JavaParser: A Fine-Grain Concept Indexing Tool for Java Problems

Roya Hosseini, Peter Brusilovsky

University of Pittsburgh, Pittsburgh, USA
{roh38,peterb}@pitt.edu

Abstract. Multi-concept nature of problems in the domain of programming languages requires fine-grained indexing which is critical for sequencing purposes. In this paper, we propose an approach for extracting this set of concepts in a reliable automated way using JavaParser tool. To demonstrate the importance of fine-grained sequencing, we provide an example showing how this information can be used for problem sequencing during exam preparation.

Keywords: indexing, sequencing, parser, java programming

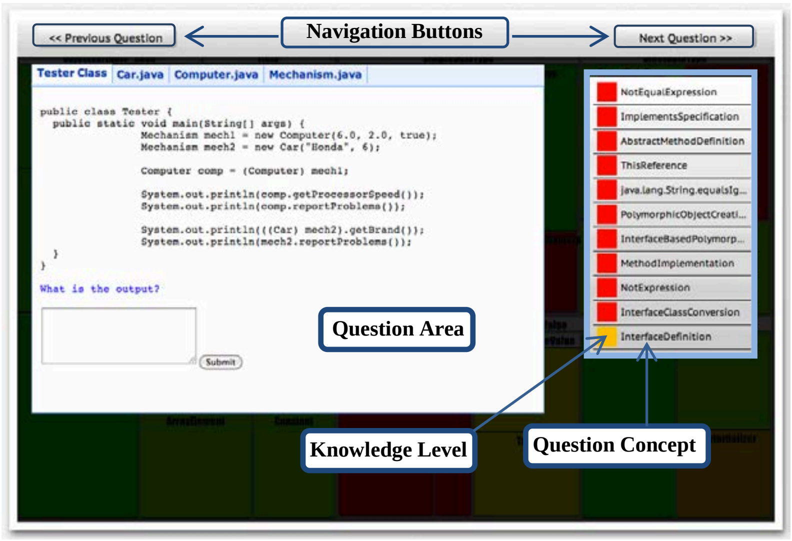
1 Introduction

One of the oldest functions performed by adaptive educational systems is guiding students to most appropriate educational problems at any time of their learning process. In classic ICAI and ITS system this function was known as task sequencing [1; 6]. In modern hypermedia-based systems it is more often referred as navigation support. The intelligent decision mechanism behind these approaches is typically based on a domain model that decomposes the domain into a set of knowledge units. This domain model serves as a basis of student overlay model and as a dictionary to index educational problems or tasks. Considering the learning goal and the current state of student knowledge reflected by the student model, various sequencing approaches are able to determine which task is currently the most appropriate.

An important aspect of this decision process is the granularity of the domain model and the related granularity of task indexing. In general, the finer are the elements of the domain model and the more precise is task indexing, the better precision could be potentially offered by the sequencing algorithm in determining the best task to solve. However, fine-grained domain models that dissect a domain into many dozens to many hundreds of knowledge units are much harder to develop and to use for indexing. As a result, many adaptive educational systems use relatively coarse-grained models where a knowledge unit corresponds to a considerably-sized topic of learning material, sometimes even a whole lecture.. With these coarse-grain models, each task is usually indexed with just 1-3 topics. In particular, this approach is used by the majority of adaptive systems in the area of programming [2; 4; 5; 7].

Our past experience with adaptive hypermedia systems for programming [2; 4] demonstrated that adaptive navigation support based on coarse grain problem indexing is surprisingly effective way to guide students over their coursework, yet it doesn’t work well in special cases such as remediation or exam preparation. In these special situations students might have a reasonable overall content understanding (i.e., coarse-grain student model registers good knowledge), while still possessing some knowledge gaps and misconceptions that could be only registered using a finer-grain student model.. In this situation only a fine-grain indexing and sequencing is able to suggest learning tasks that can address these gaps and misconceptions.

To demonstrate the importance of fine-grained indexing, we can look at an example of a system called *Knowledge Maximizer* [3] that uses fine-grain concept-level problem indexing to identify gaps in user knowledge for exam preparation. This system assumes a student already did considerable amount of work and the goal is to help her define gaps in knowledge and try to fix that holes as soon as possible. Fig. 1 represents the Knowledge Maximizer interface. The question with the highest rank is shown first. User can navigate the ranked list of questions using navigation buttons at the top. Right side of the panel shows the list of fine-grained concepts covered by the question. The color next to each concept visualizes the student's current knowledge level (from red to green). Evaluation results confirm that using fine-grained indexing in Knowledge Maximizer has positive effect on students' performance and also shorten the time for exam preparation.



```
public class Tester {
  public static void main(String[] args) {
                Mechanism mech1 = new Computer(6.0, 2.0, true);
                Mechanism mech2 = new Car("Honda", 6);

                Computer comp = (Computer) mech1;

                System.out.println(comp.getProcessorSpeed());
                System.out.println(comp.reportProblems());

                System.out.println(((Car) mech2).getBrand());
                System.out.println(mech2.reportProblems());
  }
}
```

What is the output?

🟥	NotEqualExpression
🟥	ImplementsSpecification
🟥	AbstractMethodDefinition
🟥	ThisReference
🟥	java.lang.String.equalsIg...
🟥	PolymorphicObjectCreati...
🟥	InterfaceBasedPolymorp...
🟥	MethodImplementation
🟥	NotExpression
🟥	InterfaceClassConversion
🟨	InterfaceDefinition

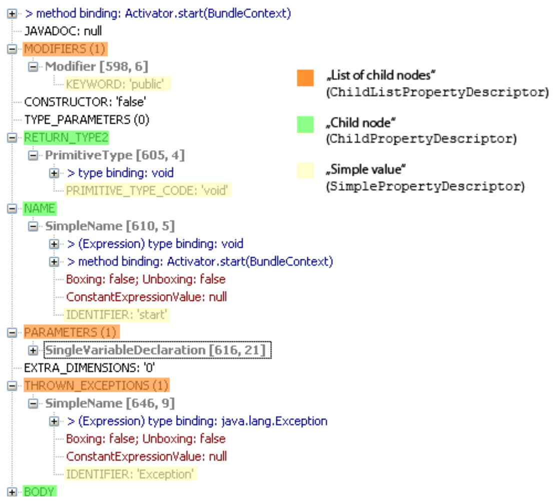
Fig. 1. The Knowledge Maximizer interface.

The problem with finer-grain indexing, such as used by the Knowledge Maximizer is the high cost of indexing. While fine-grain domain model has to be developed just once, the indexing process has to be repeated for any new question. Given that most complex questions used by the system include over 90 concepts each, the high cost of indexing effectively prevents an expansion of the body of problems. To resolve this problem, we developed an automatic approach for fine-grained indexing for programming problems in Java based on program parsing. This approach is presented in the following section.

2 Java Parser

Java parser is a tool that we developed to index Java programs with concepts of Java ontology developed by our group (http://www.sis.pitt.edu/~paws/ont/java.owl). This tool provides the user with semi-automated indexing support during developing new learning materials for the Java Programming Language course. This parser is developed using the Eclipse Abstract Syntax Tree framework. This framework generates an Abstract Syntax Tree (AST) that entirely represents the program source. AST consists of several nodes each containing some information known as *structural properties*. For example, Fig. 2 shows structural properties for the following method declaration:

```
public void start(BundleContext context) throws Exception {
        super.start(context);
}
```



```
⊞ > method binding: Activator.start(BundleContext)
  JAVADOC: null
⊟ MODIFIERS (1)
    ⊟ Modifier [598, 6]
        KEYWORD: 'public'
  CONSTRUCTOR: 'false'
  TYPE_PARAMETERS (0)
⊟ RETURN_TYPE2
    ⊟ PrimitiveType [605, 4]
        ⊞ > type binding: void
        PRIMITIVE_TYPE_CODE: 'void'
⊟ NAME
    ⊟ SimpleName [610, 5]
        ⊞ > (Expression) type binding: void
        ⊞ > method binding: Activator.start(BundleContext)
        Boxing: false; Unboxing: false
        ConstantExpressionValue: null
        IDENTIFIER: 'start'
⊟ PARAMETERS (1)
    ⊞ SingleVariableDeclaration [616, 21]
  EXTRA_DIMENSIONS: '0'
⊟ THROWN_EXCEPTIONS (1)
    ⊟ SimpleName [646, 9]
        ⊞ > (Expression) type binding: java.lang.Exception
        Boxing: false; Unboxing: false
        ConstantExpressionValue: null
        IDENTIFIER: 'Exception'
⊞ BODY
```

🟧	„List of child nodes" (ChildListPropertyDescriptor)
🟩	„Child node" (ChildPropertyDescriptor)
🟨	„Simple value" (SimplePropertyDescriptor)

Fig. 2. Structural properties of a method declaration

Table 1. Sample of JavaParser output

Source	Output
`public void start(BundleContext context) throws Exception { super.start(context); }`	Super Method Invocation, Public Method Declaration, Exception, Formal Method Parameter, Single Variable Declaration, Void

After building the tree using *Eclipse AST API*, the parser performs a semantic analyzed using the information in each node. This information is used to identify fine-grained indexes for the source program. Table 1 shows the output concepts of *JavaParser* for the code fragment mentioned above. Note that the goal of the parser is to detect the lowest level ontology concepts behind the code since the upper level concepts can be deduced using ontology link propagation. For example, as you see in Table 1, parser detects “void” and “main” ignoring upper-level concept of “modifier”.

We compared the accuracy of *JavaParser* with manual indexing for 103 Java problems and found out that our parser was able to index 93% of the manually indexed concepts. Therefore, automatic parser can replace time-consuming process of manual indexing with a high precision and open the way to community-driven problem authoring and targeted expansion of the body of problems.

3 Conclusion

Having fine-grained indexing for programming problems is necessary for better sequencing of learning materials for students; however, the cost of manual fine-grained indexing is prohibitively high. In this paper, we presented a fine grained indexing approach and tool for automatic indexing of Java problems. We also showed an application of fine-grained problem indexing during exam preparation where small size of knowledge units is critical for finding sequence of problems that fills the gaps in student knowledge. Results show that proposed automatic indexing tool can offer the quality of indexing that is comparable with manual indexing by expert for a fraction of its cost.

References

1. Brusilovsky, P.: A framework for intelligent knowledge sequencing and task sequencing. In: Proc. of Second International Conference on Intelligent Tutoring Systems, ITS'92. Springer-Verlag (1992) 499-506
2. Brusilovsky, P., Sosnovsky, S., Yudelson, M.: Addictive links: The motivational value of adaptive link annotation. New Review of Hypermedia and Multimedia 15, 1 (2009) 97-118
3. Hosseini, R., Brusilovsky, P., Guerra, J.: Knowledge Maximizer: Concept-based Adaptive Problem Sequencing for Exam Preparation. In: Proc. of the 16th International Conference on Artificial Intelligence in Education. (2013) In Press
4. Hsiao, I.-H., Sosnovsky, S., Brusilovsky, P.: Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming. Journal of Computer Assisted Learning 26, 4 (2010) 270-283
5. Kavcic, A.: Fuzzy User Modeling for Adaptation in Educational Hypermedia. IEEE Transactions on Systems, Man, and Cybernetics 34, 4 (2004) 439-449
6. McArthur, D., Stasz, C., Hotta, J., Peter, O., Burdorf, C.: Skill-oriented task sequencing in an intelligent tutor for basic algebra. Instructional Science 17, 4 (1988) 281-307
7. Vesin, B., Ivanović, M., Klašnja-Milićević A., Budimac, Z.: Protus 2.0: Ontology-based semantic recommendation in programming tutoring system. Expert Systems with Applications 39, 15 (2012) 12229-12246