Case Study on Analogical Reasoning to Provide Conceptual Support for Software Reuse

Rushikesh Amin, Mel Ó Cinnéide and Tony Veale
Department of Computer Science, University College Dublin,
Belfield, Dublin 4, Ireland.
{rushikesh.amin, mel.ocinneide, tony.veale}@ucd.ie

Abstract. Analogical Reasoning is a widely-used technique for the purpose of transferring knowledge from one domain or system to another. Existing research has already proven analogical reasoning to be valuable in the context of software reuse. In LASER (Lexical Analogy for Software-Engineering Reuse), we use analogical reasoning to find reusable software components by associating a class name with its conceptual representation in an English-language lexical-ontology such as WordNet. This enables us to provide support for software reuse without requesting any special information from the user, and without requiring that the reusable components have been tagged with any special annotations. Components are proposed for reuse by identifying the distance between them in a semantic net built on WordNet. In this paper we present the results of experiments to determine the applicability of this technique to recently-developed software.

1 Introduction

Software reuse involves the creation a software system using existing software components rather than the more expensive and time-consuming creation of a system *ab initio*. Reuse, in its broadest sense then, can be viewed as the reapplication of knowledge from one system to another [2]. In particular, reuse of software components can enhance programmer productivity and increase the quality of a software system while reducing its cost and development time. However, there are some downsides to the reuse philosophy. Firstly, developers are not always eager to use reusable components. Secondly, in order to promote reuse, it is necessary to locate, understand, evaluate, and adapt existing software components. The provision of intelligent support for these reuse tasks is not a trivial matter.

One approach to the provision of reuse support is the creation of a software component library. A major difficulty in designing software libraries is in the selection of a component representation that will facilitate the classification and retrieval processes. Another possibility is the use of UML-level reuse techniques [7, 8], but in this case developers have to annotate and index their software designs for future reuse. When viewed in the context of the current popularity of Agile Processes [11], where little up-front design is performed and even code commenting is kept to a minimum, the weaknesses of this approach become apparent.

Artificial Intelligence has contributed to overcoming these problems in past research and it has even been stated that "software reuse is the common ground where AI and software engineering will meet" [13]. In recent years various AI techniques including analogy, collaborative filtering and fuzzy logic have proven useful in the area of software reuse [3]. In our approach, called LASER (Lexical Analogy for Software-Engineering Reuse), we use code-level analogical reasoning to suggest the conceptually most relevant reuse component to the developer by using WordNet, a knowledge base of the English lexicon [5].

Analogical reasoning is a widely-used problem solving technique to transfer knowledge from one system to another [9].Carbonell introduce the concept of derivational analogy to integrate complete derivational traces to previously solved problems. Analogy involves a structural comparison of two concepts that appear substantially different on the surface but which exhibit important causal or semantic symmetries. Indeed, computational models of analogy have already shown themselves to be valuable in the development of UML level reuse techniques, e.g., the *ReBuilder* project [7, 8]. The linguistic and conceptual ability of analogy allows words and their meanings to play an important role in the effectiveness and comprehension of analogies. In modern OOP languages like Java, which encourages the use of meaningful naming for entities with conceptual correlates, like classes (categories), variables (attributes) and methods (behaviours), names and their lexico-semantic connotations can be used to obtain a greater degree of understanding of a program in terms of the concepts it is designed to model. For example, if class and variable names can be parsed to reveal conceptual categories from WordNet, LASER can use common-sense knowledge about how these categories combine to derive a conceptual representation of the software being developed. Furthermore, this conceptual level of understanding can supplement the analogical reasoning needed to retrieve 'conceptually similar' components from the source code repository.

In this paper we present our ongoing work on the application of analogy to the software reuse domain. We have completed a preliminary study by analysing the source code of *JRefactory* [6], an open source refactoring tool, to determine if class and method names follow their natural language meanings and to determine if the extension relationship between a class and its associated superclass reflects the lexico-conceptual relationship between the name of the class and the name of its parent. As a basis for our work we are using WordNet, a lexical database that provides, in addition to basic thesaurus capabilities, a form of conceptual structure that can be exploited for reasoning.

In the next section we introduce related work while in section 3 we describe the system components, including analogy modules, of LASER. In section 4 we then describe how we use WordNet as a knowledge base. Various case studies are presented in section 5 and finally, in section 6, conclusions are presented and some directions for future research are outlined.

2 Related Work

Several research efforts have aimed to provide support for software reuse through the use of analogical reasoning. Most of them provide some form of support for the software developer in retrieving and tailoring selected reusable components. From these we have selected those that are similar to our work and discuss them further here.

ReBuilder [7, 8], a software tool being developed in the AI Lab of the University of Coimbra, uses WordNet to index and retrieve software cases. ReBuilder allows analogical retrieval and mapping between UML descriptions of software systems, and uses analogical transfer to flesh out a new software design based on structural parallels with a pre-existing design. However, the benefits of this reuse scheme can only be reaped by those developers who take the time to tell ReBuilder how to appropriately annotate and index their software for future retrieval.

Ira, a prototype tool, uses analogical reasoning to reuse software specification [14]. Ira supports the software designer by obtaining customization guidance from the software designer to retrieve candidate specification from software repositories to tailor the retrieved candidate specifications to the target domain.

An interactive environment to help the software designer reuse requirement specifications is described by Spanoudakis and Constantopoulos [10]. The interactive environment deals with the object oriented specification and retrieves the reuse candidates by identifying their level of similarity.

Reuse of OOram (Object-oriented role analysis modelling) using analogical reasoning is described in [12]. They explore WordNet for finding similarities among OOram components. Retrieval phase in this case supports both structural and semantic similarities. The ranking of base models after retrieval decides what models are passed to the mapping phase.

3 Analogical Reasoning in LASER

Analogical reasoning in LASER is applied at the code level rather than at the UML level in order to develop a conceptual understanding of software being developed. The system architecture of LASER comprises the following modules as shown in figure 1.

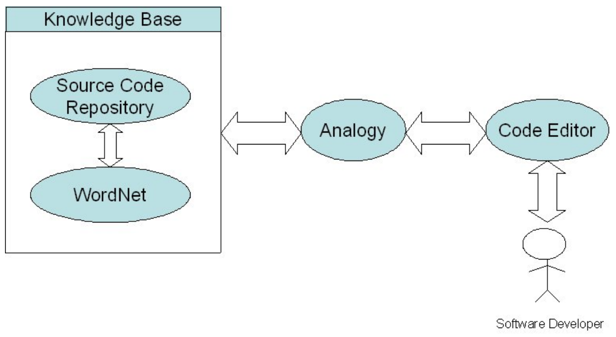

Fig. 1. System architecture of LASER

3.1 Knowledge Base

The Knowledge base in LASER comprises two main modules:

1. Source code from the open source code repository
2. WordNet

Source Code Repository: The object-oriented paradigm facilitates reuse of code by packaging the most reusable structure and behaviors into distinct classes. The programmer can extend the basic functionality of these classes and/or modify it to obtain the desired functionality. Our research currently focuses on software written in the Java programming language. In LASER, the source code repository contains a valuable wealth of Java classes. As new classes are added to the system being developed, LASER automatically updates the code repository.

WordNet: LASER uses WordNet [5],a broad coverage knowledge base of English Lexicon, to retrieve semantically similar components from the code repository. WordNet supports conceptual search rather than merely alphabetical, and is an interesting system in this respect. WordNet is built around the notion of a *synset*, a set of synonyms (or near-synonyms) that can each denote the same underlying concept. For our initial experiments with LASER we exploit the large number of noun-synsets provided by WordNet (over 70,000), as well as two semantic relations, *is-a* and *part-of*, that WordNet uses to connect these synsets into a taxonomy and a partonymy respectively. For example, `Student` *is-a* `Person` and `Classroom` is *part-of* `School`. The *is-a* hierarchy of the noun person is shown in figure 2. LASER suggests a component for reuse by first considering the lexico-semantic meaning of the target class T under construction. identifying the distance, and by then exploring similar source concepts S in WordNet whose synsets name a corresponding classes in the software repository. WordNet also allows LASER to predict potential parent classes using the same lexico-conceptual knowledge. For example, WordNet can be used to look up all synsets containing the word `Person` to find synonyms and hypernyms like `Individual` and `Entity`. This will allow LASER to propose reuse-connections between a new class called `Person` and classes called `Individual` or `Entity` if these already exist in the repository.

3.2 Analogy module

Analogy has been considered to be the process of transferring knowledge from a base case to the target case. In LASER, we apply lexically-driven analogy at code-level rather than at abstract component-level to retrieve semantically similar components from the source code repository. The computational model for analogical problem solving can be described in 5 parts [9], as depicted in figure 3. We now expand further on Retrieval, Expansion and Mapping in the context of LASER. Justification and Learning are omitted as these are issues we have not yet addressed.

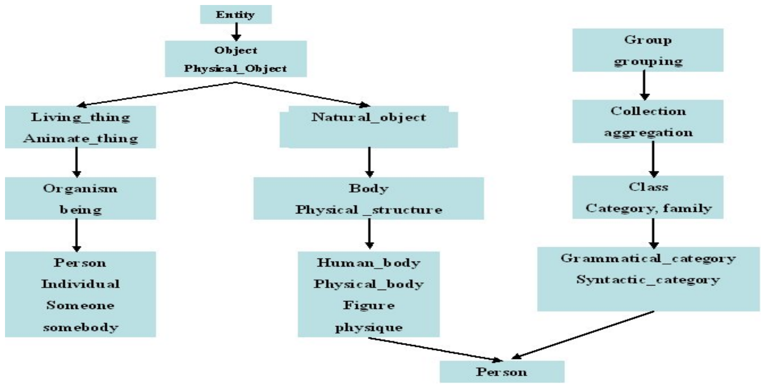

Fig. 2. *is-a* hierarchy for different senses of the noun person"

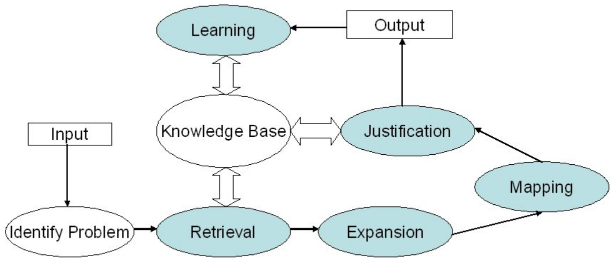

Fig. 3. Analogy Modules in LASER

Retrieval: This involves the retrieval of most relevant cases from the knowledge base. For example consider the following context:

```
public class School extends _
```

By using analogical reasoning we can implement different strategies to suggest new superclass for the class `School`. Using WordNet as a knowledge base, we can suggest a superclass that has more general meaning than `School`, for example `Institution`, `Organization` etc.

Expansion: Expansion involves the suggestion of new ideas to the software developer. Consider a context in which a developer begins to create an application for a school, starting with the creation of a class called `Student`. This may allow LASER to retrieve an existing software design for a hospital, based on the similarity of the `Student` and `Patient` classes. Structure-mapping between the nascent elements of the school system and the existing case for a hospital might then suggest that the school system needs classes for `Desk`, `Room`, `Class`, `Teacher`, `Principal` and `TeachingAssistant`. LASER may create stubs for these classes automatically and add them to the current project specification.

Mapping: Mapping refers to the transfer of knowledge from the base case to the target case. In LASER, we intend to conceptually ground the transfer process by making it lexically-driven. We use a similarity metric for mapping the target object synset and base object synset (see section 4 for more detail). The analogy module then suggests the best reuse component to the developer amongst all possible base components.

4 A Similarity Metric for WordNet

LASER therefore selects a component to be reused by identifying the conceptual distance between the target case under construction by the user and a source case from the repository. Given a new class named T by the user, this analogical target can be matched with an existing source class S from the repository if the WordNet concept (or synset) underlying both S and T are sufficiently similar according to some metric. In designing this metric we consider the three factors discussed by Gomes [7].

Consider a target class (the class under development) whose name can be mapped to a synset T, and a source class whose name is associated with synset S. Then MSCA(S, T) is the Most Specific Common Abstraction synset between S and T. For example MSCA(`cat`, `dog`) will be `carnivore`.

Taxonomical distance similarity: This measures the similarity between synset S and synset T in the WordNet ontology. The higher the value, the more similar are the synsets, and hence the closer they are.

$$T_s = 1 - \frac{D(S, MSCA(S,T)) + D(T, MSCA(S,T))}{2 * MaximumDepth} \tag{1}$$

Where D(S, MSCA(S, T)) is the distance between S and MSCA(S, T), and MaximumDepth corresponds to the longest is-a path in the WordNet taxonomy.

Equilibrium measure: This relates to the equilibrium degree of the tree composed of all synsets between S and MSCA(S, T), and all synsets between T and MSCA(S, T). The closer the value is to 1.0, then the more balanced are the distances between S-MSCA(S, T), and T-MSCA(S, T).

$$E_m = 1 - \frac{|D(S, MSCA(S,T)) - D(T, MSCA(S,T))|}{\sqrt{(D(S, MSCA(S,T)))^2 + (D(T, MSCA(S,T)))^2}} \tag{2}$$

Absolute depth measure: This is related to the taxonomical depth of MSCA(S, T). The closer this value is to 1.0, then the deeper the MSCA(S, T) is located.

$$A_d = \frac{Depth(MSCA(S,T))}{MaximumDepth} \tag{3}$$

Here Depth(MSCA(S, T)) is the minimal distance from MSCA(S, T) to a taxonomical root synset.

The analogical algorithm uses these three measures in a weighted sum to select the best mapping object from a list of candidates.

Similarity metric: We use following similarity metric between synset S and synset T.

$$if\_MSCA(S,T)\_not\_exists \Rightarrow -1. \tag{4}$$

$$if\_MSCA(S,T)\_exists \Rightarrow w1 * T_s + w2 * E_m + w3 * A_d. \tag{5}$$

Where w1, w2 and w3 are the weights associated with each factor. The values selected for this case study are: 0.55, 0.3 and 0.15 respectively (see [8]). The closer this value is to 1.0 the more similar the synsets are perceived to be.

The similarity metric we have just described will be used in section 5.3 to attempt to predict the superclass of class that is currently being developed.

5 Case Study

In this section we describe a number of different case studies intended to assess the performance of LASER. We analysed *JRefactory*, an open source refactory tool written in Java that contains 1259 classes and 6966 methods. The first case study simply tested if class and method names used in *JRefactory* were to be found in WordNet. The remaining two case studies focussed on the inheritance relationship between classes and whether correct superclass prediction is possible.

5.1 Linguistic ability of class names and method names

Our fundamental hypothesis is that most programmers follow natural language conventions in naming classes, methods and variables, even if most names are in fact compound terms. A programmer name can be mapped to a synset in WordNet if the name actually occurs in a synset (e.g., consider a class named Client or Server), of if the linguistic head of a compound name appears in a synset (e.g., consider a class named WebClient, where the capitalization convention in Java allows LASER to isolate the head Client in the compound).

One way to test this hypothesis is to determine the percentage of programmer names that can be mapped to WordNet in this way [1]. Once this is done, we also need to determine whether the names so mapped actually reflect their conceptual interpretations. To do this, we consider how many extension class and parent class pairings have names that can be mapped to WordNet synsets that in turn partake in a taxonomic *is-a* relationship.

As we discussed in [1], LASER decomposes a given class name using the capitalization standard conventionally used in Java programs. Acknowledging that there are frequent lexemes used by programmers that do not occur in WordNet (like API, int, etc.) we include the most common of these as an extra form of lexical knowledge. An example of how LASER interprets a class name is as follows:

Class Name	**Match after decomposition**			**Average match**
MyBeerCase	My=100%	Beer=100%	Case=100%	100%
MyBeerPZK	My=100%	Beer=100%	PZK=0%	66.66%

Table 1: Average accuracy match for each class with WordNet

The final result is calculated as follows.

$$R_{avg} = \sum_{i=1}^{N} (P_{avg}/N) \tag{6}$$

Where R_{avg} is the average accuracy of percentage match with WordNet.
P_{avg} is Accuracy of match per class and N is Total number of classes, 1259 in this case. Results of experiment are shown in table2.

Accuracy of match with WordNet			
	100% Match	0% Match	Average Match
Class Names	66%	0.6%	85%
Method Names	61%	3%	75%

Table 2: Class and method names match with WordNet

Although this is as yet only a pilot study, the results are very promising as regards the goals the LASER project. The vast majority of class names and their lexical components, and a strong majority of method name components, are amenable to conceptual annotation using linguistic techniques.

5.2 *IS-A* relationship between class and superclass name

The second part of our case study concerns not whether class names can be mapped to WordNet synsets, but whether class names that can be so mapped actually conform to the conceptual meaning given to them by the mapping. For example, if the name of a class and the name of its parent class can both be mapped to WordNet synsets, we should expect that the extension relationship in

Java should be mirrored by a taxonomic relationship between the corresponding synsets.

1259 pairs of class and superclass names can be found in the JRefactory repository. Perhaps surprisingly, but pleasantly so, over 84% of these pairings can be mapped to WordNet to find a corresponding *is-a* relationship. This strongly vindicates the somewhat speculative hypothesis that LASER is founded upon, namely that a linguistic analysis of program elements can yield a conceptual insight into the modelling intentions of the programmer and the actions of the software itself.

5.3 Prediction of superclass using similarity metric

We build on this result to demonstrate that the linguistic interpretation of a class name can be used to suggest an appropriate parent class from which to inherit and reuse behaviour. Suppose a developer is about to implement class named `Customer`. Based on the conceptual meaning of this class name `Customer` in WordNet, LASER can suggest the most suitable existing classes to extend, such as `Borrower`, `Agent`, `Client`, and so on.

If we consider `Customer` as synset S and `Borrower` as synset T as described in section 4, then the MSCA(S, T) is `Person`. D(S, MSCA(S, T)) value of 3 is the distance measured in *is-a* links between `Customer` and `Person`, while D(T, MSCA(S, T)) also yields a value of 3 with MaximumDepth = 17 and Depth(MSCA(S, T)) = 4. Likewise, for `Customer` and `Agent` these values are 7, 2, 9 and 17 respectively. The following table shows the value of all three factors that are used in the similarity metric of section 4.

Target Class-Base Class	**Three Factors**			**Match using similarity metric**
Customer-Borrower	T_s=0.83	E_m=1.0	A_d=0.23	0.791
Customer-Agent	T_s=0.736	E_m=0.32	A_d=0.0	0.50

Table 3: Calculation using similarity metric

The similarity metric in LASER can thus correctly predict `Borrower` as a more generic class for the class `Customer`.

For our current experimental purposes LASER gathers all class names from the same repository and uses the similarity metric to suggest the most appropriate superclass for that class to extend. We measure the accuracy of this suggestion mechanism by second-guessing every extension relationship in our JRefactory repository, to determine whether LASER would assign the same parent class as that assigned by the original developer.

Overall, out of the 1259 classes in the repository, LASER correctly predicts the same superclass as the original developer in 60% of cases. The result suggests that WordNet can indeed act as the knowledge-based backbone on which an effective software-reuse system can be built.

6 Conclusion and Future Work

Analogical Reasoning enables a software environment to provide automated support for software reuse. In our case study we present some encouraging results to prove the linguistic applicability of WordNet to reasoning in software domain. This reasoning can allow LASER to apply lexically-driven analogy at the code level for better understanding of the developer's conceptual goals.

Future work on LASER will be focus on the use of structure-mapping analogy to supplement the level of linguistic reasoning described here, We predict that, given our current results, WordNet can provide a flexible substrate on which to build such a model of software analysis.

7 Acknowledgement

The authors would like to acknowledge the financial contribution of Faculty of Science, University College Dublin to this project.

References

1. Amin, R. and Ó Cinnéide , M. and Veale, T. ,LASER: A Lexical Approach to Analogy in Software Reuse, Mining Software Repositories Workshop, Edinburgh,2004.
2. Charles W. Krueger,Software Reuse, ACM Computing Surveys (CSUR), Volume 24 Issue 2,1992.
3. Scott Henninger, An Evolutionary approach to constructing Effective Software Reuse Repositories , ACM Transactions on Software Engineering and Methodology (TOSEM),1997.
4. Tracz, W.J. and Edwards, S, Implementation Working Group Report, Reuse In Practice Workshop, Software Engineering Institute, Pitt, Pa,1989.
5. George A. Miller, WordNet,Cognitive Science Laboratory, Princeton University.
6. JRefactory, An Open Source Refactoring Tool for Java, http://jrefactory.sourceforge.net.
7. Gomes, P. and Pereira, F.C. and Paiva, P. and Ferreira, J.L. and Bento, C., Supporting Creativity in Software Design, The AISB'02 Symposium, London, UK, April-2002.
8. Gomes, P. and Pereira, F.C. and Paiva, P. and Ferreira, J.L. and Bento, C., Experiments on Software Design Novelty using Analogy, The European Conference on Artificial Intelligence ECAI'02 Workshop:, Lyon, France, July-2002.
9. S. Kedar-Cabelli, Analogy from a unified perspective, In D.H. Helman (ed.), Analogical reasoning, Kluwer Academic, 1988.
10. G.Spanoudakis, P.Constantopoulos, "Similarity for Analogical Software Reuse: A Conceptual Modelling Approach", inProc. of CAiSE '93, Int. Conf. on Advanced Information Systems Engineering, Paris, June 1993
11. Cockburn A, Agile Software Development? Addison Wesley, Boston, 2002
12. Solveig ,B.,ROSA-Reuse of Object-Oriented Specifications through Analogy: A Project Framework, IFI Report 16, ISSN 0803-6489, 1994
13. Tracz, W.,Software Reuse Myths Revisited, in the proceedings of 16th International Conference on Software Engineering, May 16-21, 1994, Sorrento, Italy, pp 271-272.
14. Maiden, N. and A. Sutcliffe, Exploiting Reusable Specifications Through Analogy. Communications of the ACM, 1992. 35(4): p. 55-64.