the certifying compiler for properties like well typedness or safe memory access. As the certifying compiler is designed to be completely automatic, it will not be able to deal with rich functional or security properties.

We propose a verification framework with the following features:

compiler from source program annotation into bytecode annotation. Thus, bytecode

`with`

theorem prover

3 Bytecode Specification Language (BCSL)

In this section, we introduce a bytecode specification language which we call BCSL (short for ByteCode Specification Language). BCSL is based on the design principles of JML (Java Modeling Language), which is a behaviorial interface specification language following the design by contract approach [2].

Before going farther, we give a flavour of what JML specifications look like. Fig. 2 shows an example of a Java class and its JML (based)4j 20.3897 2 Td (of)T5894.3882 0 Td (an)4j 15.23

Specification clauses in BCSL that are taken from JML and inherit their semantics directly from JML include:

[illegible]

Thus the \JML compiler"[2] compiles the JML source specification into user defined attributes. The compilation process has three stages:

1. compile the Java source file. This can be done by any Java compiler that supplies for every method in the generated class

```
JMLLoop_specification_attribute {
    ...
    {   u2 index;
        u2 modifies_count;
        formula modifies[modifies_count];
        formula invariant;
        expression decreases;
    }
```

substitution lemmas.

Returning back to the example, the expression c **and** st(c) **stand respbac**

resulting predicate is quantified over the expressions that may be modified by the called method. We also assume that if the invoked method terminates

related to the fact that the result type is boolean but the JVM encodes boolean expressions as integers (which is trivially true). This means that the proof obligations have also the same shape.

Another imp

[9]