained and the semantic entity cloud is updated accordingly. Exploratory search of related entities is enabled by “is related to”, which connects the AEs that are related to the current entity according to Rel_{AQ} .

System Adoption. The STAR system is currently used by about 200–300 Siemens experts world-wide. The main user categories include software and system architects, technical project managers, senior developers, and research group leaders. The tool is also embedded as a search widget in Amelie - the state of the art Siemens tool for AK management. Amelie integrates various architectural artifacts from the entire development lifecycle of different software-intensive Siemens products. STAR helps by providing contextualized search capabilities depending on the current system view (i.e., what the user currently sees on the screen) by using the inputs and the navigation history of the user.

8 Related Work

We position our work in the landscape of *Architecture Knowledge Management* and *Exploratory Search* research. Semantic Web technologies have been used as a basis for several AK management tools [3]. Some tools exploit ontologies as a basis for more intuitive visualisation of AK, thus supporting the overview and analysis of AK collections [14] or, allowing a better understanding of the dependency between architectural design decisions [4]. Other tools use light-weight ontologies as backbones for wiki-based systems to enable browsing, faceted-search and querying: (1) for managing the documentation of service-oriented

architectures [10]; (2) for AK search [22]; (3) for finding AK through structured navigation and faceted search based on concept properties, and the execution of pre-defined SPARQL queries in ArchiMind [5–7]; or (4) to manage software design rationales [15]. As ontology-based faceted search was shown to improve the effectiveness and efficiency AK retrieval compared to file-based approaches [5, 15], our work also adopts a faceted-search approach but aims to advance the state of the art with an approach to implement exploratory search strategies.

Although *exploratory search* is a rather broad and evolving concept [18], in the Semantic Web area a common approach to exploration is to suggest entities related to the current search result [16]. The majority of efforts focus on exploration of large KGs, e.g., DBpedia or Freebase, with the adoption of this concept to more restrained enterprise KGs receiving limited attention. The evaluation of exploratory search systems is a critical issue firstly because user-based evaluations are costly to conduct and secondly, because novel evaluation approaches are sought that go beyond Information Retrieval style assessment [18]. In this landscape, we propose and illustrate the concrete application of an approach for adapting the exploratory search paradigm to EKGs. We propose using statistical evaluation and comparison of metrics to reduce the need for user studies.

9 Conclusions and Lessons Learned

Adequate AK management is a prerequisite to making software architecture design more efficient and effective. Yet, software architects are only weakly supported in their exploration of AK, both in general and in the context of Siemens in particular. In this paper, we investigate implementing exploratory search on an architectural knowledge EKG, thus going beyond current faceted-search tools and illustrating exploratory search on an enterprise-wide rather than generic KG. We propose an approach to build such systems and demonstrate its application at Siemens, resulting in a faceted and exploratory search system impacting the work of 200–300 software architects. The main conclusions to our research questions in the terms of lessons learned, benefits and challenges are as follows.

Concerning **RQ1**, related to *implementing exploratory search on EKGs*, we presented our overall approach and demonstrated its use in the context of Siemens. This approach considers the EKG’s ontology as key for informing the design of relatedness metrics and puts special emphasis on the evaluation and selection of these metrics before system implementation (see conclusions to RQ2).

During EKG creation, the *ontology engineering process* was beneficial in several ways. First, it led to the identification of a *core ontology* meaningful to a heterogeneous population of software architects within Siemens. To that end, alignment with other ontologies and standards in the domain *revealed key concepts and relations in the domain*. For example, the relation between design decisions, architectural qualities and architectural elements emerged as a recurring ontology design pattern in this domain. Second, in this phase we *made explicit semantic relations* which were only implicitly present in previous non-semantic

solutions (e.g., databases). Third, we *identified missing concepts* from the organizational specific model, such as the *Rationale* concept from ISO/IEC/IEEE 42010:2010 which would be a natural extension of the STAR ontology. A challenge was that ontology creation and data migration required several iterations and took *longer than expected* by the industry partner.

Best practices from system development were (1) the *lightweight integration of semantic technologies with legacy infrastructure*, which lead to lower development costs and facilitated technology acceptance; and (2) *including exploratory search elements into faceted search* systems which were known to be effective.

In line with our hypothesis for **RQ2**, the EKG's ontology captured essential domain knowledge beneficial for *informing the design of relatedness metrics*. Yet, a key challenge was that *metric performance depends on the EKG data characteristics*. A lesson learned is that distributional statistics can give insights into which metrics are viable on the data of a given EKG early into the system development process. In our use case, we excluded information content metrics based on such evaluations. User studies, although expensive, are beneficial to assess the actual performance of the metrics but also to provide further insights into the details of the exploration task. In our use case, through a user study, we found that the task of pattern relatedness assessment itself is challenging even for human raters, and at the same time collected feedback to better define the pattern relatedness notion. The study showed that the performance of relatedness identification with *RelAQ* was reasonable (Prec = 52%; Prec@3 = 60%). As the evaluation focuses on the relatedness metrics, it is limited in providing feedback on how the actual task of AK search has been improved. Such evaluation is left for future work.

Our hypothesis for **RQ3** was that relatedness explanations derived from the underlying EKG could support exploration. Yet we could not verify this hypothesis in the current setting with explanations being rated as only moderately helpful. Nevertheless, user feedback suggests possible improvements of explanations if these would be based on more detailed semantic annotations, which are missing from the current model. In fact, lack of more fine-grained semantic relations might have had a negative influence on the evaluation results.

Future work will focus on improving the AK collections quality through (semi-) automatic mechanisms for data acquisition and cleansing similar to [15, 21]. In particular, we aim to extract more refined semantic relations between architectural elements as a pre-requisite to a better performance of exploratory search. On the improved data, we will investigate more sophisticated relatedness metrics to support exploratory search, taking also into account the characteristics of the underlying EKG when defining them. More generically, we are interested in further investigating the notion of exploratory search in EKGs, including (1) testing our approach in other settings and (2) comparing the performance of domain-dependent with domain-independent relatedness metrics.

References

1. Ameller, D., Galster, M., Avgeriou, P., Franch, X.: A survey on quality attributes in service-based systems. *Softw. Q. J.* **24**(2), 271–299 (2016)
2. Bagheri, H., Garcia, J., Sadeghi, A., Malek, S., Medvidovic, N.: Software architectural principles in contemporary mobile software: from conception to practice. *J. Syst. Softw.* **119**, 31–44 (2016)
3. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Ali, M.: 10 years of software architecture knowledge management : practice and future. *J. Syst. Softw.* **116**, 191–205 (2016)
4. De Boer, R.C., Lago, P., Telea, A., van Vliet, H.: Ontology-driven visualization of architectural design decisions. In: Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA), pp. 51–60 (2009)
5. De Graaf, K.A., Liang, P., Tang, A., van Vliet, H.: How organisation of architecture documentation affects architectural knowledge retrieval. *Sci. Comput. Program.* **121**, 75–99 (2016)
6. De Graaf, K.A., Tang, A., Liang, P., Khalili, A.: Querying software architecture knowledge as linked open data. In: IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 272–277 (2017)
7. De Graaf, K.A., Tang, A., Liang, P., van Vliet, H.: Ontology-based software architecture documentation. In: Proceedings of Joint Working Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA), pp. 121–130 (2012)
8. Dimitrova, V., Lau, L., Thakker, D., Yang-Turner, F., Despotakis, D.: Exploring exploratory search: a user study with linked semantic data. In: International Workshop on Intelligent Exploration of Semantic Data, pp. 1–8 (2013)
9. Ding, W., Liang, P., Tang, A., van Vliet, H.: Knowledge-based approaches in software documentation: a systematic literature review. *Inf. Softw. Technol.* **56**(6), 545–567 (2014)
10. Happel, H.-J., Seedorf, S., Schader, M.: Ontology-enabled documentation of service-oriented architectures with ontobrowse semantic Wiki. In: PRIMM - Process Innovation for Enterprise Software, pp. 61–80 (2009)
11. Hoffman, R.R.: How can expertise be defined? Implications of research from cognitive psychology. In: Williams, R., Faulkner, W., Fleck, J. (eds.) *Exploring Expertise*, pp. 81–100. Palgrave Macmillan UK, London (1998). https://doi.org/10.1007/978-1-349-13693-3_4
12. ISO/IEC/IEEE: ISO/IEC/IEEE 42010:2010 systems and software engineering architecture description. Technical report. ISO/IEC/IEEE (2010)
13. Jansen, A., Netherlands, T., Bosch, J.: Software architecture as a set of architectural design decisions. In: Joint Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 109–120 (2005)
14. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (eds.) *QoSA 2006*. LNCS, vol. 4214, pp. 43–58. Springer, Heidelberg (2006). https://doi.org/10.1007/11921998_8
15. López, C., Codocedo, V., Astudillo, H., Cysneiros, L.M.: Bridging the gap between software architecture rationale formalisms and actual architecture documents: an ontology-driven approach. *Sci. Comput. Program.* **77**(1), 66–80 (2012)

16. Marie, N., Gandon, F.: Survey of linked data based exploration systems. In: International Conference on Intelligent Exploration of Semantic Data, pp. 66–77 (2014)
17. Marie, N., Gandon, F., Ribière, M., Rodio, F.: Discovery hub: on-the-fly linked data exploratory search. In: International Conference on Semantic Systems, pp. 17–24 (2013)
18. Palagi, E., Gandon, F., Giboin, A., Troncy, R.: A survey of definitions and models of exploratory search. In: Workshop on Exploratory Search and Interactive Data Analytics, pp. 3–8. ACM, New York (2017)
19. Pan, J.Z., Vetere, G., Gomez-Perez, J.M., Wu, H.: Exploiting Linked Data and Knowledge Graphs in Large Organisations, 1st edn. Springer, Heidelberg (2017). <https://doi.org/10.1007/978-3-319-45654-6>
20. Shanteau, J.: The psychology of experts: an alternative view. In: Wright, G., Bolger, F. (eds.) Expertise and Decision Support, pp. 11–23. Springer, Boston (1992). https://doi.org/10.1007/978-0-585-34290-0_2
21. Soliman, M., Galster, M., Riebisch, M.: Developing an ontology for architecture knowledge from developer communities. In: IEEE International Conference on Software Architecture (ICSA), pp. 89–92 (2017)
22. Tang, A., Liang, P., van Vliet, H.: Software architecture documentation: the road ahead. In: Working IEEE/IFIP Conference on Software Architecture, pp. 252–255 (2011)
23. White, R.W., Roth, R.A.: Exploratory Search: Beyond the Query Response Paradigm. Morgan & Claypool, San Rafael (2009)