2015-04

OntolAD: a formal ontology for architectural descriptions

Symposium on Applied Computing, 30th, 2015, Salamanca.
http://www.producao.usp.br/handle/BDPI/48982

OntolAD: a Formal Ontology for Architectural Descriptions

Milena Guessi[*]
Dept. of Computer Systems
University of São Paulo, São Carlos-SP, Brazil
milena@icmc.usp.br

Dilvan A. Moreira
Dept. of Computer Science
University of São Paulo, São Carlos-SP, Brazil
dilvan@icmc.usp.br

Gabriel Abdalla
Dept. of Computer Systems
University of São Paulo, São Carlos-SP, Brazil
gabriel.abdalla@usp.br

Flavio Oquendo
IRISA - University of South Brittany
Vannes, France
flavio.oquendo@irisa.fr

Elisa Yumi Nakagawa
Dept. of Computer Systems
University of São Paulo, São Carlos-SP, Brazil
elisa@icmc.usp.br

ABSTRACT

Architecture descriptions have been the focus of several studies in which they contribute for the design, evaluation, and evolution of software systems. In parallel, ontologies have been proposed for sharing and disseminating knowledge on a particular domain. In this scenario, the ontology proposed in the ISO/IEC/IEEE 42010 standard for architecture descriptions represents an important effort towards improving architecture descriptions as it establishes a common vocabulary. Nonetheless, a formal ontology for this standard could also support automatic conformance validation and enhance architectural descriptions reuse. However, a formal ontology for this standard is not available yet. Therefore, the main contribution of this paper is the proposal of OntolAD, a formal ontology expressed in OWL 2 for the ISO/IEC/IEEE 42010 standard. We demonstrate the feasibility of our formal ontology by applying it for describing the service-oriented architecture style (SOA). We conclude this study with interesting perspectives of using this ontology in future work.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms

Documentation

Keywords

Architecture Description, Formal Ontology

[*]This author is also with IRISA Research Group at University of South Brittany.

1. INTRODUCTION

Software architectures play a central role throughout a software system's life cycle. For example, during system conception, software architectures can ensure quality attributes, such as maintainability, dependability, and interoperability [23]. Later, during evolution, software architectures become the main protection against software system aging by preserving the architect's original intent regarding structure and behavior [17]. While the software architecture is embedded in the overall system structure, the Architecture Description (AD) is a tangible artifact expressing a software architecture [4, 13]. Among others, architecture descriptions can support reuse of architectural knowledge, assessment of architectural qualities, and communication of the software architecture to its stakeholders [5, 9, 15]. Thus, architecture descriptions greatly contribute for the success of software systems. Aiming to standardize the main constructs of architecture descriptions and disseminate best practices for their creation, the ISO/IEC/IEEE 42010 standard [13] was proposed.

In parallel, we observe the proposal of ontologies, which have been traditionally defined as an explicit specification of a conceptualization [11]. Overall, an ontology establishes a common understanding about a giving domain by providing an accurate and unambiguous communication of meaning which in turn promotes interoperability, reuse, and sharing [20]. Nonetheless, different kinds of ontology exist and they can vary from a set of precise, descriptive statements about the domain of interest to formal models [20]. For example, glossaries and taxonomies can be cited as examples of informal ontologies which are expressed in natural language or structured language. Conversely, FOAF[1], which describes human relationships, and Dublin Core[2], which describes documents, can be cited as examples of formal ontologies that may be expressed in description logic or first order logic. While an informal ontology can be useful for promoting the communication among different stakeholders, a formal ontology also provides the means for processing it using computer programs.

Motivated by the benefits brought by ontologies, several studies have investigated and proposed ontologies in the

[1]`http://xmlns.com/foaf/0.1/`
[2]`http://purl.org/dc/elements/1.1/`

Software Engineering field. Besides supporting communication, ontologies could be investigated for enhancing the interoperability between systems and enriching the specification of components, artifacts, and systems [20, 22]. For example, we can find ontologies addressing the software architecture field [2], architectural decisions [16], quality criteria [7], and architectural models [14]. In particular, the ISO/IEC/IEEE 42010 standard is an example of semi-formal ontology as it uses UML[3] for depicting a conceptual model of an ontology. As a consequence, there is still no support for automatically validating the conformance of architecture descriptions to the ISO/IEC/IEEE 42010 standard, which still relies on the experience of software architects.

The main contribution of this study is to propose a formal ontology, called OntolAD, for the ISO/IEC/IEEE 42010 standard. Besides contributing for the dissemination of current best practices for creating architecture descriptions, OntolAD can support the creation and validation of architecture descriptions that aim at complying to this standard. The rest of this paper is organized as follows. First, Section 2 introduces ontologies and further elaborates on the context where this study is developed. Section 3 presents OntolAD. Section 4 presents the quality assessment of OntolAD. Section 5 demonstrates how OntolAD can be applied for describing architecture description elements. Section 6 discusses our results and outlines interesting perspectives for future work. Finally, Section 7 summarizes our contributions.

2. BACKGROUND

Architecture descriptions are important for documenting, assessing, and sharing knowledge contained in software architectures. One of the main contributions of the ISO/IEC/IEEE 42010 standard is the definition of primitive constructs for architecture descriptions, such as stakeholders, concerns, viewpoints, and architecture models. Furthermore, this standard also specifies how more complex structures, such as architecture frameworks, architecture styles, and Architecture Description Languages (ADLs), could be specified in terms of these primitive constructs. For example, an architecture framework, which guides the creation of architectural descriptions for a given domain, such as "4+1" Views [15] and "Views and Beyond" [5], can be defined in terms of stakeholders, concerns, and viewpoints it selects. An architecture style, which encompasses a set of common restrictions to the form and structure of software architectures [8], can be understood as an architecture model in this standard. Finally, an Architectural Description Language (ADL), which is any form used for expressing a software architecture, such as Wright [1], UML, and SysML[4], can be specified in terms of provided model kinds, framed concerns, aimed stakeholders, and required correspondence rules.

In parallel, we observe that ontologies have been developed aiming to support communication, interoperability, and systems engineering [21]. In regards to communication, ontologies can promote a shared understanding on a particular domain by providing unambiguous and standardized definitions for terms and their relations. In regards to interoperability, we can use ontologies for integrating different tools, techniques, and platforms. For example, an ontology can either provide a standardized terminology for its different users to comply with or the semantic foundations for creating specific translators for its users. Finally, in regards to systems engineering, ontologies can enrich the specification of components, services, and artifacts by specifying constraints and relevant assumptions. In this sense, ontologies could be used together with ADLs, such as UML or SysML, for providing complementary information to architectural models.

As stated earlier, ontologies can present different formalism levels [20]. In this study, we focus on formal ontologies aiming to support consistency checking, model validation, and reuse. Several languages can be used for describing ontologies depending on the desired formalism level, expressiveness, available tools, and community support [6, 19]. We can cite RDFS[5] (Resource Description Framework Schema) and OWL[6] (Web Ontology Language) as examples of languages for expressing formal ontologies. Moreover, both of them are W3C standards for the Semantic Web. In this study, we selected OWL as it is the most expressive language of the two mentioned before. Other important factor contributing for this decision is the existence of ODM[7] (Ontology Definition Metamodel), an OMG standard bridging the gap between UML and OWL. Since UML is a broadly accepted standard for describing software architectures, it seems interesting to use OWL for expressing a formal ontology for the ISO/IEC/IEEE 42010 standard.

3. ONTOLAD OVERVIEW

In this study, we propose OntolAD[8], an ontology expressed in OWL 2 for architectural descriptions based on the ISO/IEC/IEEE 42010 standard. Therefore, this formal ontology describes the domain of interest in terms of classes, individuals, and properties about individuals, classes, or relationships that exist among them. The purpose of this ontology is to disseminate architectural descriptions and to define a baseline for integrating different architectural description elements, such as viewpoints, model kinds, and ADLs. We used the Protégé[9] tool for creating the ontology and the Pellet Reasoner Plug-in (v. 2.2.0)[10] for consistency checking and implicit knowledge inference.

The process we followed for creating OntolAD was mostly straightforward since the ISO/IEC/IEEE 42010 standard already provides detailed definitions for all architecture description elements. The definition of each class uses axioms for determining membership conditions to the class [12]. First, we transformed primitive AD constructs into OWL primitive classes, i.e., classes with only necessary conditions. OWL can express three different types of properties, namely: (i) object properties, which define relationships between individuals; (ii) data type properties, which define relationships between individuals and literals (e.g., strings, integers, etc.); and (iii) annotation properties, which can be used to describe metadata about individuals, classes, and properties,

[3]http://www.omg.org/spec/UML/2.3/
[4]http://www.omgsysml.org/

[5]http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
[6]http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
[7]http://www.omg.org/spec/ODM/1.0/
[8]This ontology is available at https://www.dropbox.com/sh/58ro0giwi821khw/AACoMg4ZzjJuB8zClAbxg5awa?dl=0
[9]http://protege.stanford.edu
[10]http://clarkparsia.com/pellet/protege/

such as translations, comments, and definitions for these concepts. Many relationships described in the ISO/IEC/-IEEE 42010 standard are implemented as object properties in OntolAD, such as *has concern* between `Stakeholder` and `Concern`, *is governed by viewpoint* between `Architecture View` and `Architecture Viewpoint`, and *is expressed by* between `Architecture` and `Architecture Description`. To simplify our ontology, we did not provide either domain or range for object properties. Furthermore, we also identified data type properties, such as *author*, *issue date*, *version* of a particular `Architecture Description`. Currently, OntolAD has 47 object and data type properties.

In a second step, we identified which primitive classes should be transformed into defined classes, i.e., classes with at least one set of necessary and sufficient conditions. In other words, a defined class is "equivalent to" the set of sufficient and necessary conditions. Defined classes are important since the reasoner only automatically classifies classes that are defined. To do so, we analyzed the recommendations of the standard. Even though ISO/IEC/IEEE 42010 standard is a comprehensive semi-formal ontology, some design decisions had to be made in order to create OntolAD. For example, a correspondence is a relationship among architecture description elements. To map this concept into OntolAD we could either define it as a class or as an object property between individuals. Finally, we decided to define this concept as a class since this would enable creating correspondences that can be reused in other contexts.

In a third step, we added disjoint, closure, and covering axioms to OntolAD because of the open world assumption, i.e., if something is not stated does not implies it is not true. While not stated in the standard, these axioms are required for the reasoner to function properly. As OWL assumes that classes overlap unless there is an explicit statement to the contrary, we added disjoint axioms indicating which classes are disjoint. Then, closure axioms (i.e., `only`) were added to all existential restrictions (i.e., `some`) in mandatory clauses of the ISO/IEC/IEEE 42010 standard (i.e., *shall* or *must*). Conversely, recommendation clauses in the standard (i.e., *should*) have universal restrictions (i.e., `only`). Finally, covering axioms indicate allowed subclass memberships. For example, a covering axiom was included in the description of the class `Architecture Description Element` for indicating that any individual of this class can only pertain to one of its subclasses. Table 1 shows the final formal description of this class in OWL. On the other hand, there is no covering axiom in the description of class `Stakeholder`, shown in Table 2, as any individual of this class can belong to several of its subclasses (e.g., acquirer, builder, developer, user, supplier, among others) depicting the several roles a particular stakeholder can play. Moreover, the operator `and` defines a class resulting from the intersection of each restriction whereas the operator `or` defines a class resulting from the union.

Throughout the definition of OntoLAD, domain experts contributed for validating the ontology's consistency with the ISO/IEC/IEEE 42010 standard. This task was supported by Protégé as it automatically generates a graphical representation for the asserted class hierarchy depicted in Figure 1. An universal class named `Thing`, which contains all individuals, is automatically created by the language. Moreover, OWL 2 also provides an empty class named `Nothing`. The later is particularly useful for checking the consistency of the ontology. For example, the ontology might be inconsistent if `Nothing` is classified by the reasoner as one of the defined classes.

Table 1: Formal description of an architectural description element in OntolAD

Equivalent to
'Architecture Decision' or 'Architecture Description' or ADL or 'Architecture Framework' or 'Architecture Model' or 'Architecture Rationale' or 'Architecture View' or 'Architecture Viewpoint' or Concern or Correspondence or 'Model Kind' or Stakeholder
Disjoint with
System, Environment, Architecture

Table 2: Formal description of a stakeholder in OntolAD

Equivalent to
'Architecture Description Element' and (*hasConcern* some Concern) and (*isInterestedIn* some System) and (*hasConcern* only Concern) and (*isInterestedIn* only System)
Disjoint with
'Architecture Decision' or 'Architecture Description' or ADL or 'Architecture Framework' or 'Architecture Model' or 'Architecture Rationale' or 'Architecture View' or 'Architecture Viewpoint' or Concern or Correspondence or 'Correspondence Rule' or 'Model Kind' or Stakeholder

Alternatively, Figure 2 shows a graphical representation for the inferred class hierarchy that is computed by the reasoner. The inferred class hierarchy shows implicit knowledge embedded in the ontology, i.e., assertions that were not originally included but are nonetheless true. For example, the inferred class hierarchy classified `Architecture Description` under `Architecture Framework`. According to the note in Clause 6.2 of the standard, an architecture description could adhere to none, single, or multiple architecture frameworks. In this sense, the reasoner infers that an architecture description uses and extends from a particular instance of an architecture framework, which can possibly be unique in the domain. Similarly, an `ADL` is classified under `Architecture Viewpoint`. According to the note in Clause 6.3 of the standard, an ADL can define none, single, or multiple architecture viewpoints. In this sense, a given ADL can be said to comply with an architecture viewpoint even if this viewpoint is unique in the domain. Therefore, the assertions made by the reasoner are still in conformance with the standard.

According to the axioms and properties of OntolAD it has expressiveness $\mathcal{SIF}(\mathcal{D})$ which means:

$\mathcal{S}$: An abbreviation for $\mathcal{ALC}$ (where $\mathcal{AL}$ stands for attributive language and $\mathcal{C}$ stands for complex concept negation) with transitive properties. For example, the transitive property *isUsefulFor* can relate an individual of the class `Architecture` with an individual of the class `System` and an individual of the class `'Architecture Description'` with an individual of the class `Architecture`. Thus, *isUsefulFor* can be used to infer the

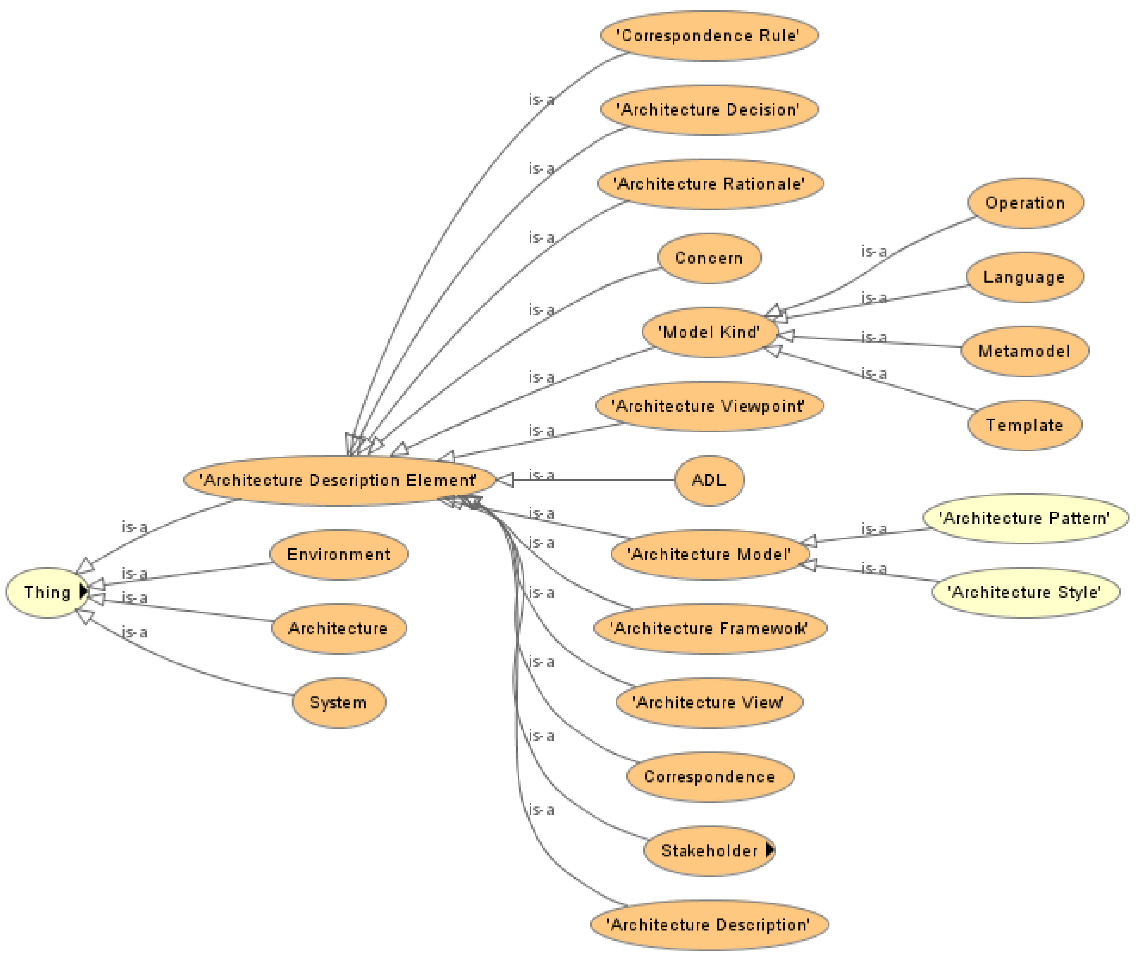

Figure 1: OntolAD asserted class hierarchy. Dark orange ellipses indicate defined classes, whereas light orange ellipses indicate primitive classes

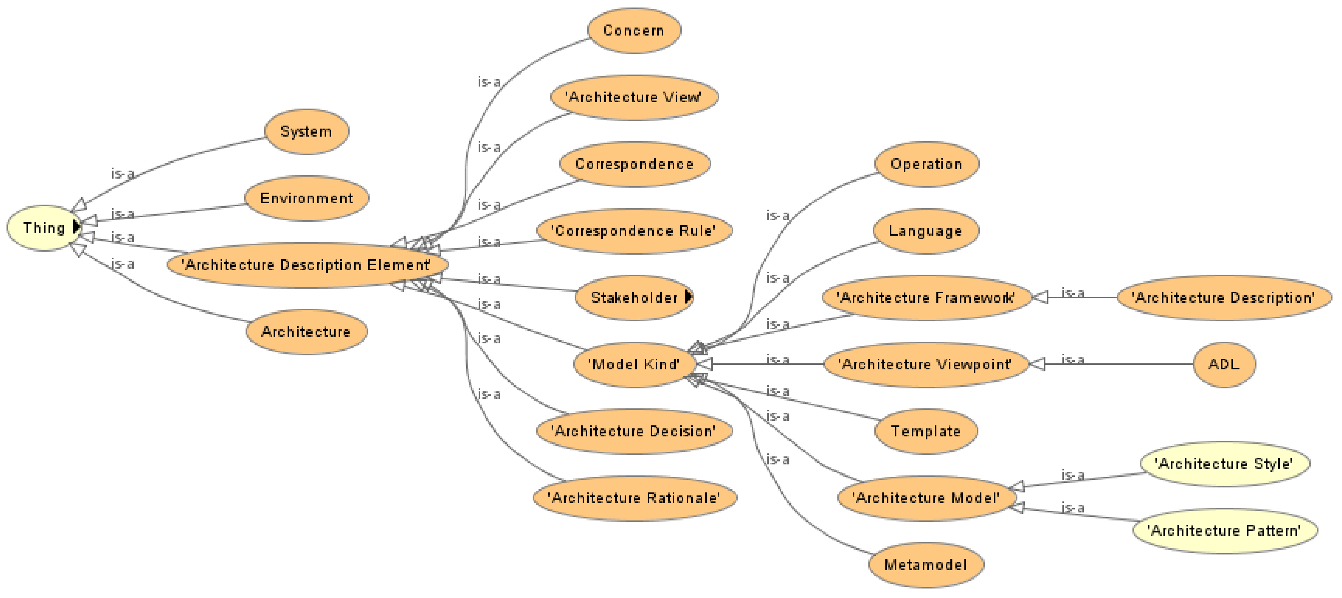

Figure 2: OntolAD inferred class hierarchy. Dark orange ellipses indicate defined classes, whereas light orange ellipses indicate primitive classes

relation between an individual of the class `'Architecture Description'` with an individual of the class `System`;

$\mathcal{I}$: Inverse properties. For example, the property *hasConcern*, which relates individuals of the class `Stakeholder` with individuals of the class `Concern`, is the inverse of *isConcernOf*, which relates individuals of the class `Concern` with individuals of the class `Stakeholder`;

$\mathcal{F}$: Functional properties can have at most one value for each individual. For example, the property *hasView* relates each individual of the class `'Architecture Viewpoint'` to at most one individual of the class `'Architecture View'`; and

$(\mathcal{D})$: Use of data type properties, data values, or literals. For example, the data properties *hasTemplate* and *hasLanguage* relate individuals of the classes `Template` and `Language` to literals respectively.

4. QUALITY ASSESSMENT OF ONTOLAD

The evaluation of ontologies plays an important role in improving their efficiency and effectiveness. In particular, the overall quality of ontologies can be assessed in regards to four different dimensions [3]:

- Syntactic quality: related to how the ontology was written;
- Semantic quality: related to the absence of contradictory concepts;
- Pragmatic quality: related to the ontology's content and usefulness for users, independently of its syntax and semantics; and,
- Social quality: related to the acceptance of the ontology by the community.

Each of these dimensions have their own associated metrics. These metrics could be either absolute (i.e., varying between zero and one) or relative (e.g., showing an average value). As a consequence, before calculating the overall quality of OntolAD, relative metrics will be normalized for fitting in the range between zero and one. Moreover, we also assume that all dimensions and metrics have equal importance to the overall quality of OntolAD.

First, Syntactic (S) quality can be measured by an equally weighted function composed by: (i) lawfulness (SL), which indicates correctness of the syntax; and (ii) richness (SR), which indicates the proportion of OWL 2 features that were actually used in OntolAD. In regards to the first metric, we can assume that SL is 1 since the syntax of OntolAD was automatically validated by Protégé. Moreover, according to the set of metrics automatically calculated by Protégé, OntolAD presents 14 of the 35 possible axioms types including class, individual, and properties. Thus, SR is equal to 0.4 (i.e., 14/35) in our ontology.

Second, Semantic (E) quality can be measured by an equally weighted function composed by: (i) interpretability (EI), which is the meaningfulness of the terms used; (ii) consistency (EC), which indicates if the terms have a consistent meaning throughout the ontology; and (iii) clarity (EA), which reflects the average number of different meanings for the terms used. In order to calculate EI, we calculated the proportion of the terms used by OntolAD that also had a definition listed in WordNet[11], a lexical database for the English language. As WordNet only recognizes single words, we analyzed separately each part of a composite name for classes, data properties, and object properties. From a total of 65 unique terms used in OntolAD 63 of them were found in WordNet which resulted in EI being equal to 0.97 (i.e., 63/65). Since we frequently checked the inferred class hierarchy using the reasoner, we assume that OntolAD has EC equal to 1. For calculating EA, we divided the quantity of terms used with only one meaning in WordNet by the quantity of terms used in OntolAD that are listed in WordNet. As a result, EA is equal to 0.13 (i.e., 8/63) in our ontology.

Third, Pragmatic (P) quality is related to the content and usefulness of OntolAD. In this sense, this quality can be measured by an equally weighted function composed by: (i) accuracy (PU), which is the proportion of axioms defined in the ontology that are true in the standard; (ii) relevance (PR), which indicates the degree to which the ontology provides information that may be useful for different applications; and (iii) comprehensiveness (PO), which is a measure of the size of the ontology. We assume that OntolAD has PU equal to 1 since it was subjected to several discussions and revisions with domain experts during its construction and revision. In order to calculate PO, we compared OntolAD with the average class and properties size of the ontologies aggregated by the Linked Open Vocabularies (LOV) endpoint[12]. OntolAD has 35 classes and 47 properties while an average ontology has 41 classes an 58 properties. Thus, PO is equal to 0.83 (i.e., 82/99). Since OntolAD is still recent, its PR is still undetermined.

Finally, Social quality (O) can be measured by an equally weighted function composed by: (i) authority (OT), which is related to the number of other ontologies linking to OntolAD; (ii) and history (OH), which indicates the number of times the ontology was accessed. Since OntolAD have been recently introduced, there is still no data available for directly assessing this dimension. Despite that, we anticipate that OntolAD has a good potential as it builds upon the ISO/IEC/IEEE 42010 standard, which is broadly recognized by the Software Architecture community. Moreover, we intend to contribute for the social quality of OntolAD by making it permanently and freely available.

Therefore, the overall quality (Q) of OntolAD can be represented by an equally weighted function composed by its syntactic, semantic, and pragmatic qualities. Table 3 summarizes the results obtained for each dimension. This evaluation highlighted strong aspects of OntolAD, such as its lawfulness, consistency, and comprehensibility. In this sense, the participation of experienced software architects and ontologists in the construction and revision of OntolAD was certainly important for its semantic and pragmatic quality. Nonetheless, these results also point out directions where OntolAD needs improvement, such as clarity and relevance. Since our ontology is based on the ISO/IEC/IEEE 42010 standard, its clarity should be improved by complying with this standard's terminology. Moreover, relevance should improve with community feedback and new applications for

[11]`http://wordnetweb.princeton.edu/perl/webwn`
[12]`http://lov.okfn.org/endpoint/lov_aggregator`

Table 3: Result of quality assessment

Metric	Dimension	Description	Result
Syntax (S)	Lawfulness (SL)	Percentage of correct syntax	1.00
	Richness (SR)	Percentage of available syntax used	0.40
	Total	1/2 SL + 1/2 SR	**0.70**
Semantic (E)	Interpretability (EI)	Percentage of terms used that exist in WordNet	0.97
	Consistency (EC)	Consistent meaning of terms throughout the ontology	1.00
	Clarity* (EA)	Average precision of words in OntolAD	0.13
	Total	1/3 EI + 1/3 * EC + 1/3 * EA	**0.69**
Pragmatic (P)	Comprehensibility (PO)	Size in comparison to other ontologies	0.83
	Accuracy (PU)	Correspondence of terms in OntolAD to ISO/IEC/IEEE 42010	1.00
	Total	1/2 PO + 1/2 PU	**0.92**
Overall Quality (Q)		Q = 1/3*S + 1/3*E + 1/3*P	**0.76**

*0 represents extreme ambiguity and 1 represents no ambiguity

OntolAD.

5. CASE STUDY: DESCRIBING AN ARCHITECTURE STYLE IN ONTOLAD

In this section, we investigate the potential of OntolAD[13] for creating architecture description elements in conformance to the ISO/IEC/IEEE 42010 standard. In particular, we select the SOA (Service-Oriented Architecture) as it is an important and widespread style in our domain. This architecture style defines a loosely coupled structure of multiple autonomous services [10]. Instead of having clients accessing services directly (as in a client-server architecture style), SOA provides a broker to mediate the communication between services and clients. In this sense, it is essential that services and clients agree upon the communication protocol, which basically comprises the service's name, location, and exchanged data requirements used by the broker.

A new defined class named `SOA` was added to OntolAD for describing this style. We chose to describe it as a class and not an individual since we understand that this style can be referenced by several concrete architectural descriptions in our domain. By defining `SOA` under `Architecture Style`, we inherit all conditions that `SOA` must comply with. Furthermore, `SOA` must also comply with the conditions of an `Architecture Model`, i.e., it must have at least one relationship *frames* to an individual of the class `Concern` and at least one relationship *is governed by model kind* to an individual of the class `Model Kind`. In OntolAD, a `Model Kind` could be expressed by a `Language`, a `Metamodel`, an `Operation`, or a `Template` with at least one relationship `frames` to an individual of the class `Concern`. In this case study, we only created a new class for the `SOA Metamodel`. Nonetheless, SOA style could also be realized by other model kinds.

Next, we created individuals for the classes required for the definition of `SOA` and `SOA Metamodel` classes. This activity can also be understood as the instantiation of the classes defined in the ontology. Relevant concerns addressed in SOA are [10]:

- Autonomy: related to minimal or absent dependence of one service to others;
- Composability: related to the ability of being composed into a larger structure, possibly providing a more complex service;
- Reusability: related to the ability of being reused in different contexts;
- Statelessness: related to minimal or absent dependence of state related information of the client or other services; and
- Discoverability: related to the existence of an external description that can ease the discovery of a service in a search mechanism.

According to OntolAD, a metamodel must have at least one *has entity* relationship to some `Entity` individual and a *has relationship* to some `Relationship` individual. The individuals created for the class `Entity` encompass [10, 18]: (i) Service Provider, which is the element that publishes a service; (ii) Service Client, which is the element that requests a service; and (iii) Service Broker, which is the element that permits service discovery. Meaningful relationships among services in the SOA style can be summarized as *Communicate*, *Listed In*, *Register*, and *Sign Up*. Finally, we also added some data type properties (i.e., relationships between individuals and literals) for these individuals. For example, the Service Provider has an attribute named "Communication Protocol" indicating the communication standard or vocabulary. Table 4 shows the final formal description of the SOA style while Table 5 shows the description of the SOA metamodel. Figure 3 shows a fragment of OntolAD with the new classes added.

Table 4: Formal description of SOA style in OntolAD

Equivalent to
'Architecture Style' and (frames some Autonomy, Reusability, Composability, Statelessness, Discoverability) and (frames only Autonomy, Reusability, Composability, Statelessness, Discoverability) and (isGovernedByModelKind some SOAMetamodel) and (isGovernedByModelKind only SOAMetamodel)

[13]The ontology referred in this study case is also available at `https://www.dropbox.com/sh/58ro0giwi821khw/AACoMg4ZzjJuB8zClAbxg5awa?dl=0`

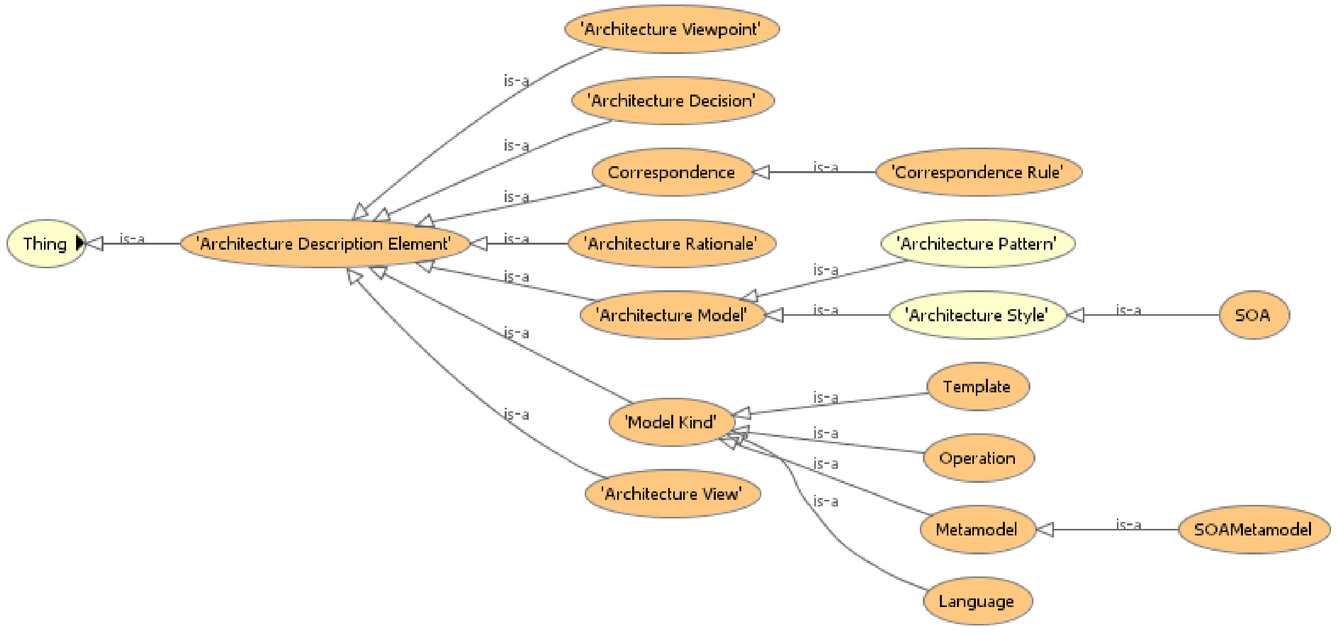

Figure 3: OntolAD showing SOA style asserted class hierarchy

Table 5: Formal description of SOA metamodel in OntolAD

Equivalent to
Metamodel and (hasEntity some ServiceProvider,ServiceBroker,ServiceClient) and (hasRelationship some SignUp,Register,Communicate,ListedIn) and (hasEntity only ServiceProvider,ServiceBroker,ServiceClient) and (hasRelationship only SignUp,Register,Communicate,ListedIn)

6. DISCUSSION AND FUTURE PERSPECTIVES

Ontologies constitute an important research line in the Software Architecture field. Besides exploring ontologies for disseminating knowledge in this domain, there are several applications where ontologies can be useful. For example, ontologies could be investigated for integrating different tools, techniques, and platforms. In addition, ontologies can provide richer specifications of components, services, and artifacts, including both documents and models. Our study aims at contributing to the later by providing a formal ontology expressed in OWL 2 for the ISO/IEC/IEEE 42010 standard that could be processed and analyzed with software tools.

Using OWL 2 to express a formal ontology for architectural descriptions brings additional advantages: (i) it is a comprehensive approach for describing architecture descriptions, which could be extended and reused; (ii) it can be used as an interchange format between ADLs, viewpoints, and frameworks; (iii) it can be processed by both machines and humans; and (iv) it is not bound to a particular platform, technology, or technique.

The case study presented in Section 5 demonstrates how OntolAD could be used in practice to describe an architecture style in conformance to the ISO/IEC/IEEE 42010 standard. Because SOA is by itself a comprehensive topic in the Software Architecture domain, it was out of the scope of this paper to further detail this architecture style. Nonetheless, this case study showed that an architecture description element could be created using this ontology. Motivated by this preliminary result, we anticipate four main perspectives for advancing this ontology in future works:

- Addition of new classes for describing well-known architecture styles, patterns, frameworks, and ADLs aiming to create a broader body of knowledge;
- Addition of new individuals aiming to populate the ontology and establish concrete relationships among them. This knowledge will help to clarify the different roles played by stakeholders and, as a consequence, indicate how the architecture description could be tailored for a given audience;
- Application of OntolAD to describe concrete architecture descriptions. This study will provide more evidences on the use of this ontology for exposing breaches in the design with the assistance of reasoners;
- Integration of OntolAD into modeling tools for supporting automatic validation and consistency checking.

Regarding limitations of this study, it is important to notice that the creation of OntolAD required additional assertions from the developers. We documented our assertions and design decisions in the description of the classes aiming to gather the community feedback for validating its construction and, as a result, improving OntolAD's social quality.

7. CONCLUSIONS

Several informal and semi-formal ontologies have been proposed in the Software Architecture field mostly aiming to enhance knowledge sharing. But, formal ontologies should also be explored in this domain aiming to provide the means for transforming current architecture descriptions into models that can be automatically processed, assessed, and reused. This study presents a first step towards formalizing architecture descriptions of software systems based on the recommendations of the ISO/IEC/IEEE 42010 standard. Moreover, our ontology called OntolAD is expressed in OWL 2. Besides demonstrating the feasibility of OntolAD in a case study, where it was used for expressing an architecture style, we also outline several research lines where OntolAD can be of assistance. For example, this formal ontology can be used for expressing all kinds of architecture description elements, such as architecture frameworks, patterns, and viewpoints. In future work, we intend to continue investigating how OntolAD can be integrated to current modeling tools for supporting the creation and validation of architecture descriptions. As a result, we expect that OntolAD could support both practitioners and researchers in creating architecture descriptions that comply with current best practices.

Acknowledgments

This study is supported by grant 2012/24290-5, São Paulo Research Foundation (FAPESP). Furthermore, the Protégé resource used in this study is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

References

[1] R. J. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon University, Maio 1997.

[2] T. L. Babu, M. S. Ramaiah, T. V. Prabhakar, and D. Rambabu. ArchVoc: Towards an Ontology for Software Architecture. In *SHARK/ADI'2007*, pages 1–5, Minneapolis, USA, 2007.

[3] A. Burton-Jones, V. C. Storey, V. Sugumaran, and P. Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering* , 55(1):84 – 102, 2005.

[4] P. Clements. A survey of architecture description languages. In *IWSSD' 1996*, pages 1–10, Germany, 1996.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2 edition, 2011.

[6] O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data & Knowledge Engineering*, 46:41–64, 2003.

[7] R. C. de Boer, P. Lago, A. Telea, and H. van Vliet. Ontology-driven Visualization of Architectural Design Decisions. In *(WICSA/ECSA'2009)*, pages 51–60, Cambridge, UK, 2009.

[8] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274, 1995.

[9] D. Garlan and M. Shaw. An Introduction to Software Architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Company, 1993.

[10] H. Gomaa. *Software Modeling Design*. Cambrigde University Press, 2011.

[11] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[12] M. Horridge. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Technical report, The University of Manchester, 2011. `http://130.88.198.11/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf`.

[13] ISO. ISO/IEC/IEEE 42010 - Systems and Software Engineering — Architecture Description, December 2011.

[14] H. H. W. Jing Sun and T. Hu. Design Software Architecture Models using Ontology. In *SEKE'2011*, pages 1–6, Miami, USA, 2011.

[15] P. Kruchten. Architectural Blueprints - The "4+1" View Model of Software Architecture. *IEEE Software*, 12(6):42–50, 1995.

[16] P. Kruchten. An ontology of architectural design decisions in software intensive systems. In *II Groningen Workshop on Software Variability*, pages 54–61, 2004.

[17] P. Kruchten, H. Obbink, and J. Stafford. The Past, Present and Future of Software Architecture. *IEEE Software*, 23(2):22–30, 2006.

[18] M. P. Papazoglou and W.-J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[19] J. R. G. Pulido, M. A. G. Ruiz, R. Herrera, E. Cabello, S. Legrand, and D. Elliman. Ontology languages for the semantic web: A never completely updated review. *Knowledge-Based Systems*, pages 489–497, 2006.

[20] M. Uschold. Building ontologies towards a unified methodology. In *Expert Systems' 1996*, Cambridge, UK, 1996. `http://www.aiai.ed.ac.uk/project/pub/documents/1996/96-es96-unified-method.pdf` (09/14/2014).

[21] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2), 1996. Also available as AIAI-TR-191 from AIAI, The University of Edinburgh.

[22] W3C. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. [*On-line*], 2006.

[23] A. I. Wasserman. Towards a discipline of software engineering. *IEEE Software*, 13(6):23–31, 1996.