Philipps-Universität Marburg
Department of Mathematics and Computer Science
Programming Languages and Tools group

Comparison of Java Bytecode Toolkits

Context: The ModBEAM Project

Java bytecode is a very versatile intermediate representation to which several languages (Java, Scala, Groovy, etc.) are compiled. It is always available even for closed-source code therefore targeted by many modern code analysis tools. The programming languages and tools group is developing a toolkit for program analyses and transformations of Java bytecode, named *Modular Bytecode Engineering and Analysis based on Models* (*ModBEAM*) [1, 2]. It contains a *meta model*[1] of Java bytecode defined in the ECORE format of the "Eclipse Modeling Framework" (EMF), the prevalent standard in model-driven engineering. ModBEAM also provides and *Eclipse plug-in* which can represent Java bytecode files as *instance models*[2] according to its meta model. This allows the usage of a wide variety of powerful model analysis and transformation tools, available in the EMF ecosystem, to inspect and possibly modify the bytecode model. For possibly transformed bytecode models, the ModBEAM plug-in can again generate regular Java bytecode.

Info

Java Bytecode	
Java	
ASM, BCEL, Javasssist	
Theory	▰▰▰▰▰▰▱▱
Practice	▰▰▱▱▱▱▱▱
Thesis (BSc, MSc)	
Prof. Christoph Bockisch	

Problem Statement

Beside the ModBEAM tool developed in the research group, several other bytecode toolkits exist. In fact, ModBEAM internally uses the ASM toolkit to read and write bytecode to andf from it's own format (i.e. ModBEAM models). The various bytecode toolkits each have different philosophies for representing and handling Java bytecode. However, no systematic comparison between the existing tools exists.

Assignment

The bytecode toolkits should be compared by different attributed that also have to be identified in this assignment. An initial set of relevant attributes are the supported Java versions, how actively they are maintained and their performance. In particular, the data models for the bytecode representation should be compared and tasks identified that are especially well or badly supported by the tools. Ideally, small demonstrators for representative tasks should be implemented using each bytecode toolkit. This should facilitate a comparison if how suitable they are for each task. The comparison should include ModBEAM and the most well-known and widely used Java bytecode toolkits. Bytecode toolkits that still actively maintained are: Javassist, ASM, BCEL and WALA/Shrike.
Just like ModBEAM, also other toolkits build on top of the previously mentioned core toolkits, for example ByteBuddy which internally uses ASM. But unlike ModBEAM, these "second tier" toolkits typically focus on a domain-specific use case, whereas the "first tier" toolkits and ModBEAM aim to be general-purpose. The focus of this study should be on the general-purpose approaches. However, including domain-specific approaches would be an interesting extension to this assignment.

References

[1] C. Bockisch, G. Taentzer, N. Nassar, and L. Wydra. Java bytecode verification with OCL: Why, how and when? *Journal of Object Technology*, 19(3):3:1–16, October 2020.

[1]If the term meta model is not known, think of is as the classes in an object-oriented program.
[2]Assuming a meta model is comprised of classes, think of an instance model as thier runtime objects in a specific execution.

[2] B. M. Yildiz, C. Bockisch, A. Rensink, and M. Aksit. An MDE approach for modular program analyses. In J. B. Sartor, T. D'Hondt, and W. D. Meuter, editors, *Companion to the 1st Int. Conf. on the Art, Science and Engineering of Programming*, pages 15:1–15:5. ACM, 2017.

Resources

- Javassist: `https://www.javassist.org/`
- ASM: `https://asm.ow2.io/`
- BCEL: `https://commons.apache.org/proper/commons-bcel/`
- WALA: `https://github.com/wala/WALA`
- ModBEAM: `https://gitlab.uni-marburg.de/fb12/plt/modbeam-mt/modbeam`