A Formal Approach for Specifying Access Control Security Features of Java Modules

Carlos E. Rubio Medrano
M.S. Student in Computer Science
Department of Computer Science
The University of Texas at El Paso
Advisor: Yoonsik Cheon, Ph.D.

ABSTRACT

Computer security has become a crucial issue in recent years. Access control techniques allow developers to specify and protect access to sensitive resources within a system by specifying access control constraints to be enforced either by the system itself or by an external monitor [1]. We propose a formal approach for specifying such constraints and enforcing them at runtime. As the specification of access control constraints is written in a formal behavioral interface specification language such as JML, it is declarative and can be checked at runtime by the runtime assertion checker of the specification language.

SIGNIFICANCE

- Misure of sensitive resources, such as system files, database files, network ports, among others, has been recognized as an important source of malicious or unintended malfunctions of software systems.
- By means of a formal specification and runtime checking, access to sensitive resources can be better controlled, avoiding possible vulnerabilities in the overall security of the system.
- The specification of access control constraints becomes a part of the contract of a module.

BACKGROUND

The Java Modeling Language (JML)

The Java Modeling Language (JML) [2] is a behavioral interface specification language for Java to describe the behavior of contracts of program modules such as classes and interfaces. In JML, a contract between a client of a module and its implementer is specified by using pre-conditions, post-conditions, and frame conditions, among others. The semantics of a contract is that: if a method’s pre-condition holds—i.e., it has been completely satisfied by its client—before the execution of m, then m’s post-condition can be assumed once m’s execution is completed. A JML specification or contract is commonly added to Java source code files as a special annotation comment.

APPENDIX DESCRIPTION

The key idea of our approach is to extend the JML language to be able to write assertions about access control features of a Java program module. We introduce two facilities for this, one based on the JVM’s permission system and the other based on the visibility of the Java programming language. Thanks to these extensions, an access control contract can be written as a part of the interface specification of a module.

Permission-based Approach

The first approach introduces a new JML expression to express the permissions that a Java module needs to be granted to behave correctly. This new ‘permission expression’ takes as a parameter the permission to be evaluated, which is represented by a constructor signature of a class derived from the java.security.Permission class (see an example below). The permission expression evaluates to true if the permission checking process (see below) grants the specified permission; otherwise, it evaluates to false.

In the example above, the precondition of the setSSN method is extended with a specification of access control constraints. The permission expression states that the client should have an appropriate EmployeePermission permission to call the method, and the called_by expression states that only the corp.office.Manager class can call this method. Both are boolean expressions, and thus can be used in the precondition.

IMPLEMENTATION

A prototype tool for the proposed approach has been implemented by using Polyglot [3], an extensible Java compiler framework developed at Cornell University. The Polyglot framework facilitates the implementation of an extension to the Java language by defining a new, extended grammar based on the standard Java grammar. The JML annotations, containing access control contracts, are first syntax- and type-checked and then translated to plain Java programs by translating programs to runtime assertion checking code. The translated Java source code can be compiled with any Java compilers such as javac.

CURRENT STATUS

The syntax and informal semantics of the extension to JML have been defined, including the semantics of inheritance. A prototype tool has been implemented to translate access control contracts into runtime checking code at the source code level. A case study is being performed to evaluate the approach by using a realistic application.

CONCLUSION

We made two small extensions to the JML language in order to formally specify the access control features of a Java program module. The first extension is based on the permission system currently in use by JVM. The second extends the access modifiers of the Java programming language in order to specify the modules that can access a protected resource. A prototype tool has been implemented to translated both extensions to runtime assertion checking code. A case study is being performed to evaluate the effectiveness of the extensions.

REFERENCES


This work is supported in part by the National Science Foundation under Grants No. CNS-0509299 and No. CNS-0540592, the Computing Alliance of Hispanic Serving Institutions and the Mexican Chihuahua State Government under a collaboration agreement with The University of Texas at El Paso.