UML Class Diagram Composition Using Software Requirements Specifications

Alexey Tazin
Department of Electrical and
Computer Engineering
Northeastern University
Boston, MA 02115
Email: tazin.a@husky.neu.edu

***Abstract*—Consider a scenario of collaborative software engineering process in which different software engineering teams create different versions of software design that satisfy given software requirements specifications. Often the software designs are expressed as UML diagrams, e.g., class diagrams. In this process, the class diagrams developed by different teams have to be reconciled and only one final diagram should be included in the final design. The reconciliation process may include the selection of the "best" parts of each proposed diagrams and used to compose the optimal diagram from the selected parts. It is difficult for software engineers to achieve such a goal manually since manual process is very time consuming, error prone, and not scalable for large software projects. We are developing an approach and algorithms for automating the selection of the best subdiagrams developed by different teams and composing them into one diagram that is optimal with respect to a given objective function. The final diagram must satisfy software requirements specifications that were satisfied by each proposed diagram. The developed approach will be evaluated experimentally by generating random software requirements (expressed in SPARQL and OWL), generating random class diagrams satisfying these requirements, generating composed diagrams that satisfy these requirements, and computing two class diagram metrics for each composed diagram. These metrics are number of classes and number of associations. This approach was implemented using Jena, OWL ontologies, ArgoUML, and SPARQL.**

I. Introduction

During large scale software development it is almost impossible to come up with an optimal software design. A software engineering team may develop a class diagram using some software requirements specifications and then over time perform refactoring of the diagram. The refactored diagram may or may not be optimal. In this case, the original and the refactored diagrams should be reconciled. The reconciliation process may include the selection of the best parts of the original and refactored diagrams and composition of the optimal diagram from the selected parts. Also, different teams may create different versions of class diagram that satisfy given software requirements and these versions should be reconciled in terms of optimization. In another scenario, different teams may create diagrams modeling different parts of a system where each diagram has parts that have the same functionality and satisfy the same software requirements specifications. It is desirable to eliminate such redundant diagram parts. This elimination process may include the selection of the best parts among the original diagrams and composition of the optimal diagrams from the selected parts. It is difficult for software engineers to achieve such reconciliation of diagrams or elimination of redundant diagram parts manually since manual process is very time consuming, error prone, and not scalable for large software projects. In this paper we show the basic steps of the process of automating the selection of the best subdiagrams developed by different teams and composing them into one diagram that satisfies a set of stakeholder needs and is optimal with respect to a given objective function. One of the issues that we need to address in this research is the issue of evaluation of the proposed approach. In particular, we need to insure that the proposed approach is applicable to a wide variety of types of software requirements related to class diagrams. Towards this aim, we have developed a classification of functional software requirements using minimal UML metamodel. The current minimal metamodel includes Class, Association, Property, DataType and Generalization classes and will be expanded to support commonly used class diagram elements. Each class of this classification is represented using SPARQL ASK query [1] template with variables for class names, primitive data types and cardinalities.

II. Related Work

There are some works [2][3] that describe implementation of 3-way merge for UML class diagrams. The main disadvantage of this type of merge is that it requires an original diagram to be used as base of merging two diagrams derived form it. Chechik et al. [4] describe two merge operators for models: algebraic merge and state machine merge. For both operators relationships between model elements are created manually, although the merge is done automatically using a tool. The approach used by Nejati et al. [5] and Odyssey-VCS [6] compares UML diagrams based on the syntax only. The method used by Al-Khiaty and Ahmed [7] compares classes using their names, attribute names, operation names, relationship names, and neighbors. All the names are compared based on their semantic similarity using WordNet [8]. Comparison accuracy depends of weight assignment of each similatity metric. The method used by Nejati et al. [5] compares states in hierarchical statechart models using behavioural matching technique. It does not take into consideration the purpose

and responsibility of model elements in more general sense. Chechik et al. [4] present some common types of overlaps between concepts in different models with informal semantics, semi-formal semantics, and formal semantic properties. This method does not provide any systematic way to discover overlaps between elements for class diagram with informal semantics or semi-formal semantics. The overlaps are determined manually. The method used by Costa et al. [9] uses the 3-way merge. The method uses rules describing the semantics of the class diagram according to UML metamodel. These rules are used to infer indirect relationships between class diagram elements. This method supports comparison of only limited number of class diagram types. Some methods [10][11] use graph isomorphism [12]. The method used by Rao and Gupta [13] compares diagrams as graphs based on their heuristics. These methods do not recognize the semantics of diagram. The approach used by Fahrenberg et al. [14] computes the conjunction of class diagrams by interpreting them as views of the same model. This approach does not support matching of two diagrams with different classes but satisfying the same set of requirements. Semantic approaches used by Maoz et al. [15] enumerate some examples of instances which represent the difference between two diagrams. These approaches are incomplete due to small number of examples that can be created.

III. Solution

In this section we describe the basic steps of the proposed process of developing an optimal UML class diagram based on multiple diagrams developed by different teams. These steps are based on the following assumptions. Two software engineering teams are given the same stakeholder needs (written in text) and develop two UML class diagrams based on these needs. The diagrams are developed using ArgoUML tool. We will support more UML tools in the future. There is one diagram per package. The stakeholder needs are (manually) converted to SPARQL queries. Different names referring to the same thing are resolved using WordNet, and axioms. The process begins with inputting such two diagrams and a SPARQL representation of the stakeholder needs into the proposed tool.

1) Read two UML diagrams developed using ArgoUML.
2) Read stakeholder needs expressed as SPARQL queries.
3) In each of the diagrams, identify all possible subdiagrams.
4) For each pair of subdiagrams compute their union.
5) For each union, compute the quality metric.
6) Map each union to Web Ontology Language (OWL) [16] using Ontology Definition Metamodel (ODM) based tool UMLtoOWL that converts ArgoUML class diagrams to OWL.
7) Enrich each union ontology with inferred object and data properties that correspond to derived relationships and attributes in UML using SPARQL Update axioms.
8) Identify unions that satisfy the given set of stakeholder needs.
9) Select the union that has the highest value of the quality metric.

This process will be iterative, i.e., the results returned by the tool will be shown to the teams who will make the decision on the final diagram.

A. What is subdiagram?

For the purpose of finding subdiagrams, we use the bidirected connected multigraphs [17] $G = (V, E)$, where V is a set of vertices of the diagram, and E a set of edges of the diagram. Each edge is a set of two vertices in case of undirected, or an ordered pair of vertices in case of directed.

A subdiagram G^* of a given diagram G is defined as a bidirected connected or disconnected multigraph $G^* = (V^*, E^*)$ where

- $V^*(G^*) \subseteq V(G)$ is the finite set of subdiagram vertices,
- $E^*(G^*) \subseteq E(G)$ is the finite set of subdiagram edges.

B. How to Find Subdiagrams and Generate Union?

We execute an exhaustive search to find all possible subsets of a set A that includes a class, an interface, or a group of classes or interfaces related with generalization relationship. In the future we are planning to consider groups of classes and interfaces related with associations. The time complexity of this algorithm is $O(2^{ceil(n/m)})$, where n is number of classes and interfaces in the diagram and m is average number of classes and interfaces in each A_i. The time complexity of the union will be $O(2^{2ceil(n/m)})$. If each A_i on average has two classes and $m = 2$ accordingly, the time complexity of the union algorithm will be $O(2^n)$. We are going to generate unions that are connected graphs and do not have classes with the same name but different structure. This would reduce time complexity of the union algorithm.

C. Representing UML Class Diagrams in OWL

The UML class diagram to OWL conversion is based on mapping between UML elements and OWL entities offered by Ontology Definition Metamodel (ODM) specfication [18]. To achieve the mapping we use an existing tool - UML2OWL by Leinhos [1]. The tool is described in [19]. The tool supports class diagrams in XMI 1.2 format implemented by Poseidon 4.1. We modified the tool to enable reading class diagrams in XMI 1.2 format implemented by open source ArgoUML.

D. Stakeholder Need Query

We express stakeholder need statements related to class diagrams as SPARQL ASK queries. These queries are executed against ontological representations of class diagrams. A stakeholder need statement is satisfied by a class diagram if the corresponding SPARQL ASK query can be executed against the class diagram ontology and return true. The example below shows a need statement expressed using SPARQL query. This query refers to Cook and Appetizer classes, some object property, and some restriction on this property and class

[1] http://diplom.ooyoo.de/

Appetizer, where the Cook class is subclass of this restriction. The restriction includes minimum cardinality of 0. This need statement could be articulated in English as:

"Cook can have appetizer recipes."

This query can be mapped to SPARQL as shown below:

```
ASK {:Cook rdf:type owl:Class.
     :Appetizer rdf:type owl:Class.
     :Cook rdfs:subClassOf ?r1.
     ?r1 rdf:type Restriction.
     ?r1 owl:onProperty ?p1.
     ?r1 owl:onClass Appetizer.
     ?r1 owl:minQualifiedCardinality
     '0'^^xsd:nonNegativeInteger }
```

E. Union Ontology Inferences

Stakeholder need statements related to class diagrams describe to the elements of the class diagrams. A stakeholder need statement is satisfied by a diagram if the diagram includes all the elements described by this stakeholder need statement. Some of these elements are not present in the diagram initially developed by a software engineer but are inferable based on other elements not directly described by the stakeholder need statement. These inferable elements are derived properties and derived associations in UML.

A stakeholder need statement expressed as a SPARQL ASK query includes elements of the class diagram ontology in its ASK clause. Such a query can be answered using the class diagram ontology and return true if all elements included in the query ASK clause are present in the class diagram ontology. Some of these elements corresponding to derived properties and derived associations in UML must be inferred and asserted using axioms.

In our experiments we used an inference engine to infer and assert object properties, data properties, and property restrictions; the inferences are based on general OWL axioms in the class diagram ontology and on SPARQL Update queries [20] that assert new axioms into the ontology. The elements inferred and asserted by the query are specified in its INSERT clause. The following example shows an example of axiom inserting query. This axiom infers and asserts an object property and restriction corresponding to derived association in UML. The axiom is represented in ontological terms that correspond to the UML terms. The SPARQL Update query uses Jena reasoner [21]; the inference takes advantage of the general OWL *subClassOf* transitivity axiom.

1) Axiom description: If there are two classes A and B and a directed association from class A to class B with cardinality $0..*$ at its navigable end, we can infer a directed association with cardinality $0..*$ at its navigable end from class A to any subclass of class B.

2) SPARQL Update query: The following is query assert a new object property and restriction for the match pattern specified in the WHERE clause.

```
INSERT
{ ?p2 rdf:type owl:ObjectProperty.
  ?p2 rdfs:domain ?c1.
  ?p2 rdfs:range ?c3.
  _:r2 rdf:type owl:Restriction.
  _:r2 owl:onProperty ?p2.
  _:r2 owl:onClass ?c3.
  _:r2 owl:minQualifiedCardinality
  '0'^^xsd:nonNegativeInteger.
  ?c1 rdfs:subClassOf _:r2
}
WHERE
{ ?c1 rdfs:subClassOf ?r1.
  FILTER (NOT EXISTS
  {?c1 rdf:type owl:Restriction}).
  ?r1 rdf:type owl:Restriction.
  ?r1 owl:onClass ?c2.
  ?r1 owl:minQualifiedCardinality
  '0'^^xsd:nonNegativeInteger.
  ?r1 owl:onProperty ?p1.
  ?c3 rdfs:subClassOf ?c2.
  FILTER (?c3 != ?c2 ).
  ?p1 rdfs:domain ?c1.
  BIND(IRI(CONCAT(STR(?c1),
  STRAFTER(STR(?c3), '#'))) AS ?p2)
}
```

Each newly created restriction will have a different anonymous name. This is handled internally by Jena [22]. The name of a newly created object property is obtained via concatenating names of the classes corresponding to the variables $c1$ and $c3$.

3) Query match RDF graph: OWL semantics can be encoded in RDF [23]. RDF models are essentially collections triples (predicate, subject, object) that constitute graphs. The RDF graph shown in Fig. 1 includes a subgraph corresponding to the match pattern of the above SPARQL Update query WHERE clause and the RDF triples asserted during the query execution process. Resources $c1, c2, c3, p1, p2, r1, r2$ of the graph correspond to the variables in the query. The query can find multiple occurrences of subgraphs corresponding to the match pattern and assert multiple groups of triples specified in the INSERT clause.

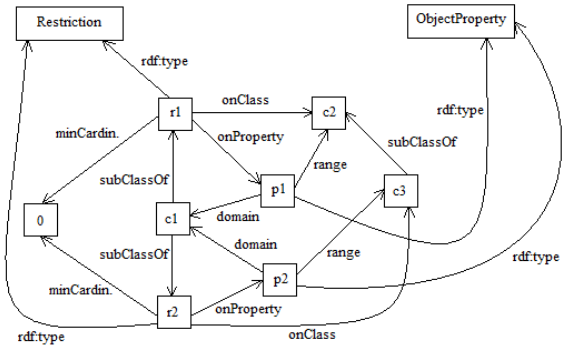

Fig. 1: Axiom SPARQL Update query RDF graph.

4) Class diagram: Fig. 2 shows a class diagram that does not include any derived properties and associations. We convert this class diagram to OWL ontology, execute the above axiom query against this ontology, assert a new object property and restriction. This new property and restriction correspond to the *hasAppetizer* derived directed association from class Cook to class Appetizer. This is shown in Fig. 3. Also, Fig.

3 shows the correspondence between classes and associations, and axiom query variables.

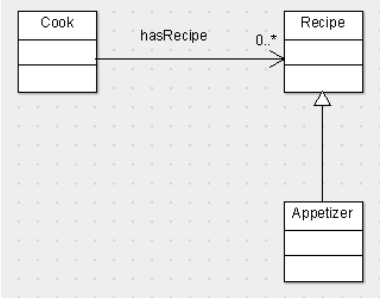

Fig. 2: Class diagram that does not include derived associations and properties.

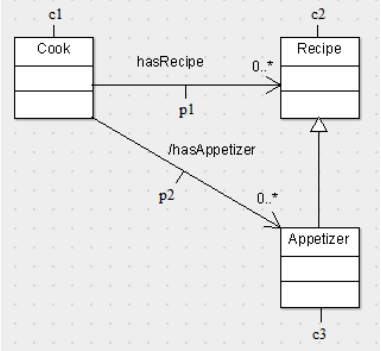

Fig. 3: Class diagram that includes derived association.

IV. Plan for Evaluation and Validation

The developed approach will be verified by generating random software requirements, generating random UML class diagrams satisfying these requirements, and generating an optimal composed diagram that satisfies these requirements using number of classes and number of associations class diagram metrics. The optimal composed diagram can be generated using these two metrics based the goal of the ultimate application to be designed. If the goal is maintenance, then we might want to maximize the number of classes metric. The number of associations should be minimized in any case in order to reduce complexity of the diagram. In the future, we are also planning to consider more metrics. Each software requirement is generated as SPARQL query that asserts OWL facts. For each query template we will generate a set of more specific templates with specified primitive data type and cardinality variables. For cardinality constraints we will use 0, 1, a positive integer, and many. Each such template will be used to generate a query with arbitrary class names and a set of diagrams satisfying this query. We are going to generate each random query by randomly selecting a template with specified primitive data type and cardinality variables out of all the available such templates and randomly specifying class names. For each randomly generated query we will generate a random diagram satisfying this query using previously generated diagrams satisfying queries with arbitrary class names. These randomly generated diagrams will be merged into one diagram and become subdiagrams of this diagram. Then, we need to randomly connect these subdiagrams with directed associations. This approach will be used for generating a random set of queries and a pair of class diagrams that will satisfy these queries.

V. Expected Contributions

1) Most of diagrams are developed based on given stakeholder needs. In case of diagram merging it is important to verify if the merged diagram satisfies the same stakeholder needs. Our method provides means for such verification.
2) Our method allows to generate semantically equivalent merged diagrams, rather than just one merged diagram, thus giving an opportunity for a software developer to select a solution that better satisfies the designers objectives expressed in terms of design quality metrics. The designer can also select the best subdiagrams from any two given diagrams and compose them into one diagram that is optimal with respect to a given objective function.
3) Classification of software requirements specifications using UML class diagram metamodel.
4) Representation of software requirements using a formal language SPARQL.
5) Automated method of satisfaction of software requirements by class diagram.
6) Generating random class diagrams for a given set of software requirements.
7) Inference of indirect associations in the class diagram.

VI. Current Status

We developed a software implementation of our approach with support of limited number of axioms and full support of all elements of a minimal UML metamodel. We developed a classification of functional software requirements using minimal UML metamodel. We outlined a random class diagram generation algorithm. We are planning to finish experimental validation of our approach with more design quality metrics, expand minimal UML metamodel to all most frequently used class diagram elements, and write all the axioms in April 2018.

References

[1] W3C, "SPARQL 1.1 Query Language." [Online]. Available: https://www.w3.org/TR/sparql11-query/
[2] M. Alanen and I. Porres, "Difference and union of models," *International Conference on the Unified Modeling Language*, 2003.
[3] Eclipse Project, "Emfcompare." [Online]. Available: https://www.eclipse.org/emf/compare
[4] M. Chechik, S. Nejati, and M. Sabetzadeh, "A relationship-based approach to model integration," *Innovations in Systems and Software Engineering*, 2012.
[5] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," *ICSE 2007*, 2007.

[6] L. Murta, C. Correa, J. G. Prudencio, and C. Werner, "Towards odyssey-vcs 2: Improvements over a uml-based version control system," *In proceedings of the 2008 international workshop on Comparison and versioning of software models (CVSM 08), Leipzig, Germany*, 2008.

[7] Mojeeb Al-Rhman Al-Khiaty and M. Ahmed, "Uml class diagrams: Similarity aspects and matching," *Lecture Notes on Software Engineering, Vol. 4, No. 1*, 2016.

[8] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet::similarity - measuring the relatedness of concepts," *HLT-NAACL*, 2004.

[9] V. Costa, R. Monteiro, and L. Murta, "Detecting semantic equivalence in uml class diagrams," *SEKE*, 2014.

[10] R. S. Rao and M. Gupta, "Design pattern detection by greedy algorithm using inexact graph matching," *IJERT*, 2013.

[11] ——, "Design pattern detection by sub graph isomorphism technique," *International Journal Of Engineering And Computer Science*, 2013.

[12] G. Chartrand, *Introduction to graph theory*. New York: Dover Books, 1984.

[13] R. S. Rao and M. Gupta, "Design pattern detection by a heuristic graph comparison algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, 2013.

[14] U. Fahrenberg, M. Acher, A. Legay, and A. Wasowski, "Sound merging and differencing for class diagrams," *FASE 2014: Fundamental Approaches to Software Engineering, pp 63-78*, 2014.

[15] S. Maoz, J. O. Ringert, and B. Rumpe, "A manifesto for semantic model differencing," *In MODELS, pp. 194203. Springer*, 2011.

[16] W3C, "Web ontology language reference." [Online]. Available: http://www.w3.org/2004/OWL/

[17] F. Harary, *Graph Theory*. Addison-Wesley, 1994.

[18] OMG, "Ontology definition metamodel." [Online]. Available: http://www.omg.org/spec/ODM/

[19] S. Leinhos, "Owl ontology extraction and modelling from and with uml class diagrams - a practical approach," Master's thesis, University of the Federal Armed Forces Munich, 2006.

[20] W3C, "SPARQL 1.1 Update." [Online]. Available: https://www.w3.org/TR/sparql11-update/

[21] The Apache Software Foundation, "Reasoners and rule engines: Jena inference support." [Online]. Available: https://jena.apache.org/documentation/inference/

[22] ——, "Apache jena." [Online]. Available: https://jena.apache.org/

[23] W3C, "Resource description framework (rdf)." [Online]. Available: www.w3.org/RDF/