Summary

AspectJ is a Java extension that adds the concept of Aspect-Oriented Programming. It offers a framework for modularization and an easy (and simple) way of observing and manipulating program flows (with join points and advices). Observation is pre-defined by AspectJ's join points [1] and manipulation by its advices [2]. Depending on the implementation we would want to achieve, AspectJ could offer everything we need without low-level aspects of Bytecode Engineering. A pre-runtime analysis is not part of AspectJ and the structure of classes can only be analyzed in an elaborated implementation. The strength of AspectJ is the high-level approach that still offers a wide variety of tools for possible observation- and manipulation-needs without the need of implementing low-level constructs.

[1] http://www.eclipse.org/aspectj/doc/next/progguide/semantics-joinPoints.html

[2] http://www.eclipse.org/aspectj/doc/next/progguide/quick-advice.html

ASM is a Bytecode Engineering library. With ASM one can analyze and manipulate programs and dynamically generate classes at runtime (load time). It is a lower level approach than AspectJ and operates directly on the code of programs instead of its flow. ASM offers two APIs [3]: one as a visitor-based approach to bytecode, the other as a tree-based object structure. In the context of trace monitoring we would need to use ASM to inject Tracer calls into methods at a bytecode level. It is obvious that this approach is very powerful but also implies lower-level coding than AspectJ.

[3] http://download.forge.objectweb.org/asm/asm4-guide.pdf