Querying Software Architecture Knowledge as Linked Open Data

Klaas Andries de Graaf*, Antony Tang†, Peng Liang‡, and Ali Khalili*
*VU University Amsterdam, The Netherlands
†Swinburne University of Technology, Australia
‡Wuhan University, China

***Abstract*—It is difficult for software professionals to find all the architectural knowledge they need from architecture documentation, and this results in wasted time and mistakes in projects. This is the case even when architecture documentation is indexed by an ontology and stored in a semantic wiki. We present a prototype tool called AK-Finder which queries architectural knowledge stored in a semantic wiki as Linked Open Data using a SPARQL endpoint. Our tool retrieves knowledge from the semantic wiki and answers questions for architectural review, design, and development activities. The tool exemplifies how systems used in the software project lifecycle can integrate and improve access to architectural knowledge in a semantic wiki.**

***Index Terms*—Ontology-based software architecture documentation, Architectural knowledge retrieval, Semantic wiki, Ontology, SPARQL, Query, Endpoint, Linked Open Data**

I. Introduction

Documentation of software architecture serves three important purposes: it is used for education, system analysis, and stakeholder communication [1]. Architectural Knowledge (AK) is contained in SA documentation. AK can be defined as "*the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context/environment*" [2]. File-based documents, e.g., text and diagram files, are often used in industry practice to store AK [3].

Ontology-Based Software Architecture Documentation (OBSAD) makes use of ontologies for non-linear organisation of AK via classes and relationships, and this allows document writers to comprehensively organise AK and relationships between AK for the needs of document users. Recent studies provide evidence that the use of OBSAD improves the efficiency and effectiveness of AK recovery [4] and AK retrieval [5], compared to file-based AK organisation. Even though AK retrieval via OBSAD is an improvement over the use of file-based SA documentation, there is no easy and quick way to search and find AK without understanding the intricate relationships in an ontology.

In this paper we present a prototype tool, called AK-Finder, which retrieves knowledge from OBSAD stored in a semantic wiki. The AK-Finder (Architectural Knowledge Finder) tool provides a quick and easy way to query the knowledge in a semantic wiki through the use of predefined queries. Users can make use of these queries by supplying the contexts or parameters. The queries retrieve AK to support design and development activities and answer questions. In this work, we demonstrate the semantic queries by using architecture documentation review proposed by the Software Engineering Institute (SEI) in [6].

Our study illustrates how access to AK in a semantic wiki can be improved by querying the AK as Linked Open Data via SPARQL queries and AK-Finder. Our prototype tool AK-Finder supports software professionals in using and adapting SPARQL queries to retrieve AK during various software project activities. AK-Finder exemplifies how systems used in the software project lifecycle, e.g., modeling, CASE, and IDE tools, can integrate and implement access to the AK stored in a semantic wiki, and to AK in other knowledge bases that provide a SPARQL query endpoint.

Section II describes background and materials, Section III details on the AK-Finder tool, and Section IV and V report related work and conclusions respectively. Screenshots and queries of the AK-Finder tool can be found in Appendix A and on http://www.archimind.nl/ICSAToolAppendix.pdf.

II. Background and Materials

"**An ontology**" refers to a formal domain model in which concepts and relationships among concepts are described [4]. The classes and relationships in an ontology can be used for organising AK in SA documentation. Each distinct ontology class and relationship has properties and descriptions that explicitly define their meaning, allowing different AK users to interpret them consistently and unambiguously. Relationships in an ontology allow its users to see how AK instances are interrelated, e.g., "*requirement X is realized by component Y*", and thereby improves traceability between AK. The use of ontologies for SA documentation is described in more detail in [5], [7], [8].

Semantic wikis allows for presentation and navigation of classes and relationships in an ontology and provide features of traditional wiki-like systems, e.g., centralised access to and editing of documentation, versioning, and collaboration mechanisms. See http://www.archimind.nl/archimindLOD/ to access the **ArchiMind** semantic wiki that is queried by the AK-Finder tool presented in this paper. See https://github.com/kadevgraaf/archimind for the source code of ArchiMind.

ArchiMind is based on the OntoWiki tool [9] . OntoWiki offers web-based visualization, search, and management of (ontology and its instances in) knowledge bases. We made adaptations to version 0.9.5 of OntoWiki in order to optimize it for storage and retrieval of SA documentation.

File-based documentation content, e.g., from word processors and UML tools, and its layout are stored in **wikipages** using a WYSIWYG editor in ArchiMind [7]. 'Wikipage' is an ontology class and its instances store documentation content.

Our tool demonstrates the use of SPARQL queries (SPARQL is detailled below) on an **example SA document corpus** that is stored and semantically annotated in ArchiMind semantic wiki. The example SA document[1] describes the architecture of a social network system for software developers that answers questions of developers and analyses the development community. Please see [10] for more details.

The SA document was annotated using an **AK ontology**. During a study in [10] several authors of this paper used the AK needs expressed in architecture document review questions in [6] to build the AK ontology depicted in the figure on http://www.archimind.nl/archimindLOD/AKontologyLegend.png

SPARQL (SPARQL Protocol and RDF Query Language) is part of the semantic web technology stack and used to query knowledge bases containing ontology instances stored in RDF. The use of SPARQL queries is to some extent similar to how SQL (Structured Query Language) is used to query values, rows, and columns from tables in relational databases. However, SPARQL queries retrieve classes, class instances, relationships between classes, and instances of relationships that are stored in a graph-pattern using triples (subject, predicate, object). SPARQL queries are considerably better aligned with users' mental models of a domain. Unlike SQL queries which reflect the specific structure of a database and how the data is stored in tables within it, in SPARQL queries the focus is on user's understanding of the domain [11].

III. AK-Finder - Querying Architectural Knowledge as Linked Open Data

ArchiMind semantic wiki is derived from OntoWiki, which is a semantic wiki and a Linked Open Data server [12]. AK in the SA documentation in ArchiMind is accessible online as Linked Open Data (LOD). LOD technologies allow for data integration on the World Wide Web [13], i.e., online data published on different domains. Linking data using various W3C technologies, e.g., RDF and SPARQL, allows persons and machines to explore the data online as a web of data [14]. This is part of the semantic web paradigm [13].

The AK instances in ArchiMind are accessible to other online systems via a static URI containing the ontology URI and identifier of the AK instance. For example, a client-server architectural pattern stored in the SA documentation in ArchiMind semantic wiki can be directly accessed, listed, or embedded in other online systems used in the software development lifecycle, e.g., via HTML, browsers, or CURL, on URI http://www.archimind.nl/archimindLOD/index.php/view/r/sadocontology1.owl:Client_server_pattern.

[1]The SA document content can be read in Wikipages in ArchiMind.

The client-server pattern that is accessible via URI, can also be referenced and retrieved by other online systems via a SPARQL query sent to ArchiMind's SPARQL query endpoint. The URL of the SPARQL query endpoint in ArchiMind is http://www.archimind.nl/archimindLOD/index.php/sparql.

The SPARQL query endpoint in AchiMind allows online systems to query the AK in the SA documentation as Linked Open Data, and retrieve the query results outside of ArchiMind. The online systems can query and retrieve all the AK in ArchiMind, combine it with the AK or information in the online system, show the AK to users in their user interface. This supports software project activities that involve the use of AK without requiring professionals to navigate to and use ArchiMind as a separate tool to retrieve AK.

Our prototype tool, called AK-Finder, uses the SPARQL query endpoint in ArchiMind. The relationship between AK-Finder and ArchiMind is depicted in Figure 1. AK-Finder can be accessed at http://softcode.nl/AK-Finder/index.php (on a separate server from the installation of ArchiMind semantic wiki). See **https://github.com/kadevgraaf/AK-Finder** for the source code of AK-Finder on GitHub.

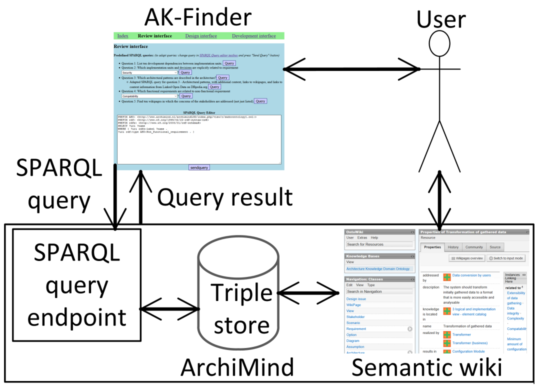


Fig. 1. Overview - Relationship between AK-Finder and ArchiMind

AK-Finder is written in PHP, HTML, and Javascript. These technologies were chosen because they are commonly used in web development, which allows many people to adapt, derive from, refactor, and host our code. To communicate with the SPARQL query endpoint in ArchiMind, AK-Finder makes uses of the sparqllib.php (made by Christopher Gutteridge).

Our tool can be extended to work with systems built in PHP and our tool can be embedded in webpages using an Iframe. For systems that are built in other programming languages there are other libraries to make use of ArchiMind's SPARQL query endpoint, e.g., RDFLIB for Python-based systems. Systems used in the software project lifecycle may include libraries such as sparqllib.php and RDFLIB and add functions to query the AK in ArchiMind, combine the retrieved AK with AK and other information in the online system, and thereby support users in activities involving retrieval of AK.

AK-Finder has three interfaces (separate webpages) that support three software project activities; architecture documentation review, architecture design activities, and development activities, which are detailed on in the next sections.

A large textarea inputbox in the three interfaces of AK-Finder allows users to edit and execute complete SPARQL queries directly. A screenshot of the review interface in Figure 2 shows the textarea for editing SPARQL queries. Lopez *et al.* argued in [15] that configuration of SPARQL queuries in a user interface is feasible.

A. Architectural Review Interface in AK-Finder

In a previous study [8] we applied the review approach in [6] by answering five SEI (Software Engineering Institute) architecture review questions from the semantic wiki. These review questions are part of a structured approach for reviewing architecture documentation proposed in an SEI technical note by Nord *et al.* in [6].

We implemented five SEI review questions in SPARL queries. In the review interface of our tool (see Figure 2) these SPARQL queries are sent to the SPARQL query endpoint in ArchiMind semantic wiki, and the endpoint returns the same query results as if they were executed in ArchiMind.

The SPARQL queries for answering the SEI review questions are shown in Appendix A and below. Other online systems, systems used by professionals in the software lifecycle or more generic systems that use SPARQL endpoints such as yasgui.org, can send the same queries in Appendix A to the endpoint in ArchiMind, and will get the same results.

Listing 1 shows how SEI Question 3 is answered. It needs the prefix "*rdf*" and "*AKO*" from Listing 2 in Appendix A. *?patterns* is a variable name which is used to store output of the query. The query matches triples (subject, predicate, object) in the knowledge base (the semantic wiki) by searching for any subject (variable *?patterns*) that has predicate *rdf:type* and object *AKO:Pattern*. The query retrieves all instances of class (or "*rdf:type*") 'Pattern' in the AK ontology (prefix "*AKO*", see Section II and http://www.archimind.nl/archimindLOD/AKontologyLegend.png) in ArchiMind.

```
SELECT ?patterns
WHERE {?patterns rdf:type AKO:Pattern}
```

Listing 1. SPARQL query for Question 3: Which architectural patterns are described in the architecture?

To improve usability, and eliminate the cost of rewriting SPARQL queries, we created buttons via which users of AK-Finder can execute the predefined SPARQL queries for architecture review (see Figure 2 in Appendix A). A software professional using the SPARQL queries does not need to know the intricate details of the AK ontology to formulate queries. This aims to reduce errors and save time when a professional is unfamiliar with an AK ontology or SA document content.

After a user presses one of the buttons to execute a predefined query, the results of the query, i.e., answers to SEI review questions, are shown in a table. Each column in the table lists query variables ("*?some_variable_name*" in the queries), e.g., a class instance or relationship name, and each row lists variables that are related to each other. Most results are URIs that include the name of AK, which users can click to see the AK in more detail in ArchiMind semantic wiki. The SPARQL query that was executed is shown in a large textarea which allows users to adapt the query or write a new query.

An architect working in AK-Finder, or a modeling or CASE tool which integrates AK-Finder, could, for example, perform SEI architecture documentation review Question 4 (see Appendix A): *Is the relationship between requirements documented and understood?* [6] by querying AK stored in ArchiMind Semantic wiki. The architect does not need to switch to ArchiMind to perform the architecture documentation review, but can retrieve related AK and perform review of a specific requirement in AK-Finder, or in a modelling or CASE tool which integrates AK-Finder. The end of Appendix A details on a comparison between AK retrieval using standard semantic wiki features and AK retrieval using AK-Finder, to indicate possible benefits of predefined queries in AK-Finder.

Moreover, users can adapt the topics of two predefined SPARQL queries in a dropdown box. This allows users to answer the same type of question with a different topic. For example, instead of answering SEI Question 4 to find the requirements related to 'Compatibility', a user can use the dropdown to retrieve requirements related to 'Maintainability'.

The last predefined query also retrieves a reference to a DBpedia entry that is related to a pattern stored in ArchiMind Next to referencing, AK from multiple sources queried and combined by adapting the SPARQL query endpoint on line 21 of query.php in the AK-Finder code (https://github.com/kadevgraaf/AK-Finder).

B. Interface for design activities

This interface shows how design activities can be supported by integrating querying of AK in tools in the software project lifecycle. A dropdown box select element allows users (e.g., architects and designers) to retrieve AK elements of a certain type (class in an ontology) and then retrieve additional information and relationships of the previously retrieved AK elements. The underlying code and queries can be adapted to dynamically list AK of a certain type or retrieved information about individual AK elements based on data or based on the selected interface in a design or CASE tool in the software project lifecycle.

One predefined query identifies requirements that are not realized by at least one element in the architecture. Another predefined query identifies which requirements are not yet addressed by a decision. Based on this identification a user can start work on defining requirements, making decisions, or interrelating the requirements and decisions to other AK.

C. Interface for development activities

This interface shows two examples of development tasks involving work on two components. Hyperlinks which trigger SPARQL queries allow users to visit related AK in ArchiMind and retrieve AK to show requirements and decisions related to the components. If no related requirements or decisions are retrieved then a message indicating an architectural rule violation is shown. The interface illustrates how integration of AK from ArchiMind to an IDE or case tool is possible - the code and queryies can be adapted to dynamically add tasks and generate queries based on data or the selected interface in an IDE or CASE tool, for example, to show AK about a component whilst working on code in that specific component.

IV. Related Work

In [15] and [16] Lopez *et al.* used SPARQL queries for the implementation of a tool for visualising, sharing, and reusing non-functional requirements and design rationale in the NDR ontology. One SPARQL query is listed in [16], which relates arguments to a decision. Our direct use of SPARQL queries in an external tool (AK-Finder) to retrieve AK stored in a semantic wiki as linked open data via a SPARQL query endpoint is different to the work in [15] and [16].

In [17] Tang *et al.* use inline queries in Semantic Media Wiki (SMW), to support traceability for requirements specifications and architecture design. The inline SMW-queries in [17] have SMW-specific syntax which differs from our use of generic SPARQL query syntax. Querying via SPARQL is proposed in [17], which is possible via an SMW-extension.

OntoWiki, on which ArchiMind is based, has been used in a study on Distributed Requirements Elicitation by Lohmann *et al.* in [18] and Semantification of Requirements Engineering in [12]. Their approach for semantifying requirement and publishing these as Linked Open Data is similar to our approach, however, we also provide a tool that makes use of the AK published as linked open data and we focus on other types of AK and describe queries to support architecture documentation review. The study in [18] demonstrates the suitability of OntoWiki, and derived tools, for software architecture knowledge management.

V. Conclusions and future work

AK retrieval can be difficult and error-prone, even from SA documentation in a semantic wiki with exploration and browsing features. The AK-Finder tool allows users to execute predefined knowledge queries. In this work, we use the SEI architecture review approach as an example to illustrate how AK-Finder works. Through AK-Finder, users can retrieve AK to evaluate architecture design. The tool hides the complexity of SPARQL and the ontology from everyday users.

Predefined SPARQL queries implemented in AK-Finder retrieve AK about various aspects of a software project. Specific search inputs such as dropdown boxes and a text input allow users to specify the information that is sought. By using such implementations they can easily query the knowledge base to understand the intricate details of an architecture design.

Our implementation of AK-Finder and use of the SPARQL query endpoint of ArchiMind semantic wiki demonstrates how tools used in the software project lifecycle can integrate and improve access to AK stored in ontology-based software architecture documentation.

In future studies we plan to investigate the use of SPARQL queries for more AK retrieval activities, additional completeness checking of AK, and estimate costs and benefits of using SPARQL queries in software projects.

VI. Acknowledgments

This research has been partially sponsored by the Dutch ”Regeling Kenniswerkers”, project KWR09164, ”Stephenson: Architecture knowledge sharing practices in software product lines for print systems” and by the Natural Science Foundation of China (NSFC) project No. 61472286 "CSAAD: Combining Software Architecture and Agile Development".

References

[1] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.

[2] P. Lago and P. Avgeriou, “First workshop on sharing and reusing architectural knowledge,” *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 5, pp. 32–36, 2006.

[3] D. Rost, M. Naab, C. Lima, and C. von Flach Garcia Chavez, “Software architecture documentation for developers: A survey,” in *European Conference on Software Architecture (ECSA)*. Springer LNCS, 2013, pp. 72–88.

[4] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros, “Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach,” *Science of Computer Programming*, vol. 77, no. 1, pp. 66–80, 2012.

[5] K. A. de Graaf, P. Liang, A. Tang, and H. van Vliet, “How organisation of architecture documentation affects architectural knowledge retrieval,” *Science of Computer Programming - Special Issue on Knowledge-based Software Engineering*, vol. 121, pp. 75–99, March 2016.

[6] R. L. Nord, P. C. Clements, D. Emery, and R. Hilliard, “A structured approach for reviewing architecture documentation,” SEI, Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-2009-TN-030, 2009.

[7] K. A. de Graaf, “Annotating software documentation in semantic wikis,” in *Proceedings of the fourth workshop on Exploiting semantic annotations in information retrieval*, ser. ESAIR ’11. ACM, 2011, pp. 5–6.

[8] K. A. de Graaf, P. Liang, A. Tang, W. R. van Hage, and H. van Vliet, “An exploratory study on ontology engineering for software architecture documentation,” *Computers in Industry*, vol. 65, no. 7, pp. 1053 – 1064, 2014.

[9] S. Auer, S. Dietzold, and T. Riechert, “Ontowiki a tool for social, semantic collaboration,” in *5th International Semantic Web Conference*. Springer LNCS, 2006, vol. 4273, pp. 736–749.

[10] K. A. de Graaf, P. Liang, A. Tang, and H. van Vliet, “Supporting architecture documentation: A comparison of two ontologies for knowledge retrieval,” in *International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 2015, pp. 3:1–3:10.

[11] I. Polikoff, “Comparing sparql with sql,” http://www.topquadrant.com/2014/05/05/comparing-sparql-with-sql/, 2014.

[12] S. Lohmann, P. Heim, S. Auer, S. Dietzold, and T. Riechert, “Semantifying requirements engineering – the softwiki approach,” in *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS 08)*, 2008, pp. 182–185.

[13] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American Magazine*, vol. 284, no. 5, pp. 34–43, 2001.

[14] T. Berners-Lee, “Linked data,” http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[15] C. López, L. M. Cysneiros, and H. Astudillo, “Ndr ontology: Sharing and reusing nfr and design rationale knowledge,” in *Proceedings of the 2008 First International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 2008, pp. 1–10.

[16] C. López, P. Inostroza, L. M. Cysneiros, and H. Astudillo, “Visualization and comparison of architecture rationale with semantic web technologies,” *J. Syst. Softw.*, vol. 82, pp. 1198–1210, August 2009.

[17] A. Tang, P. Liang, V. Clerc, and H. van Vliet, “Supporting co-evolving architectural requirements and design through traceability and reasoning,” in *Relating Software Requirements to Software Architecture*. Springer, 2011, pp. 59 – 85.

[18] S. Lohmann, T. Riechert, and S. Auer, “Collaborative development of knowledge bases in distributed requirements elicitation,” in *Software Engineering 2008*, 2008, pp. 22–28.

Appendix A
Implementation of SEI architecture review

We implemented 5 SEI (Software Engineering Institute) architectural review questions (see Figure 2). Listing 2 shows the prefixes we used in the SPARQL queries. These prefixes are shorthands for the ontology constructs that we use in our SPARQL queries. RDF syntax and RDF Schema allow use of their constructs, e.g, rdf:type and rdfs:subClassOf to query for classes. 'AKO' contains concepts from the AK ontology (see Section II).

```
PREFIX AKO: <http://www.archimind.nl/archimindLOD
    /index.php/view/r/sadocontology1.owl:>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>
```

Listing 2. Prefixes for used SPARQL queries

Listing 3 shows the SPARQL query to answer Question 1. All subclasses of 'Architecture' are retrieved in the first line of the WHERE statement, and in the next line all instances of these classes are retrieved. This returns instances of class 'Layer', 'Data store', etc. In the third line all instances of class 'Component' are retrieved. The aforementioned instances are implementation units in our AK ontology. The retrieved implementation units are finally matched by any relationship *(predicate variable ?p)* between them. In the AK ontology all relationships between the implementation units (i.e., instances of subclasses of Architecture) are development dependencies.

```
SELECT ?components ?p ?impl_units
WHERE { ?impl_unit_class rdfs:subClassOf AKO:
    Architecture .
?impl_units rdf:type ?impl_unit_class .
?components rdf:type AKO:Component .
?impl_units ?p ?components }
```

Listing 3. SPARQL query for Question 1: List ten development dependencies between implementation units.

Above query results in 17 correct[2] results for Question 1, whilst 10 correct results are needed (due to experiment timing constraints in [10][3]. The amount of SPARQL query results can be limited to 10, in order to match Question 1 exactly, by adding "*LIMIT 10*" at the end of the query. The same holds for the SPARQL query for Question 5 discussed later in this section. On the other hand, the query can be extended via *UNION* constructs to include all relationships to other implementation units, e.g., datastores. The query in Listing 4 extends Listing 3 to add more results.

```
UNION {?impl_unit_class rdfs:subClassOf AKO:
    Architecture .
?impl_units rdf:type ?impl_unit_class .
?datastores rdf:type AKO:Data_store .
?impl_units ?p ?datastores }
```

Listing 4. Extension of SPARQL query for Question 1 with data stores

[2]We used the answers of the SA document authors to the five questions in the experiment in [10] to set a ground truth of correct and incorrect results.

[3]due to which we considered 10 correct results as being perfect precision and recall for the question in this setting. Also see [10].)

The query in listing 5 answers Question 2 by retrieving all instances that have a certain relationship to requirement 'Security'. These relationships (e.g., *results_in*, *realized_by*) in the AK ontology relate instance '*Security*' to instances of decisions (class '*Design Issue*') and instances of the subclasses of Architecture, i.e., implementation units.

```
SELECT *
WHERE {
{AKO:Security AKO:results_in ?impl_units} UNION
{AKO:Security AKO:realized_by ?impl_units} UNION
{AKO:Security AKO:depends_on ?decisions} UNION
{AKO:Security AKO:addressed_by ?decisions}}
```

Listing 5. SPARQL query for Question 2: Which implementation units and decisions are explicitly related to requirement 'Security'?

The query in Listing 6 for Question 4 first retrieves all instances of class 'Functional Requirement', and from these we select the subset of instances that have a relationship to requirement 'compatibility'. The result is combined, using a *UNION* construct, with the results of a second (similar) subquery that finds relationships originating from requirement compatibility.

```
SELECT ?FuncReq
WHERE {{?FuncReq rdf:type AKO:
    Functional_requirement .
?FuncReq ?p AKO:Compatability} UNION
{?FuncReq rdf:type AKO:Functional_requirement .
AKO:Compatability ?p ?FuncReq }}
```

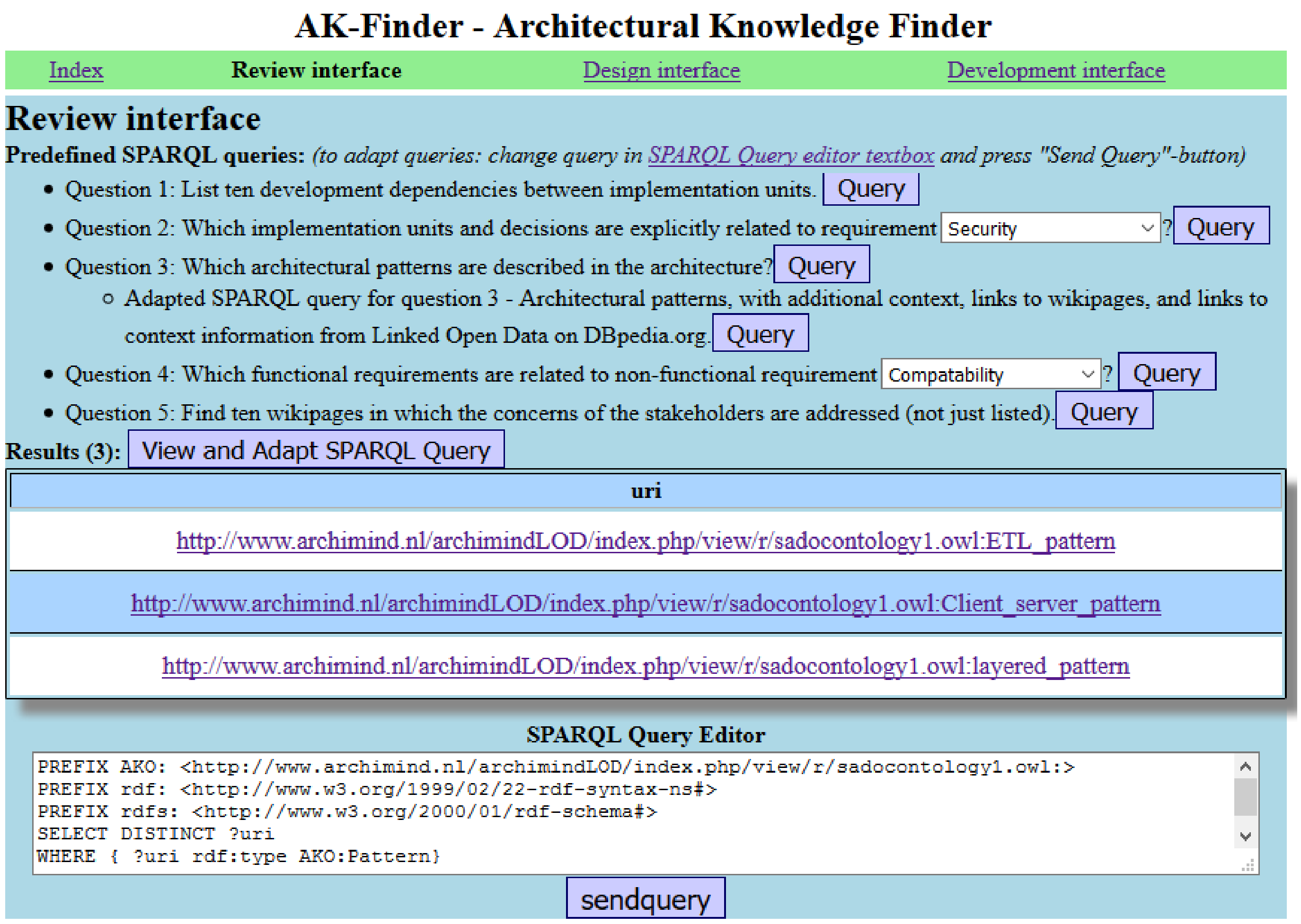
Listing 6. SPARQL query for Question 4: Which functional requirements are related to non-functional requirement 'compatibility'?

To answer Question 5 we first have to retrieve all concerns of the stakeholders. Next we retrieve all wikipages that contain knowledge about these concerns.

```
SELECT DISTINCT ?wikipage
WHERE {?concern rdf:type AKO:Concern .
?concern AKO:knowledge_is_located_in ?wikipage}
```

Listing 7. SPARQL query for Question 5: Find ten wikipages in which the concerns of the stakeholders are addressed (not just listed).

The AK ontology does not support relationships or properties to make explicit whether a concern is addressed or only listed on a wikipage. Therefore the query for Question 5 in Listing 7 returns a partially incomplete and partially incorrect answer, i.e., imperfect recall and precision. This means we need more specific relationships in the ontology to support the SPARQL query for Question 5. Adding a relationship 'addressed in' between class 'Concern' and 'Wikipage' in the AK ontology and subsequent semantic annotation of the SA document content would provide a complete and correct answer via the query.



AK-Finder - Architectural Knowledge Finder

Index | Review interface | Design interface | Development interface

Review interface

Predefined SPARQL queries: *(to adapt queries: change query in SPARQL Query editor textbox and press "Send Query"-button)*

- Question 1: List ten development dependencies between implementation units. Query
- Question 2: Which implementation units and decisions are explicitly related to requirement Security ? Query
- Question 3: Which architectural patterns are described in the architecture? Query
  - Adapted SPARQL query for question 3 - Architectural patterns, with additional context, links to wikipages, and links to context information from Linked Open Data on DBpedia.org. Query
- Question 4: Which functional requirements are related to non-functional requirement Compatability ? Query
- Question 5: Find ten wikipages in which the concerns of the stakeholders are addressed (not just listed). Query

Results (3): View and Adapt SPARQL Query

uri
http://www.archimind.nl/archimindLOD/index.php/view/r/sadocontology1.owl:ETL_pattern
http://www.archimind.nl/archimindLOD/index.php/view/r/sadocontology1.owl:Client_server_pattern
http://www.archimind.nl/archimindLOD/index.php/view/r/sadocontology1.owl:layered_pattern

SPARQL Query Editor

```
PREFIX AKO: <http://www.archimind.nl/archimindLOD/index.php/view/r/sadocontology1.owl:>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri
WHERE { ?uri rdf:type AKO:Pattern}
```

sendquery

Fig. 2. AK-Finder User Interface for Architectural Documentation Review with predefined SPARQL queries, (abbreviated) query results, and query editor.

Comparison Between Predefined Queries in AK-Finder and Manual Knowledge Traversal in ArchiMind semantic wiki

The five SEI architecture review question for which AK-Finder provides predefined SPARQL queries were also answered using standard semantic wiki features during an experiment in [10]. The experiment participants in [10] used the same ontology and same SA documentation in ArchiMind semantic wiki that AK-Finder queries via the SPARLQ query endpoint. Instead of predefined SPARQL queries in AK-Finder the participants used the standard features of ArchiMind semantic wiki, including navigation and listing of classes, semantic relationships, and instances, as well as keyword search, browsing, faceting, filtering, and semantic annotations in wikipages. See [5] and http://archimind.nl/archimindLOD/ for details and a demo of these features, and see [10] for more details and motivation around the retrieval task in the experiment.

Table I shows the efficiency ('seconds') and effectiveness ('F1score') of participants answering the 5 SEI questions in the previous experiment in [10]. On average AK-Finder returns answers to the SPARQL queries in less than a second (often around 100 milliseconds) with perfect precision and recall (F1 score: 1.0) for all questions except Question 5. Even though the AK ontology lacks a semantic relationship to correctly and completely answer Question 5 via the SPARQL query, the query was still more effective (F1 score: 0.89) than retrieval via standard semantic wiki features (F1 score 0.6).

This comparison between manual retrieval from a semantic wiki and use of predefined queries in AK-Finder is unfair, since we do not include the time-cost of writing the predefined queries. However, it gives an indication of possible benefits of predefined SPARQL queries in AK-Finder in terms of more correct and complete answers, and time-savings when the queries are used very often, thereby having a return on the time-investment of writing predefined queries.

TABLE I
TIME-EFFICIENCY (SECONDS), EFFECTIVENESS (F1 SCORE) OF MANUAL RETRIEVAL BY PARTICIPANTS IN [10] USING SEMANTIC WIKI

SEI Question	1	2	3	4	5
Efficiency (time)	1278	376	117	229	366
Effectiveness (F1 Score)	0.32	0.63	0.79	0.75	0.60
Number of measurements	19	15	12	12	10