SSQSA Ontology Metrics Front-End

MILOŠ SAVIĆ, ZORAN BUDIMAC, GORDANA RAKIĆ AND MIRJANA IVANOVIĆ, University of Novi Sad
MARJAN HERIČKO, University of Maribor

SSQSA is a set of language independent tools whose main purpose is to analyze source code of software systems in order to evaluate their quality attributes. The aim of this paper is to present how a formal language that is not a programming language can be integrated into the front-end of the SSQSA framework. Namely, it is explained how the SSQSA front-end is extended to support OWL2 which is a domain-specific language for the description of ontological systems. Such extension of the SSQSA front-end represents a step towards the realization of a SSQSA back-end which will be able to compute a hybrid set of metrics that reflect different aspects of complexity of ontological descriptions.

Categories and Subject Descriptors: D.2.8 [**Software Engineering**]: Metrics – *Complexity measures*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *Representation languages*

General Terms: Languages, Measurement

Additional Key Words and Phrases: OWL2, Ontology metrics, Complexity, SSQSA, eCST representation

1. INTRODUCTION

With the rise of the semantic web, ontologies have become a key technology to provide formal description of shared and reusable knowledge. Viewed as "explicit specification of conceptualization" [Gruber 1993], ontologies are used to define concepts and relations present in a domain in order to support reasoning, integration, and aggregation of data by autonomous software agents. Since real-world ontologies rapidly increase in size, it has become highly important to measure, evaluate and understand their complexity, in order to be able to control their maintenance and evolution.

SSQSA is a set of language-independent tools that statically analyze software systems in order to evaluate their quality attributes [Budimac et al. 2012]. The whole framework is organized around the enriched Concrete Syntax Tree (eCST) representation of source code [Rakić and Budimac 2011b]. The motivation for this work was to explore the possibility to use the eCST representation to compute metrics which reflect the complexity of ontological descriptions. In order to obtain the eCST representation of ontology, the SSQSA front-end has to be extended to support a language for the description of ontological systems. The aim of this paper is to explain how the SSQSA front-end is extended to support OWL2 language in functional-style syntax.

The rest of the paper is structured as follows. The next section presents the related work. Section 3 covers the integration of OWL2 into the SSQSA framework. In the next section are discussed the benefits of the eCST representation of ontology. The last section concludes the paper and gives directions for future work.

This work was partially supported by the Serbian Ministry of Education, Science and Technological Development through project *Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support*, no. OI174023. The authors also would like to thank Rok Žontar for fruitful discussions on ontology metrics.
Author's address: M. Savić, Z. Budimac, G. Rakić, M. Ivanović, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, email: {svc, zjb, goca, mira}@dmi.uns.ac.rs; M. Heričko, Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ulica 17, 2000 Maribor, Slovenia, email: marjan.hericko@uni-mb.si.

In: Z. Budimac (ed.): Proceedings of the 2nd Workshop of Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), Novi Sad, Serbia, 15.-17.9.2013, published at http://ceur-ws.org

2. RELATED WORK

2.1 Ontology metrics

In recent years, various metrics for measuring the complexity of ontological descriptions were proposed. Inspired by Chidamber and Kemerer [1994] metrics suite, Yao et al. [2005] proposed three cohesion metrics which are defined on a graph that represent subsumtion dependencies between ontological concepts. Orme at al. [2006] introduced three coupling metrics which are defined on the graph representation of ontology. Tartir et al. [2005] introduced OntoQA metric suite that contains 12 structural metrics also defined on ontological graph. Zhang et al. [2010] also proposed several new graph-based structural metrics for ontology evaluation. Their metrics suite, among others, contains metrics adopted from the Chidamber-Kemerer suite (NOC, DIT, CBO). Žontar and Heričko [2012] analyzed software metrics from the Lorenz-Kidd, Chidamber-Kemerer and Abreu metric suites in order to determine which of them can be adopted for ontologies. The results of their study show that graph-based software metrics can be adopted for ontology evaluation.

2.2 SSQSA Framework

The SSQSA framework consists of two parts, SSQSA front-end also known as eCST Generator, and the set of SSQSA back-ends, individual tools that operate on the eCST representation of source code. The main characteristic of eCST representation is that it contains so called universal nodes, language-independent markers that denote the meaning of concrete language constructs. The architecture of SSQSA is presented in Figure I. Also, it is shown how the architecture is planned to be extended with a new back-end in order support the analysis and evaluation of ontological systems.

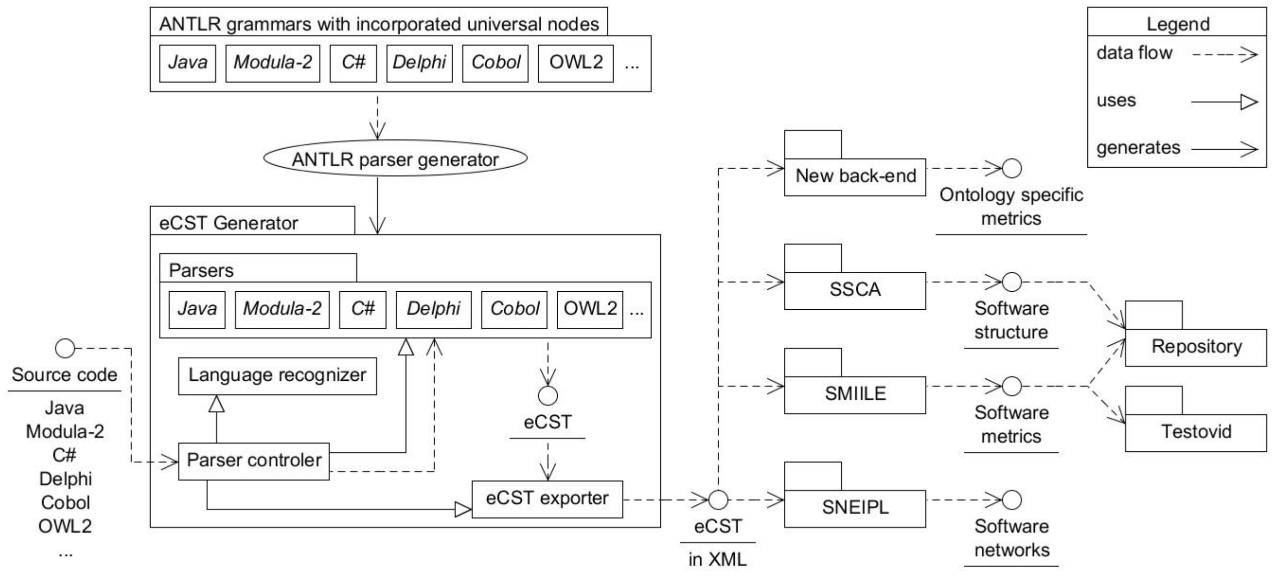

Figure I. Extended SSQSA architecture

SSQSA originated from a language-independent software metrics tool SMIILE [Rakić and Budimac 2011a]. SMIILE uses the eCST representation to calculate metrics reflecting internal complexity of software entities such as LOC and Cyclomatic complexity [McCabe 1976]. It is also integrated with Testovid, a semi-automated assessment system for students' programs, in order to provide metric-based qualification of programming assignments [Pribela et al. 2012]. SSCA was the first SSQSA back-end which extended the applicability of the eCST representation [Gerlec et al. 2012]. This tool tracks and analyzes changes in the hierarchical structure of software entities and stores its results in a repository that also contain metric values obtained using SMIILE. The last realized SSQSA back-end is SNEIPL [Savić et al. 2012]. This tool extracts dependency networks formed by software entities that can be used to analyze the design complexity of software systems under the framework of complex network theory. Obtained networks can be also viewed as fact-bases required for reverse engineering activities and used to calculate metrics related to software design.

Currently SSQSA supports six general-purpose, imperative programming languages: Java, C#, Delphi, Modula-2, Pascal and Cobol. Therefore, this work is the first attempt to extend the SSQSA front end to produce the eCST representation of a declarative, domain-specific language.

3. INTEGRATION OF OWL2 LANGUAGE INTO THE SSQSA FRONT-END

eCST Generator uses parsers generated by the ANTLR [Parr and Quong 1995] parser generator to produce the eCST representation of source code that is provided as input. The advantage of using ANTLR to describe languages supported by SSQSA is the ANTLR grammar notation itself. This notation enables modification of syntax trees through tree rewrite rules that are attached to grammar productions. Therefore, in order to integrate OWL2 into the SSQSA front-end the following steps have to be made:

1. Realization of ANTLR grammar which describes OWL2 FSS,
2. Identification of OWL2 language constructs that corresponds to existing eCST universal nodes,
3. Incorporation of eCST universal nodes into tree-rewrite rules of the grammar in order to obtain eCST representation of parsed text.

3.1 Step 1 – ANTLR grammar for OWL2 FSS

The formal specification of OWL2 FSS in Extended Backus-Naur form (EBNF) can be found in the official W3C OWL2 language specification [Motik et al. 2012]. The ANTLR grammar notation closely follows EBNF, thus the grammar in [Motik et al. 2012] can be easily adopted for ANTLR. At this stage of the integration, the realized grammar is tested using ten ontologies from TONES[2] repository which are previously converted into OWL2 FSS using Protégé[3]. The results are summarized in Table I. It can be seen that the parser generated from the grammar successfully parsed more than 1.4 millions of lines of real-world ontological axioms in less than three minutes.

Table I. Results of testing of ANTLR grammar for OWL2 FSS using ontologies from TONES repository.

ONTOLOGY NAME	LOC	Parse time [s]
CTON (Cell Type Ontology)	144252	23
FMA (Foundational Model of Anatomy)	316101	41
Gene Ontology Edit	233608	22
Human Disease	476111	59
Teleost Taxonomy	182656	17
GEO Skills	20506	2
Matr Mineral	46	0.04
OBO Relation Ontology	25412	2
SC Ontology	23707	2
Software Ontology	5931	0.5

3.2 Step 2 – Universal nodes

OWL2 FSS language contains four types of tokens: keywords, separators, identifiers and constants. For each of mentioned lexical categories there are already introduced eCST universal nodes. Ontological axioms are marked with STMT universal node which is used to mark individual statements in imperative

[2] http://owl.cs.manchester.ac.uk/repository/
[3] http://protege.stanford.edu/

programming languages. Elements of an axiom are also marked with existing universal nodes (TYPE, ARGUMENT_LIST, and ARGUMENT). The PACKAGE_DECL universal node denotes that entities declared in an eCST sub-tree rooted at this node are mutually visible. Therefore, PACKAGE_DECL corresponds to the declaration of ontology. Declarations of ontological entities (concepts, roles and individuals) are marked with ATTRIBUTE_DECL universal node which is used to denote declarations of global variables in imperative programming languages. Ontological expressions that can be nested (class and data range expressions) are marked with the EXPR universal node.

OWL2 is a declarative, domain-specific language. Before the integration of OWL2, SSQSA supported several programming languages none of them being declarative or domain-specific. OWL2 axioms represent explicitly stated relations among ontological entities. Therefore, we introduced three new universal nodes that denote different categories of explicitly stated relations in general:

1. BINARY_RELATION (BR) marks binary relations
2. SYMMETRIC_RELATION (SR) marks symmetric n-ary relations
3. PARTIALLY_KNOWN_BINARY_RELATION (PKBR) marks binary relations in which one of the arguments is not known at the moment.

With BINARY_RELATION are marked all OWL2 relations that denote subsumptions and assertions. The SYMMETRIC_RELATION universal node is associated with relations indicating the equivalent and disjoints classes, same and different individuals, and equivalent and disjoint object properties. The PARTIALLY_KNOWN_BINARY_RELATION universal node marks object property domain and object property range relations. The newly introduced universal nodes are currently used only in the eCST representation of ontological descriptions. However, they can be used to mark explicitly stated binary and symmetric relations in other descriptive languages as well. Explicitly stated relations among entities in already supported imperative programming languages are marked with specific, more concrete universal nodes, such as EXTENDS and IMPLEMENTS. Those universal nodes can be viewed as sub-concepts of the BINARY_RELATION universal node.

3.3 Step 3 – Tree-rewrite rules

Once the correspondence between constructs of a concrete language and eCST universal nodes is identified, it is pretty straightforward to incorporate universal nodes into tree rewrite rules of the grammar. For example, it has been identified that ontology declarations correspond to the PACKAGE_DECL universal node. Therefore, PACKAGE_DECL universal node will be incorporated in the tree rewrite rule of the production that describe ontology declaration as the following excerpt from the OWL2 FSS grammar shows:

```
ontology :  'Ontology'  '(' (ontologyIRI versionIRI?)? importo* annotation* axiom* ')'
            -> ^(PACKAGE_DECL
                    ^(KEYWORD 'Ontology')
                    ^(SEPARATOR '(')
                    (ontologyIRI versionIRI?)?
                    importo* annotation* axiom*
                    ^(SEPARATOR ')')
                );
```

Besides the PACKAGE_DECL universal node, two other universal nodes are also incorporated in the rule: KEYWORD and SEPARATOR to mark keywords and separators in ontology declaration, respectively.

Figure II shows how a simple ontology named “PL” looks in the eCST representation. The complete description of the ontology in the functional-style syntax is as follows:

```
Ontology (:PL
     SubClassOf(:C :CPP)
)
```

The `SubClassOf` axiom states that each program written in the programming language C is at the same time valid C++ program.

`

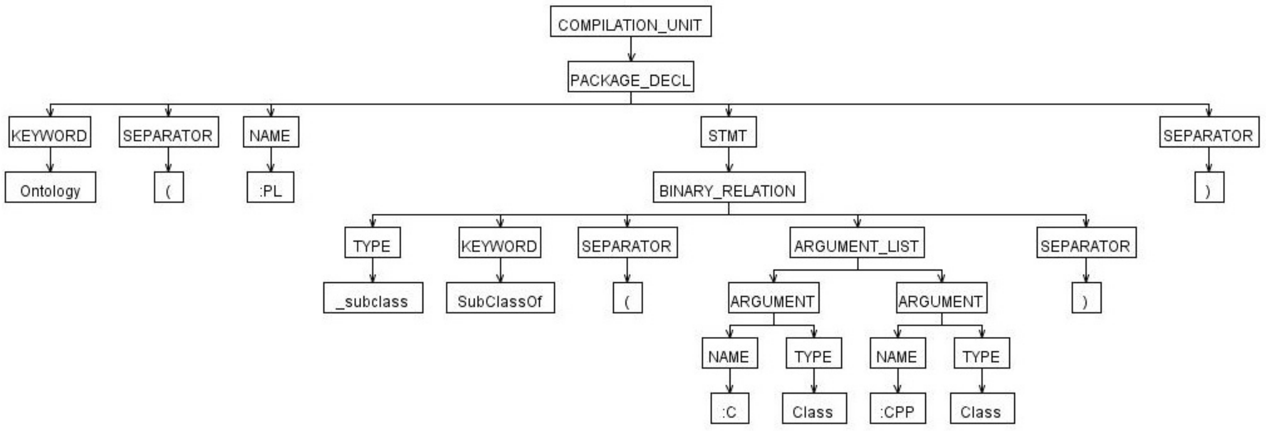

Figure II. eCST representation of a simple ontology.

4. BENEFITS OF OWL2 INTEGRATION INTO SSQSA

Metrics that reflect complexity of a description written in a programming or formal language can be classified as follows:

1. Metrics of internal complexity reflect lexical and syntactical complexity of the description or some of its parts. Lexical complexity measures are derived from the lexical elements of a language and reflect the complexity that is related to the volume of the description. Representative metrics which belong to this category are LOC family of metrics and Halstead [1977] complexity measures. Syntactical complexity is related to the compositional (structural) complexity of concrete language constructs. Cyclomatic complexity is an example of widely used measure of syntactical complexity.
2. Metrics of design complexity reflect the complexity of dependency structures among identifiers introduced in the description. Those metrics quantify inheritance, coupling and cohesion relationships among entities represented by the identifiers. Representative examples are CBO, NOC, DIT and LCOM metrics from the Chidamber-Kemerer metrics suite.
3. Hybrid metrics combine metrics of internal and design complexity. Examples are WMC and RFC from the Chidamber-Kemerer metrics suite, and the Henry-Kafura complexity [Henry and Kafura 1981].

As it can be seen from the review of related works on ontology metrics, the complexity of an ontological description is viewed as some measure of complexity of underlying graph representation. In other words, already introduced ontology metrics belong to the category of design complexity metrics. The integration of OWL2 into the SSQSA front-end provides the eCST representation of ontology. This representation can be used to define (or adopt) and compute metrics of internal complexity, which is not possible in the graph-based representation of ontology. For example, Halstead complexity metrics adopted for ontologies can be calculated in the same way as for software systems: by counting eCST universal nodes representing lexical categories. Similarly, the statement and expression level universal nodes can be used to derive syntactical complexity measures. Currently, it is possible to use the SMIILE back-end to obtain LOC and Halstead metrics for ontological descriptions. SMIILE also calculates cyclomatic complexity (CC) for software systems, but this metric cannot be adopted for ontology evaluation, since there are no OWL2 language elements that correspond to branch and loop statements. However, the predicate counting procedure used for the computation of the CC metric in SMIILE can be adopted to derive the complexity of nested OWL2 class and data range expressions (by counting EXPR universal nodes in eCST).

The ATTRIBUTE_DECL universal node can be used to recognize the declarations of ontologies, declarations of ontological concepts, roles and named individuals in represented ontological description. Relations among those entities can be identified by the analysis of eCST sub-trees rooted at BR, SR and

PKBR universal nodes (see Section 3.2). This means that the graph representation of ontology can be extracted from the eCST representation of ontology. Therefore, an ontology metrics tool that is based on the eCST representation also can be able to compute metrics of design complexity. Finally, metrics of internal and metrics of design complexity can be combined to obtain hybrid complexity metrics. The extraction of the graph representation of ontology is fundamentally different problem that the extraction of software networks due to the structural difference between ontological and software entities. The hierarchy tree representation of an ontological description can be obtained using the SNEIPL back-end, but SNEIPL cannot be used to identify horizontal dependencies (dependencies between entities of the same type) among ontological concepts and individuals. Those ontological entities are structurally atomic, i.e. they are not composed out of other ontological entities. To the contrary, software entities (classes, functions, etc.) are not structurally atomic: the definition of a software entity *A* associates the name of *A* with a body that contains the structure of *A*. Horizontal dependencies between software entity *A* and other entities are contained in the body of *A*, while horizontal dependencies between ontological entities are independent of ontological declarations. Since the SNEIPL back-end cannot be used to obtain the graph representation of ontology, a new SSQSA back-end that computes graph-based ontological metrics will be developed (see Figure I). This back-end will reuse and adopt modules from SMIILE to compute metrics of internal complexity, as well as modules from SNEIPL to form the hierarchy tree representation which is the first step in the extraction of ontological graph (identification of ontological entities and vertical dependencies).

The SSCA back-end constructs and compares hierarchy trees of two consequent versions of a software system in order to determine changes in vertical dependencies (dependencies among entities at different levels of abstraction). The hierarchy tree representation of an ontological description can be obtained from the eCST representation in the same way as for software systems: it is entirely determined by the hierarchical structure of eCST universal nodes in concrete eCSTs. This means that this back-end can be applied for ontologies in order to identify which concepts and named individuals are added or removed in the next version of ontology, and to what extent. Finally, with the design and development of new SSQSA back-ends it will be investigated whether they can be applied to analyze both software and ontological systems.

5. CONCLUSION AND FUTURE WORK

In this paper we described how the SSQSA front-end is extended to support OWL2 language in functional-style syntax. It is also shown that the eCST representation of ontologies can be used to compute metrics that reflect both internal and design complexity of ontological descriptions. Therefore, our future work will include the development of a SSQSA back-end, as shown in Figure I, that uses the eCST representation of ontology to compute metrics reflecting different aspects of complexity of ontological descriptions. In our future work we will also investigate whether recently introduced metrics of cognitive complexity of programs written in object-oriented languages [Misra et al. 2012] and metrics of complexity of web services descriptions [Basci and Misra 2012] can be adopted and used for ontology evaluation.

REFERENCES

Dilek Basci and Sanjay Misra. 2012. Metric suite for maintainability of eXtensible Markup Language web services. *IET Softw.* 5, 3, 320-341.

Zoran Budimac, Gordana Rakić, and Miloš Savić. 2012. SSQSA architecture. In *Proceedings of the Fifth Balkan Conference in Informatics* (BCI '12). ACM Conf. Proc. 1479, 287-290.

Shyam R. Chidamber and Chris F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20, 6, 476-493.

Črt Gerlec, Gordana Rakić, Zoran Budimac, and Marjan Heričko. 2012. A programming language independent framework for metrics-based software evolution and analysis. *Computer Science and Information Systems* 9, 3, 1155-1186.

Thomas R. Gruber. 1993. A translation approach to portable ontology specifications. *Knowl. Acquis.* 5, 2, 199-220.

Maurice H. Halstead. 1977. *Elements of software science.* Elsevier North-Holland, Amsterdam.

Sallie M. Henry and Dennis G. Kafura. 1981. Software structure metrics based on information flow. *IEEE Computer Society Trans. Software Engineering* 7, 5, 510-518.

`

Thomas J. McCabe. A Complexity Measure. 1976. *IEEE Trans. Software Eng.* 2(4): 308-320.

Sanjay Misra, Murat Koyuncu, Marco Crasso, Cristian Mateos and Alejandro Zunino. 2012. A Suite of Cognitive Complexity Metrics. In *Computational Science and Its Application ICCSA* 2012. Lecture Notes in Computer Science, Vol. 7336 Springer Berlin Heidelberg, 234-237.

Boris Motik, Peter F. Patel-Schneider and Bijan Parsia. 2012. OWL2 web ontology language structural specification and functional-style syntax (second edition). Retrieved July, 2013 from http://www.w3.org/TR/owl2-syntax/

Anthony M. Orme, Haining Yao, and Letha H. Etzkorn. 2006. Coupling Metrics for Ontology-Based Systems. *IEEE Softw.* 23, 2, 102-108.

Terence J. Parr and Russell W. Quong. 1995. ANTLR: a predicated-LL(k) parser generator. *Softw. Pract. Exper.* 25, 7, 789-810.

Ivan Pribela, Gordana Rakić, and Zoran Budimac. 2012. First Experiences in Using Software Metrics in Automated Assesssment. In *Proc. of the 15th International Multiconference on Information Society* (IS), *Collaboration, Software and Services in Information Society* (CSS), Vol. A, 250-253.

Gordana Rakić and Zoran Budimac. SMIILE Prototype. 2011a. In *Proc. Of International Conference of Numerical Analysis and Applied Mathematics* ICNAAM2011*, Symposium on Computer Languages, Implementations and Tools* (SCLIT), AIP Conf. Proc. 1389, 853-856.

Gordana Rakić and Zoran Budimac. Introducing Enriched Concrete Syntax Trees. 2011b. In *Proc. of the 14th International Multiconference on Information Society (IS), Collaboration, Software and Services in Information Society (CSS)*, Vol. A, 231-234.

Miloš Savić, Gordana Rakić, Zoran Budimac, and Mirjana Ivanović. 2012. Extractor of software networks from enriched concrete syntax trees. In *Proc. Of International Conference of Numerical Analysis and Applied Mathematics* ICNAAM2012, *Symposium on Computer Languages, Implementations and Tools* (SCLIT), AIP Conf. Proc. 1479, 486-489.

Samir Tartir, Budak I. Arpinar, Michael Moore, Amith P. Sheth, and Boanerges Aleman-Meza. 2005. OntoQA: Metric-based ontology quality analysis. In *Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources.*

Hongyu Zhang, Yuan-Fang Li, and Hee Beng Kuan Tan. 2010. Measuring design complexity of semantic web ontologies. *J. Syst. Softw.* 83, 5, 803-814.

Rok Žontar and Marjan Heričko. 2012. Adoption of object-oriented software metrics for ontology evaluation. In *Proceedings of the Fifth Balkan Conference in Informatics* (BCI '12). ACM Conf. Proc. 1479, 298-301.