Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 339 (2018) 43–62

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

An Ontology-based Approach for Sharing, Integrating, and Retrieving Architectural Knowledge

María Luciana Roldán[1] Silvio Gonnet[2] Horacio Leone[3]

Universidad Tecnológica Nacional, Facultad Regional Santa Fe
Instituto de Desarrollo y Diseño, CONICET
Santa Fe, Argentina

Abstract

The Architectural knowledge (AK) generated during software architecture projects is a valuable asset for software organizations. Although many organizations have adopted supporting tools to capture the produced AK, still there exist some difficulties in making it available to be retrieved by the consumers. Moreover, the boundaries of knowledge of an organization can be expanded to AK repositories that are shared or made public by other organizations. Thus, organizations have to deal with heterogeneity in knowledge representation, which makes complex the integration of knowledge from several sources. There is a need of an approach for sharing, integrating, and retrieving AK from various sources, which enables organizations to revisit and retrieve both their own and others' past decisions, as a basis for upcoming decisions. This paper describes an ontology-based approach for sharing, integrating, and retrieving knowledge from different AK sources, which is based on ISO/IEC/IEEE 42010. A proof of concept was developed to apply the approach on an AK management tool, and a scenario of knowledge retrieving was carried out.

Keywords: Software architecture, Ontologies, Software architecture knowledge, Knowledge retrieving

1 Introduction

As software systems become bigger and evolve, their software architectures become more complex and, software architects need to cope with a constantly growing amount of architectural knowledge. The knowledge generated during the software architecture process in several projects is a valuable asset for software organizations that can be shared in a community. Although many organizations have adopted some strategies and tools to support to the architectural knowledge (AK) producers (architects, designers, documenters), still there exist some difficulties in making the AK available to the AK consumers (maintainers, developers, reviewers, other

[1] Email: mailto:lroldan@santafe-conicet.gov.ar
[2] Email: mailto:sgonnet@santafe-conicet.gov.ar
[3] Email: mailto:hleone@santafe-conicet.gov.ar

https://doi.org/10.1016/j.entcs.2018.06.004

organizations, other management tools, etc.). We have found that, in fact, industrial organizations make attempts of retrieving and reusing artifacts, decisions and their rationale from previous projects, but they find this task difficult due to some barriers [4]. One of the barriers that inhibit the recovering and use of AK is that architecture knowledge management (AKM) tools do not provide efficient knowledge recovering features for it. In order to take advantage of this knowledge, organizations need adequate approaches for efficiently exploring and recovering it. Additionally, the boundaries of AK retrieving can be expanded to include AK from repositories shared or made public by other organizations. In this case, the barriers are the heterogeneity in knowledge representation and the integration of knowledge from several sources. Therefore, there is a need of an approach that enables the integration, sharing, and retrieving of architectural knowledge from existent AK sources, which should enable organizations to revisit and retrieve both their own and others' past decisions, as a basis for upcoming decisions.

Ontologies and Semantic Web are powerful mechanisms for representing knowledge and encoding its meaning. They can be used to model and represent the knowledge found in several sources, such as the knowledge generated and stored in AKM tools, in order to enable its integration, exploration, recovering, and exploitation as a reusing asset. This paper describes a preliminary ontology-based approach named SAK-SIR, which is based on ISO/IEC/IEEE 42010 standard, for integrating, sharing, and retrieving knowledge from different SAK sources. Then, as a proof of concept, we apply the approach for sharing, integrating, and retrieving knowledge on the knowledge provided from an AKM tool the authors have developed in previous works (TracED). A scenario of knowledge retrieving is developed regarding that TracED metamodel includes concepts for supporting the capture of the complete evolution of a Software Architecture Design Process (SADP).

The paper is organized as follows. Background on ontologies and semantic web technologies, and the ISO/IEC/IEEE 42010 is given in Section 2. In Section 3, the possible sources of SAK are identified and characterized, and the SAK-SIR approach is introduced. In Section 4, we apply the approach integrating the AKM metamodel and repository of TracED tool [14], [16], and retrieving SAK from historical SA projects. Section 5 discusses related word. Section 6 presents the conclusions.

2 Background

2.1 Ontologies for knowledge retrieving

Ontology-based techniques have gained acceptance as a means for tagging and performing semantic searches. Additionally, ontologies can be used to provide a common understanding for different terminologies and make the knowledge available for future use. In a previous paper [17], we have taken the first step to construct an ontology for software architecture knowledge (SAKOnto), which include concepts, relations, and constraints to represent knowledge from several categories: i) general knowledge, ii) context knowledge, iii) design knowledge, and iv) reasoning knowledge [19]. By definition, an ontology is an explicit formal specification of the

concepts (also referred to as classes) in a domain and the relations among them [21]. Ontologies are used in various application domains to facilitate a common understanding of the information structures in a domain and to enable reuse of domain knowledge. Ontologies enable a hierarchical classification of interrelated domain concepts. They can be represented using RDF [4] schema and OWL [5], which is a more expressive language. The use of RDF makes ontologies human readable and machine interpretable, allowing querying of and inference over knowledge [7]. OWL is a language for defining and instantiating Web ontologies. An OWL ontology may include descriptions of classes, their properties and instances. Given an ontology, the OWL formal semantics specify how to derive its logical consequences (infer new knowledge), in other words, facts that are not literally present in the ontology, but entailed by the semantics.

Regarding the potential of ontologies for knowledge retrieving, it should be noted that the RDF triples in a dataset, represent relationships between knowledge objects that are explicitly described and directly accessible. As a result, SPARQL [6] queries are considerably better aligned with users' mental models of a domain. Queries that have to traverse a chain of connections are particularly complex in traditional queries over relational databases, while very simple in SPARQL.

With RDF, schema level information is stored and queried the same way as data. It means that the conceptual data model could be fully explored through queries. It also makes it easy to create flexible driven queries. A key advantage of RDF is that it was designed specifically to readily merge disparate sources of data. Correspondingly, SPARQL includes a syntax to call two or more data sources within a single query. Therefore, these semantic technologies are suitable for being employed for retrieving knowledge from heterogeneous knowledge sources.

2.2 The ISO/IEC/IEEE 42010 standard and AKM tools

The ISO/IEC/IEEE 42010 International Standard for describing software architecture [9] provides a core ontology for the description of software architectures. The standards' core metamodel consists of stakeholders, system concerns, environment and architecture. Architecture is described in terms of architecture description, which consists of system elements, relationships between system elements, principles of the system design, and principles that guide the evolution of system over its life cycle. The ISO/IEC/IEEE 42010 provides a model characterizing the constructs that need to be captured for documenting SADPs and their rationale. The standard considers that an AD element is any construct in an architecture description. An architecture description includes one or more architecture views. An architecture view expresses the architecture of the system-of-interest in accordance with an architecture viewpoint. A view is governed by a viewpoint. An architecture view is composed of one or more architecture models. An architecture model uses modelling conventions appropriate to the concerns to be addressed. These conven-

[4] https://www.w3.org/TR/rdf-schema/
[5] https://www.w3.org/TR/owl-features/
[6] https://www.w3.org/TR/rdf-sparql-query/

tions are specified by the model kind governing that model. A model kind may be documented by specifying a metamodel that defines its core constructs. A metamodel presents the AD elements that comprise the vocabulary of a model kind. The metamodel should present entities, attributes, and relationships, which are all Architecture Description elements. The standard also states that when a viewpoint specifies multiple model kinds it is often useful to specify a single viewpoint metamodel unifying the definition of the model kinds. Furthermore, it is often helpful to use a single metamodel to express multiple, related viewpoints. Architecture rationale records explanation, justification or reasoning about architecture decisions that have been made. A decision can affect the architecture in several ways. These can be reflected in the architecture description as follows: requiring the existence of AD elements; changing the properties of AD elements; triggering trade-off analyses in which some AD elements, including other decisions and concerns, are revised; raising new concerns.

The most widely known AKM tools so far [19] have an associated repository to store and manage AK. In fact, one of the key differentiating factors among the AKM tools is the data model of the corresponding repository. Such data models define the architectural constructs and their relationships with each other, and thus represent architecture design knowledge that is used or generated during software architecture design and analysis. A data model can help organizations to define and obtain data on various aspects of their architectural assets and design rationale during the software architecture process [4].

Several AKM approaches of the last ten years describe the connection between design decisions and other software artifacts and how decisions can be represented, which is aligned with the ISO/IEC/IEEE 42010 standard. Therefore, integrating these concepts in an ontology would not be a difficult task, given the advantages of languages like RDF and OWL for defining hierarchical relationships and set any kind of semantic association between concepts and instances.

However, each AKM approach proposes its own architectural description elements, which includes a series of specific concepts that need a suitable representation in an ontology that is compatible with the concepts of the ISO/IEC/IEEE 42010 standard. Many of the concepts could be missing in the standard, or could be present but expressed in a generic way. This means that in some cases, it is required the work of an expert or ontology engineer for the identification and specification o of concepts, relations, properties and rules in order to define the ontology.

3 The SAK-SIR Approach

The approach for Sharing, Integrating, and Retrieving Software Architecture Knowledge (SAK-SIR approach) is scoped in the AK producer-consumer model revisited by Tang et al. [19], which describes the general activities and stakeholders of AK. That producer-consumer model has at the heart the concept of Reasoning Knowledge, which includes design decisions, rationale, alternatives considered, tradeoffs made, and forces identified. Related to this knowledge are: General Knowledge,

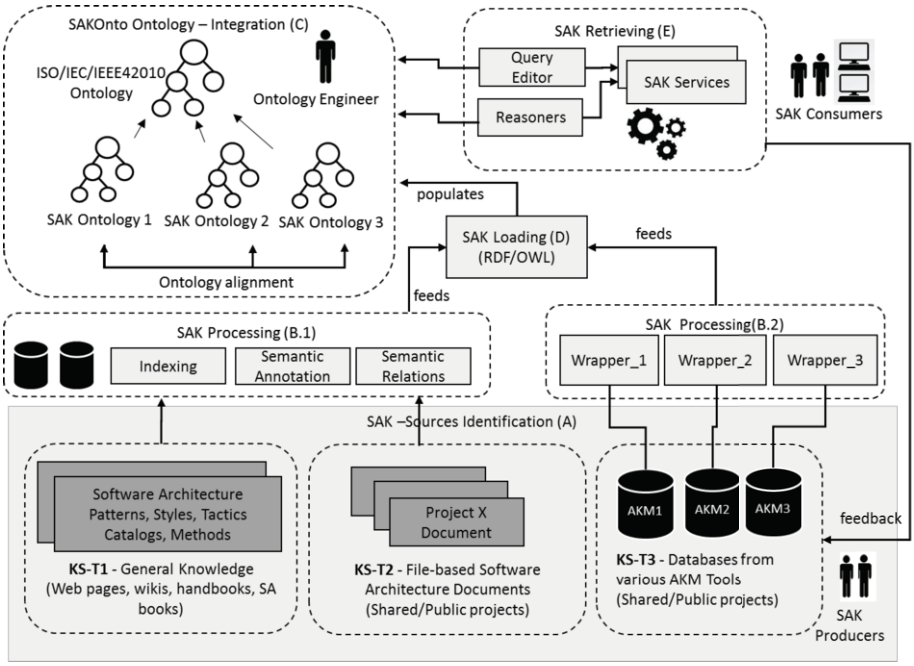

Fig. 1. An overview of the SAK-SIR approach.

Context Knowledge, and Design Knowledge, which could be traced to the Reasoning Knowledge. General Knowledge includes knowledge that is usable in any system design (e.g. architectural styles, patterns, and tactics), whereas Context Knowledge contains the knowledge about the problem space, which is specific to a system, e.g. architectural significant requirements and project context. Design Knowledge includes the knowledge about the design of the system, e.g. architectural models or views. Together these four types of knowledge make up the concept of Architectural Knowledge. A Consumer can Learn and Search/Retrieve this AK. In addition, a Consumer can evaluate the Reasoning/Design Knowledge. A Producer produces Reasoning Knowledge by architecting, and Design Knowledge by synthesizing. The Producer can also distill and apply General Knowledge while architecting/synthesizing and integrate Context Knowledge. Last, the Producer can share the produced Architectural Knowledge.

The SAK-SIR approach focus in how the producers share the software architecture knowledge (SAK) they own, how to integrate it with other knowledge sources, and provides guidelines on tools to make possible to consumers to explore and retrieve knowledge. SAK-SIR consists of five stages, not being all of them completely computer-supported, as some human expert work is still necessary. They are: A)*SAK Sources Identification*, B) *SAK Processing*, C) *SAK Integration*, D) *SAK Loading*, and E) *SAK Retrieving*. An overview of SAK-SIR approach is shown in Figure 1.

3.1 SAK Sources Identification Stage (A)

SAK sources can be characterized regarding whether they are internal or externals to an organization, regarding if their content is file-based, or ontology-based, or structured in a data-base, regarding the kind of AK they provide (General, Context, Design, or Reasoning), etc. In this work, the following knowledge sources types have been identified (KS-TX):

- **KS-T1**: Software architecture texts and publications, web pages and catalogues, standards. This knowledge is general and not related to specific architecture design projects from an organization.
- **KS-T2**: Software Architecture Documents, Architectural Decision Templates, Wikis (file-based approaches, ontology-based approaches). Their content are derived from specific architecture design projects.
- **KS-T3**: Metamodels and projects repositories from Architectural Knowledge Management tools.

KS-T1 sources (Figure 1) are public and provide general AK knowledge, like recurring SA practices, architectural patterns, styles and tactics, as well as knowledge about methods and documentation approaches [3], [20]. KS-T1 sources are mostly textual documents. KS-T2 and KS-T3 SAK sources can be internal to an organization or shared/made public by an external organization, and provide knowledge about SA projects developed by the organization that produced it. KS-T2 sources provide knowledge that is scattered in several documents, which are the products of different SA projects. They came mostly from documents created by software architects with some kind of text processor. In addition, they can be documents generated from file-based approaches, or employing a template fashion. In this group, we consider ontology-based documentation approaches that implement a basic software ontology and semantic wiki tools, thus to address the limitations of file-based documentation for knowledge retrieving. KS-T3 sources are from AKM tools [11], [5]. This knowledge comes from, both the metamodel on which is based the implementation of the AKM tool and the SA projects developed with the tool. Within KS-T3 category, some approaches for SA development that involve the use of ontologies about general software architecture knowledge can be also considered. The identification of the characteristics of a SAK source derives in the selection of the suitable methods and techniques that have to be applied in order to process the knowledge and make possible its integration with other available SAK.

3.2 SAK Processing Stage (B)

Depending on the knowledge source, the techniques and mechanisms applied in this stage vary. In the case of textual documents (KS-T1 and KS-T2), this stage comprises a series of extraction techniques based on Natural Language Processing (NLP), and the identification of terms and attributes, which has to be performed using sophisticated tools. Parsing of web pages can be used also. Some annotations or tagging mechanisms are also necessary in order to add the semantic. For these

types of knowledge sources, a specific domain ontology should be created. The objective for defining an ontology is twofold: to identify terms (or individuals) and their relationships in the documents (text files); and to make possible the integration of these knowledge sources with available knowledge residing in other ontologies. The last activity should be done as part of SAK Integration stage.

When KS-T3 sources are considered, some adapting mechanisms are needed. It may be necessary the programming of specific-purpose wrappers or adapters to convert data to a common format, or the use of an API provided by the AKM tool to enable accessing its repository.

3.3 SAK Integration Stage (C)

This stage constitutes the core of the SAK-SIR approach. The activities of knowledge integration are supported by the SAK Ontology Network. The SAK Ontology Network is organized in four ontology levels, which are illustrated in Figure 2. They are the following:

L1) ISO/IEC/IEEE 42010 Standard Ontology Layer. The ISO/IEC/IEEE 42010 standard [9] ontology constitutes a basis for describing all the concepts included in the SAK ontology network. Therefore, all the networked metamodel and domain ontologies inherit the same foundational concepts and relations, which is essential for ontology integration.

L2) Metamodels layer. This layer defines the concepts, relationships, and rules that represent the terms of the Architectural Descriptions that an AKM approach implements. This layer is built on L1 mainly through the extension of the hierarchy's concepts and object properties, using RDF Schema constructs like *rdfs:subClassOf* and *rdfs:subPropertyOf*.

L3) Software Architecture Domain layer. This layer, which is built on L1 and L2 layers, includes the concepts that are specific from SA domains. A domain comprises a set of concepts, regarding the building block types and their possible relations to be employed in a SA project for describing a software architecture. A domain also can include the definition of generic design decisions, like design patterns and styles.

L4) Projects Layer. This layer comprises individuals that populate the SAK ontology, whose *rdf:type* is given by concepts or object properties from any of the above layers.

Although all the AK metamodels and domain ontologies integrated in the network share the same conceptual basis, they still need to be aligned with respect to their specific knowledge, to make possible retrieving knowledge from different ontologies. RDF and OWL provides useful constructs to specify property axioms, like equivalence relations between classes (*owl:equivalentClass*) and relations to other properties (*owl:equivalentProperty* and *owl:inverseOf*), and logical property characteristics (*owl:SymmetricProperty* and *owl:TransitiveProperty*).

The ontology that underlies the SAK network and constitutes L1 layer was bor-

rowed from others authors proposal. Guessi et al. [6] proposed OntolAD, an ontology expressed in OWL 2 for architectural descriptions based on the ISO/IEC/IEEE 42010 standard. That ontology was built through the transformation of the primitive AD constructs into OWL classes. Many relationships described in the ISO/IEC/IEEE 42010 standard were implemented as object properties in OntolAD. However, the authors did not provide either the domain or range for object properties. Therefore, the ontology was enriched which adequate domain and range for each object property in order to add semantic to the relationships among concepts. In addition, some axioms on OntolAD were too restrictive and we interpreted that did not reflect the standard semantic, so they were eliminated.

It should be noted, that the SAK-SIR approach it is not fully computer-supported, since human intervention is necessary. At integration stage, the role of an ontology engineer or expert is crucial to guaranty quality and expressiveness of the knowledge model. For example, regarding the integration of knowledge from a KS-T3 source, the ontology engineer (with the help of the architects of the involved organizations) should identify the concepts defined in the metamodel of the AKM tool that wants to integrate. Thus, these concepts have to be defined as an extension of concepts in the IEEE 42010 standard ontology (*st* prefix). For example, let us consider that the AKM tool metamodel ontology (*akm* prefix) includes a specific type of view for describing detailed decisions using a template fashion. Then, the class *akm:DetailedDecisionViewType* has to be defined as subclass of the *st:ArchitectureViewType*, as well as the necessary data properties, annotation properties and object properties that involve that concept, and other concepts related to this kind of view that are not defined in the ontology yet.

3.4 SAK Loading Stage (D)

While the previous stage concerned the integration of a metamodel/scheme in the ontology network (the meaning of the vocabulary of properties and classes of an AKM tool or set of documents), SAK Loading stage concerns the population of that scheme. The individuals that populate the ontologies come from the knowledge/artifacts generated during several SADP projects.

Ontology population can be done by manually creating the necessary instances and triples, or by means of a specific loader program that transform data from repositories/data bases in convenient RDF triples. The JENA ontology API can be used for that.

3.5 SAK Retrieving Stage (E)

This stage comprise a set of tools and mechanisms for exploring and retrieving the SAK that can be gathered from internal/external sources and integrated in the SAK ontology network. Due to the diverse needs of knowledge of the SAK consumers, the approach has to provide a flexible interface for placing queries on the ontologies (*Query Editor*, in Figure 1). On the one hand, it has to be intuitive and user-friendly. Moreover, on the other hand, it has to be powerful enough to create complex queries.

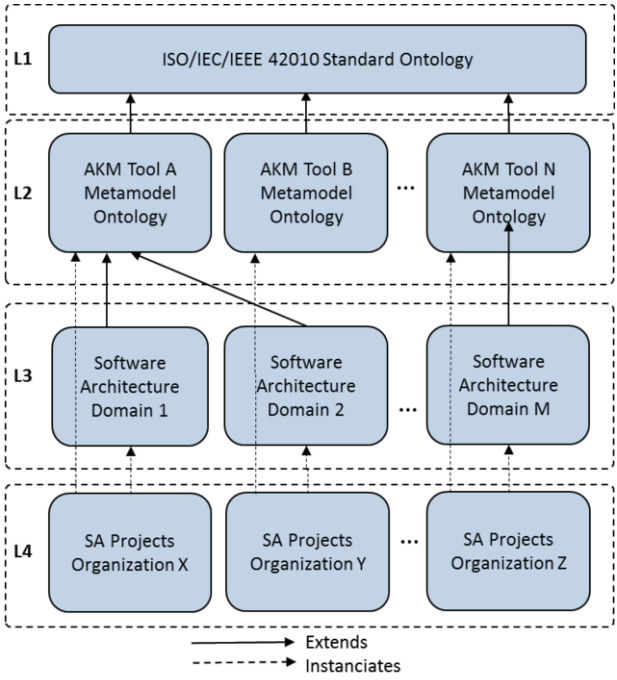

Fig. 2. SAK Ontology Network Layers.

A query editor should assist the users in the queries definition. Complex inferences about the SAK knowledge available can be made by means of a reasoner to generate non-explicit knowledge. Additionally, a set of predefined SPARQL queries can be available for being executed as services, which would achieve interoperability among different AKM tools.

4 A Scenario of Applying the SAK-SIR Approach

Although the proposed approach is intended to integrate heterogeneous architectural knowledge available from several sources, in this work, we reduce the scope to achieve the integration of the concepts of TracED with the ISO/IEC/IEEE 42010 standard, and to populate a dataset with the knowledge gathered from different SAK projects. Then, some queries are placed to retrieving valuable knowledge from the evolution of a SA.

4.1 SAK Sources Identification

TracED [7] is an AKM developed by the authors that is available for being employed for research and educational purposes. The metamodel on which TracED it is based is easily accessible and has been published [14], [16]. TracED repository preserves the knowledge captured in the several projects that were developed on different SA domains, with a structured representation and some textual documents (Decisions

[7] http://traced-doc.appspot.com

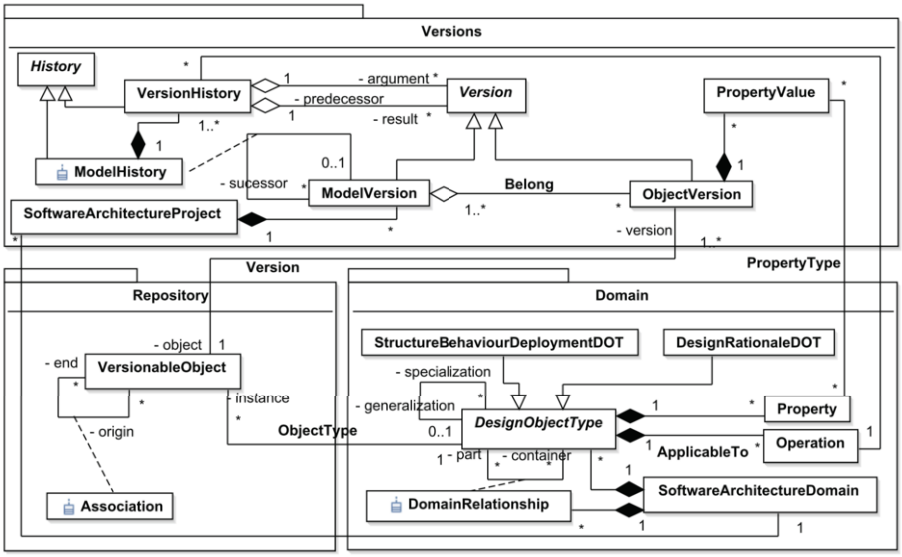

Fig. 3. TracED metamodel for supporting versioning and tracing of SA model versions.

templates), which is supported by a traditional database. However, the repository of TracED tool, has not been shared with other parties nor made public, and is not available an API for accessing to its content. Therefore, it constitutes an internal source of knowledge for the authors' organization, which means that the structure of TracED repository is known and accessible through a database connection.

4.2 SAK Processing

The aforementioned characteristics (the fact that TracED is an internal knowledge source and, therefore, the organization knows the tool's metamodel) make unnecessary to perform actions in this stage of the approach.

4.3 SAK Integration

In a previous work [17], SAKOnto ontology was built from scratch. A methodology based on competency questions was employed for guiding the SAKOnto developing process [7]. The metamodel of TracED was analyzed and their main terms and relationships were identified and defined in the ontology. However, to apply the SAK-SIR approach, and achieve a successful integration in the ontology network, the ontology needs to be specified based on the concepts of ISO/IEC/IEEE42010 standard. That means that some concepts will be defined at the Metamodel Layer, as "specializations of" some classes and object properties that the standard defines. Thus, the integration stage involves adding into the ontology network the ontology that includes all the concepts, relations, and axioms for representing the TracED metamodel (Figure 3).

Metamodel Layer

TracED [14], [16] considers an architectural design decision as the transformation between two states of the software architecture. The tool adopts an operational perspective where design decisions are materialized by the execution of a sequence of design operations. Design operations are the actions performed by a designer, which operate on the products of the design process. These products, called *design objects* in this proposal, constitute an abstraction of what is eventually desired to be realized in the real word, and are the results of other design decisions made by the designers. Typical *design objects* are the structural elements of the artifact that is being designed (i.e., the components and connectors that comprise a software architecture), and the specifications to be met (i.e., quality requirements such as modifiability or performance). These objects evolve as the SADP takes place, giving rise to several versions. The set of versions of the design objects at a given point of the design process constitute a *model version* of the software architecture. A *model version* describes the state of the design process at that point.

The design decisions made during a SADP project can be materialized in sequences of operations. A *sequence of operations* ϕ is applied on a *model version* (called *predecessor model version*) in order to generate a new model version (called *successor model version*). Under this approach, the SADP follows a tree-structure, where each node represents a model version. The *initial model version* is the root of the tree and each arc represents a *sequence of operations* applied to a *predecessor model version* to obtain a new *model version*. Therefore, a model version is the result of the history of the design process given by all sequences of operations applied since the initial model version.

Figure 3 shows the main elements of TracED metamodel, where *design objects* are represented at two levels; the *repository* level and the *version level* (see *Versions* and *Repository* packages). The *Repository* level keeps a unique entity for each *design object* that is created and/or modified due to the natural progress that takes place during a design project. Any entity kept in the repository is regarded as a *versionable object*. Furthermore, associations among the different *versionable objects* are also held in the repository to represent the configuration of the architectural model (*Association*, Figure 3). The *Version* level maintains the different versions resulting from the evolution of each design object, which are called *object versions*. The relationship between a *versionable object* and each of its object versions is captured by the *Version* association. Therefore, for a given design object, a unique instance is kept in the repository, and all the versions it assumes along the design process in different model versions belong to the versions level.

Therefore, TracED represents the evolution of an architectural model as a *history*, where a new *model version* is generated by applying a *sequence of operations* on a predecessor model version. Each sequence of operations applied on a model version is captured by means of a *model history* link (represented by the class association *Model History*, Figure 3), and each *operation* executed in the sequence of operations is captured by a *version history* link (represented by class *VersionHistory* Figure 3) that also keeps traces among the *object versions* on which the *operation* was ap-

plied and the arising object versions as a result of the operation execution. As it is shown in Figure 3, a *Model History* link is an aggregation of various *VersionHistory* links. In the *Domain* package, the central class is *Design object type*, which provides the elements for defining the possible building block types of an architecture model or view. *Design object types* are specialized according to the view type they are suitable to describe in TracED. Given that TracED supports view types such as Component and Connector, Deployment [3], and Architecture rationale templates. *Design Object Type* is specialized in *StructureBehaviourDeploymentDOT* and *RationaleDOT* classes. The properties of each type are specified by a set of instances of *Property* class. Furthermore, the possible relationships among the different *design object types* of a domain are described by means of *Domain Relationship*. TracED metamodel also provides elements for the specification and execution of operations applicable in a domain [14], [16], [15]. Operations are defined in terms of add, delete, modify primitives and other domain operation, enabling the materialization of operations as complex as a pattern applying and to set explicit traces with the requirements achieved. Due to space constraints, these details are hidden in this case study.

Integrating TracED metamodel in the ontology network implies the definition of each term as an *owl:class* and setting adequate *rdf:subClassOf* properties with the respective concepts in *L1*. Figure 4 shows that *traced:SoftwareArchitectureDomain* is a specialization of *st:Metamodel* class.

The *traced:DesignObjectType*, *traced:DomainRelationship*, and *traced:Property* concepts are defined as specializations of *st:Entity*, *st:Relationship*, and *st:Attribute* classes respectively. On the other hand, *traced:ModelVersion* is defined as subclass or *st:ArchitectureModel* and *traced:ModelHistory* is included as subclass or *st:ArchitectureDecision*.

Figure 4 shows the definition of the metamodel concepts as classes and the correspondent object properties, which were inserted in a RDF graph prefixed as *traced*. For each class, a set of property annotations was added to maintain the metadata about each concept in the ontology. The *rdfs:label* and *rdfs:comment* annotation properties were used to represent the name of the concept and the description of the concept. In addition *rdfs:isDefinedBy* annotation property indicates the ontology where the subject was defined.

Domain Layer

TracED enables defining several domains, depending on the context of the system whose architecture is going to be designed. The specific design object types of the domains were included at the domain layer of the SA ontology network as subclasses of the concepts in the standard and metamodel layers, thus employing *rdfs:subClassOf* and *rdfs:subPropertyOf* properties. In Figure 5 a partial view of the integration of a domain is shown, where *IaaSCloudDomain*, a domain that defines design object types for the architecture of a cloud system, is specified in the ontology. In this domain, the design object type is specialized in specific component for cloud systems: *Wrapper*, *DataBase*, and *Synchronizer*.

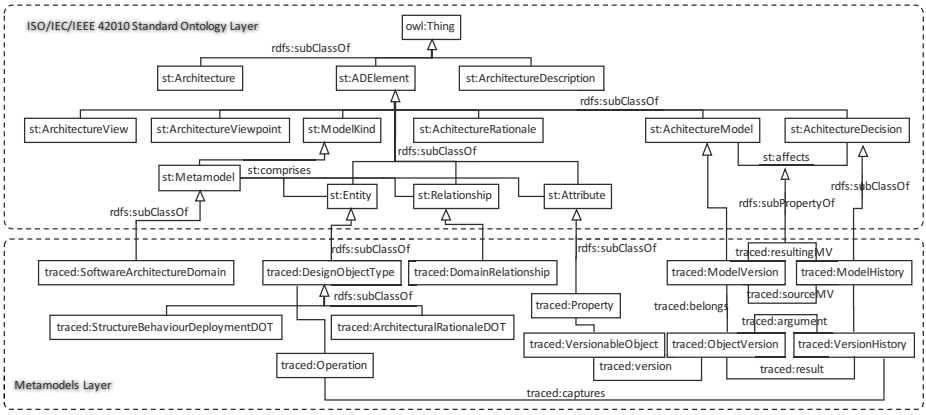

Fig. 4. TracED metamodel concepts for supporting versioning and tracing of SA model versions integrated in the SAK Ontology Network.

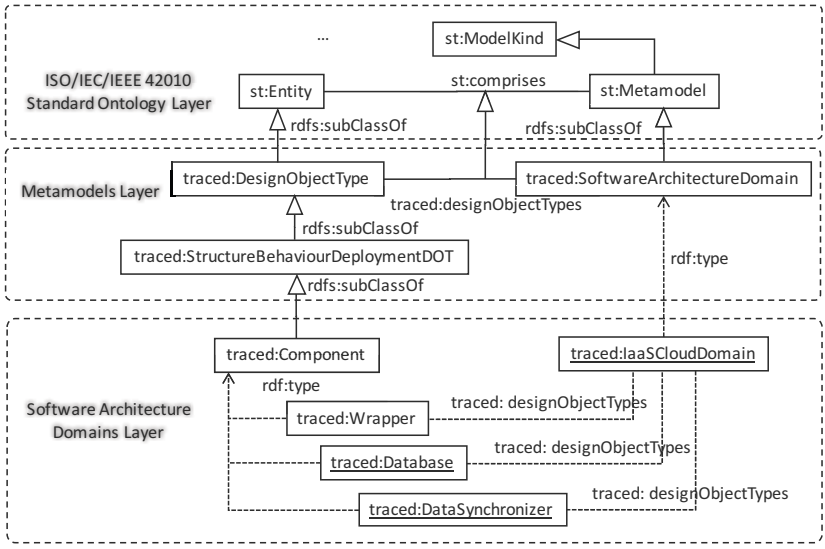

Fig. 5. Partial view of integration of ISO/IEC/IEEE 42010 standard concepts and TracED metamodel and domain concepts.

4.4 SAK Loading

We loaded the ontologies on an Apache Fuseki Server. Regarding the scope of this work, we created a dataset "SAKOnto" that has two RDF triples graphs. The first graph was loaded from the available IEEE 42010 standard ontology (OntolAD, prefix *st:*), which imports the namespace of OntolAD, and the second graph from the TracED ontology (prefix *traced:*). TracED graph was populated with the knowledge gathered from projects developed using TracED tool.

Accessing to TracED database is straightforward since it is an internal tool of the author's organization, so the projects' data were imported as RDF triples, by means of a loader program implemented with the JENA API. Next, it is described a fragment of project developed in a cloud migration domain that was loaded from TracED repository [15].

SportsInc is a firm traditionally dedicated to the wholesale of sportswear and apparels. In the last decade, the firm expanded its business to e-commerce retail sale, maintaining hardware and software infrastructure for running the web store. Due to the increasing number of online sales, the firm decided to migrate the web store to the cloud to decrease infrastructure and maintenance costs, eliciting new quality requirements of scalability, performance, and availability to be satisfied by the software architecture. Still, *SportsInc* decided keeping operative the legacy system, which mainly is dedicated to wholesale orders and distribution management. For security reasons, *SportsInc* wants to keep full control of databases, which will remain deployed on-premises.

Although TracED provides a visual user interface, where operations are executed using forms to input the parameters values, and the resulting model versions are presented using a components-and-connectors viewpoint, the architecture decisions made in this project are presented in the compact format of sequences of design decisions applied on a model version.

The first model version of this project was mv_0, where $SportsIncSystem_{v1}$ was the system, and the *Retailing* module architecture was given by existent $Presentation_{v1}$, $BusinesLogic_{v1}$, and $DataAcces_{v1}$ components, among other components of the *Wholesale* module and $DBSportsInc_{v1}$ database. The first decision was materialized in the sequence of operations ϕ_1, which is applied on mv_0 and consists on the identification of new quality requirements that the architecture must satisfy.

$$\begin{aligned}\phi_1 = \{&addQualityRequirement(SportsIncSystem_{v1}, \text{'Scalability'}),\\ &addQualityRequirement(SportsIncSystem_{v1}, \text{'Performance'}),\\ &addQualityRequirement(SportsIncSystem_{v1}, \text{'Availability'})\}\end{aligned}$$

The next decision made was to make an agreement with a provider of cloud services, under an *IaaS* contract, to migrate the components of the Retailing subsystem (web store). Therefore, the *Cloud Service Provider* (CSP) provides a Web server virtual machine, on which the web store's presentation layer is going to run. Similarly, business logic layer and data access layer were moved to the cloud, thus allocating them on the top of an Application server virtual machine. In order to minimize the users' web store perceived latency, a database virtual server was created, to locate a replica of the database. These decisions were captured by the following sequences of operations, starting from mv_1.

$$\phi_2 = \{addCloudProviderNetwork(SportsIncSystem_{v1}, \text{'CSPNetwork'})\}$$

$$\begin{aligned}\phi_3 = \{&addVirtualMachine(CSPNetwork_{v1}, \text{'VM-WebServer'}),\\ &addVirtualMachine(CSPNetwork_{v1}, \text{'VM-AppServer'}),\\ &addVirtualMachine(CSPNetwork_{v1}, \text{'VM-DBServer'})\}\end{aligned}$$

$$\phi_4 = \{reallocateComponent(Presentation_{v1}, \textit{VM-WebServer}_{v1}),$$
$$reallocateComponent(BusinessLogic_{v1}, \textit{VM-AppServer}_{v1}),$$
$$reallocateComponent(DataAccess_{v1}, \textit{VM-AppServer}_{v1})\}$$

To keep a secure access to *SportsInc* virtual private network, a *VPN Gateway service* was contracted. This decision is materialized in the sequence of operations ϕ_5 applied on model version mv_4, thus generating mv_5.

$$\phi_5 = \{addComponent(CSPNetwork_{v1}, \textit{'VirtualNetworkGateway'},$$
$$[\textit{'Resp-ConnectVPN'}], [\textit{'PortVPNG1'}, \textit{'PortVPNG2'},$$
$$\textit{'PortVPNG3'}, \textit{'PortVPNG4'}, \textit{'PortVNPG5'}])\}$$

Then, *user authentication*, *data validation*, and order submission responsibilities were assigned to the user interface of the web component (*Presentation layer*) for enabling sales representatives to entry orders in the system (sequence of operation ϕ_7 on model version mv_5).

$$\phi_6 = \{addResponsibility(Presentation_{v2}, \textit{'Resp-OrderFormPresentation'}),$$
$$addResponsibility(Presentation_{v2}, \textit{'Resp-Autentication'}),$$
$$addResponsibility(Presentation_{v2}, \textit{'Resp-OrderFormDefaultValues'}),$$
$$addResponsibility(Presentation_{v2}, \textit{'Resp-OrderFormValidation'}),$$
$$addResponsibility(Presentation_{v2}, \textit{'Resp-Submit'})\}$$

The next decision, materialized in the sequence of operations ϕ_7 applied on model version mv_6, involved the applying of a cloud pattern that added a wrapper component, which would be in charge of encapsulating and forwarding order submissions from the web system to the *OrdersManagement* component in *Wholesale* subsystem, which, in turn, would place them in the master database. Orders forwarded by the *wrapper* were channelized through the *VPN gateway* component.

$$\phi_7 = \{addWrapper(CSPNetwork_{v1}, \textit{'Orders-Wrapper'},$$
$$[\text{'Resp-ReceiveRequest'}, \textit{'Resp-ForwardRequest'}], [\textit{'PortOW1'}, \textit{'PortOW2'}],$$
$$Presentation_{v2}, VPNGateway_{v1})\}$$

In order to improve the web system performance and reduce latency, the architect decided to replicate the database and deployed it on *VM-DBServer* (represented by sequences of operations ϕ_8, ϕ_9, and ϕ_{10}). The replicated database contains a reduced data schema, which includes the product catalogue, costs, and current offers. Additionally a synchronization service was incorporated, which runs on-premises and communicates the master and replicated databases through the *VPN gateway* (some decisions where dismissed to reduce the project size).

$$\phi_8 = \{replicateDataBase(DBSportsInc_{v1}, \textit{'DBSportsInc-Catalogue'},$$
$$\textit{'Products catalogue and marketing schemes'}, \textit{VM-DBServer}_{v1})\}$$

$$\phi_9 = \{addConnector(CSPNetwork_{v1}, \text{'DataAccess-DBCatalogue'}, [\text{'Role31'}, \text{'Role32'}], [PortDA2_{v2}, PortDBC1_{v1}])\}$$

$$\phi_{10} = \{synchronizeDatabases(SportsIncSystem_{v1}, \text{'DBSynchronizer'}, [\text{'Resp-FindChanges'}, \text{'Resp-UpdateSchema'}], [\text{'PortSync1'}, \text{'PortSync2'}], DBSportsInc_{v1}, DBSportsInc\text{-}Catalogue_{v1}, VPNGateway_{v1})\}$$

Populating the ontology with the knowledge about this project means that for each executed sequence of operations, an individual has to be created, as an instance of *traced:ModelHistory*. Furthermore, each model history is composed by instances of *VersionHistory*, which represent each applied operation. The resulting model version is an instance of *traced:ModelVersion*, and each generated object version is an instance of the *traced:ObjectVersion*. An instance of *traced:ObjectVersion* maintains an object property to its respective *traced:VersionableObject*, which, in turn, has an object property (relationship) with the respective design object type.

4.5 SAK Retrieving

Once loaded the project in the dataset, a series of SPARQL queries were executed using the Apache Jena Fuseki server. Next, some example of queries to retrieve the SAK about the *SportsInc* project are proposed:

(i) Which were the architectural elements affected due to certain decisions (applied operations or sequence of operations)?

(ii) Which design operations/patterns were applied when evolving from model version X to model version Z?

(iii) Which quality requirements were changed, added or improved?

(iv) Which new architectural elements were generated as a consequence of performing X design decision?

(v) Which responsibilities were added to component C?

Figure 6 shows the model versions generated from *ModelVersion_0* to *ModelVersion_4*. Particularly, the operations applied during *sequenceOfOperations_4* are shown along with the design objects affected by each operation.

5 Related Work

Some of the software architecture ontologies that have been proposed in literature [2], [8] are focused in the representation of general knowledge, like architectural patterns for achieving certain qualities, tactics, types of views and their corresponding building blocks, and other knowledge like the trade-off relationships that exist among quality requirements. ArchVoc [2] is an approach to identify these kind ok knowledge about software architectures, which employ different techniques. On the one hand, the approach performs manually searches through back-of-the-book index of some of the major texts in Software Architecture, and on the other hand, employs

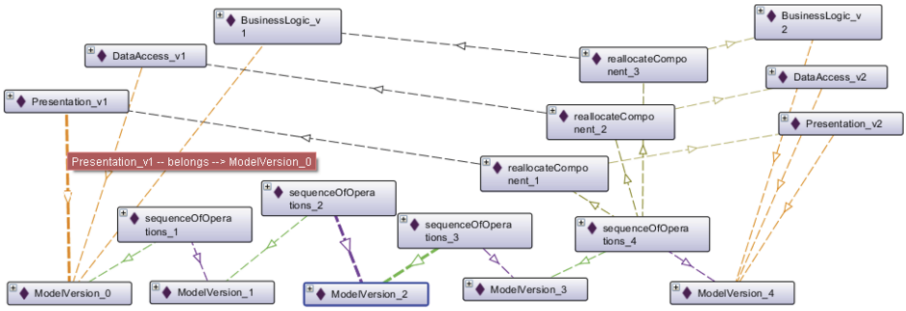

Fig. 6. Knowledge retrieving from *SportsInc* project. In this example a query enables to find out the model versions that were successively generated from ModelVersion_0 to ModelVersion_4. Particularly, the operations applied during sequenceOfOperations_4 are shown along with the design objects affected by each operation.

a semi-automatic parsing technique on Wikipedia pages. Only general architecture knowledge is considered to construct the ontology, skipping the knowledge generated on projects that software organizations can share.

Other authors [1], [10], [13], [18] proposed ontologies to support the design of software architectures and the representation of architectural decisions along with rationale. However, these ontologies are not related to other supporting tools for architects, which constitute the sources of knowledge to feed such ontologies. Other ontologies have been proposed to provide semantic to file-based software architecture documents, in order to improve the retrieval of knowledge [11], [5]. Lopez et al. [11], use the Toeska Architecture Ontology and the NFR Design Rationale (NDR) ontology to represent, recover and explore software architecture and rationale information from text documents. The extraction mechanism is based on Natural Language Processing (NLP) techniques, and the identification among terms and attributes has to be performed using sophisticated tools. De Graff et al. [5] employed a similar approach aimed at addressing the issues of file-based documentation. They implement a lightweight software ontology and a semantic wiki tool. However, these proposals have not explored the integration of the ontology's terms with the IEEE 42010 concepts. Since these proposals are aimed to retrieve knowledge from sources that provide file-based documents, the set of techniques employed can be adopted as a solution in the Processing stage of SAK-SIR approach.

The work of Figueiredo et al. [8] proposes a search mechanism to help to find AK. To achieve this goal, they propose the use of application domain ontologies to enable the retrieval of the rationale related to the software construction. This architectural knowledge is relevant in a virtual community environment where many projects share the same domain abstractions. On the contrary, our approach is aimed to share projects of organization that manage different abstractions for representing AK.

Regarding SADP evolution knowledge retrieving, Capilla et al. [8] presented a review of research tools for AK management, and evaluated their support for producers and consumers of AK. Among other features, they reviewed their support on "evolution" of AK, analyzing if the tools offer some kind of versioning or decision history mechanisms. They report that few approaches consider evolution o history

traces among design elements, their versions, and the decisions that generated them. Other authors [22], [12] proposed a documentation framework for design decisions, which provides a Decision Chronological viewpoint. This view shows the evolution of architecture decisions in chronological order. The objective of the chronological view is to show all versions of every architecture decision of a system and to enable keeping track of every state change of a decision. For instance, a decision that was tentative, then became decided and finally approved is represented with three instances in one chronological view. This approach defines the concept of iterations as versions of the architecture as a whole, which seems to be similar to the concept of model version proposed in TracED. However, different from the knowledge captured by TracED approach, the architectural decision artifact used for representing a decision provides just partial information about the decision made. In order to complete the knowledge about an architectural decision, the framework provides a decision detailed viewpoint, which is implemented as a template, thus employing a textual representation. Therefore, the approach lacks of a adequate representation of the design elements that are versionable (like components, connectors, interfaces, responsibilities, etc.), and prevents to recording how they have changed during the software design process (as a consequence of design decisions, like operations of refinement, change of property values, splitting, delegation, merging, etc.). It is worth to clarify, that TracED tool enables capturing SADP knowledge as it takes place, different from other AKM tools where the decision documentation is made after the fact, which constitutes the main reason why the chronological views offered by those tools are not employed in practice.

6 Conclusion

In this work, the *SAK-SIR approach* was presented. It aims at integrating heterogeneous knowledge sources, and providing knowledge retrieving and semantic reasoning capabilities, thus making possible sharing SAK among several organizations or individuals. ISO/IEC/IEEE 42010 standard ontology was used as the basis for defining a SAK ontology network. The approach was validated with the integration of the metamodel of *TracED* tool in the SAK ontology network. The resulting ontology was populated with knowledge from the projects of TracED repository. The concepts, relations, and instances included enable the representation of knowledge about the evolution of the *SADP*, while keeping compliance with the standard.

Employing ontologies for SAK representation enables to easily maintain several relationships among concepts and individuals by means of triplets, and provides flexible capabilities for the retrieving knowledge setting queries through the triples. These tools contribute to address some challenges on SAK retrieving that other authors have stated [4]: a) to understand the ripple effect of decisions; b) to identify and track the root causes of changes and estimate better the impact analysis; c) to understand the evolution of the system by capturing the decisions made, their alternatives and chronology, which can help to revert the original considerations in case of bad decisions; and d) to understand the underpinning reasons of decisions

and build on experience.

As a future work, other available AK knowledge sources will be integrated and alignment techniques will be employed for matching different metamodel/domain ontologies.

Acknowledgment

This work was supported by CONICET, ANPCyT, and UTN.

References

[1] Akerman A. and J. Tyree, *Using ontology to support development of software architectures*, IBM Syst. J. **45** (2006), 813-825.

[2] Babu L. T., M. Seetha Ramaiah, T.V. Prabhakar, and D. Rambabu, *ArchVoc–Towards an Ontology for Software Architecture*, in: *Proc. Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent*, SHARK/ADI '07: ICSE Workshops 2007, Minneapolis (2007), pp. 5-5.

[3] Bass, L., P. Clements, and R. Kazman, "Software Architecture in Practice: Third Edition," Addison-Wesley Professional, 2012.

[4] Capilla, R., A. Jansen, A. Tang, P. Avgeriou, and M.A. Babar, *10 years of software architecture knowledge management: Practice and future*, Journal of Systems and Software **116** (2016), 191-205.

[5] De Graaf, K. A., A. Tang, P. Liang, and H. Van Vliet, *Ontology-based Software Architecture Documentation*, in: *Proc. 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture* (WICSA/ECSA) (2012), pp. 121-130.

[6] Guessi, M., D. Moreira, G. Abdalla, F. Oquendo, and E. Nakagawa, *OntolAD: a formal ontology for architectural descriptions*, in: Proc. 30th Annual ACM Symposium on Applied Computing (SAC '15), ACM, New York, USA (2015), pp. 1417-1424.

[7] Gruninger M., and M.S. Fox, *Methodology for the Design and Evaluation of Ontologies*, in: *Proc. IJCAI Workshop on Basic Ontological in Knowledge Sharing*, Montreal, Canada (1995).

[8] Figueiredo, A. M., J. C. Dos Reis, M.A. Rodrigues, *Improving Access to Software Architecture Knowledge An Ontology-based Search Approach*, International Journal Multimedia and Image Processing (IJMIP) **2** (2012), pp. 124-149.

[9] ISO/IEC/IEEE Systems and Software Engineering - Architecture Description, ISO/IEC/IEEE 42010, pp. 1-46, 2011.

[10] Kruchten, P., P. Lago, and H. van Vliet, *Building up and exploiting architectural knowledge*, in: *Proc. QoSA?05: Second International Conference on Quality of Software Architectures*, Springer, Berlin/Heidelberg (2006), pp. 43-58.

[11] López, C., 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 (77) (2012), 66-80.

[12] Manteuffel, C., D. Tofan, H. Koziolek, T. Goldschmidt, and P. Avgeriou, *Industrial implementation of a documentation framework for architectural decisions*, in: *Proc. IEEE/IFIP Conference on Software Architecture* (WICSA'14)(2014) pp. 225?234.

[13] Marwat, M., S. Jan, and S.Z. A. Shah, *Software Architectural Design Ontology*, International Journal of Computer, Electrical, Automation, Control and Information Engineering **7** (2013), 1535-1538.

[14] Roldán, M. L., S. Gonnet, and H. Leone, *TracED: a tool for capturing and tracing engineering design processes*, Advances in Engineering Software **41** (2010), 1087-1109.

[15] Roldán, M. L., S. Gonnet, and H. Leone, *Captura del razonamiento y evolucin de arquitecturas de software mediante la aplicacin de operaciones arquitectnicas orientadas a objetivos*, in: *Proc. 44 Jornadas Argentinas de Informtica* ASSE 2015 (2015), pp. 299-313.

[16] Roldán, M. L., S. Gonnet, and H. Leone, *Operation-based approach for documenting software architecture knowledge*, Expert Systems, Londres **33** (2016), pp. 313-348.

[17] Roldán, M. L., S. Gonnet, and H. Leone, *SAKOnto: una ontologa de conocimiento sobre arquitecturas de software*, in: *Proc. Congreso Nacional de Ingeniera en Informtica/Sistemas de Informacin* CONAISII 2016, Salta (2016).

[18] Sun, H., H. Wang, and T. Hu, *Design Software Architecture Models using Ontology*, in: *Proc. 23rd International Conference on Software Engineering and Knowledge Engineering* (SEKE'11), Eden Roc Renaissance, Miami Beach, USA (2011).

[19] Tang, A., P. Avgeriou, A. Jansen, R. Capilla, and M. Ali, *A comparative study of architecture knowledge management tools*, Journal of Systems and Software **83** (2010), 352-370.

[20] Taylor, R. N., N. Medvidovic, and E. Dashofy, "Software Architecture: Foundations, Theory, and Practice," Wiley, 2009.

[21] Uschold, M., M. Gruninger, *Ontologies: Principles, Methods and Applications*, Knowledge Engineering Review **11**, 1996.

[22] van Heesch, U., P. Avgeriou, and R. Hilliard, *A documentation framework for architecture decisions*, Journal of Systems and Software **85** (2012), pp. 795-820.