RAC for JML using Prolog

Presented By: Amritam Sarcar

Seminar Outline

- Overview
- Motivation
- Current Work
- Problem
- Objective
- My Approach
  - Introduction
  - Formalization
  - Top Level Design
  - Integration with Eclipse Architecture
- Demo
- Conclusion
- Future Work

Overview

- The Java Modelling Language (JML) is the most popular BISL for Java.
- Tools exist: RAC to FSPV with ESC.
- Mainstream developer technologies: RAC and ESC

Motivation(1/2)

- **Literature Survey**
  - Symbolic Animation of JML Specifications - F.Bouquet, et.al
  - A Prolog-oriented extension of Java programming based on generics and annotations - Mirko Viroli, et.al
  - A Verified Compiler For A Structured Assembly Language - P.Curzon

Motivation(2/2)

- Prolog is a logic programming language.
- Associated with AI and computational linguistics.

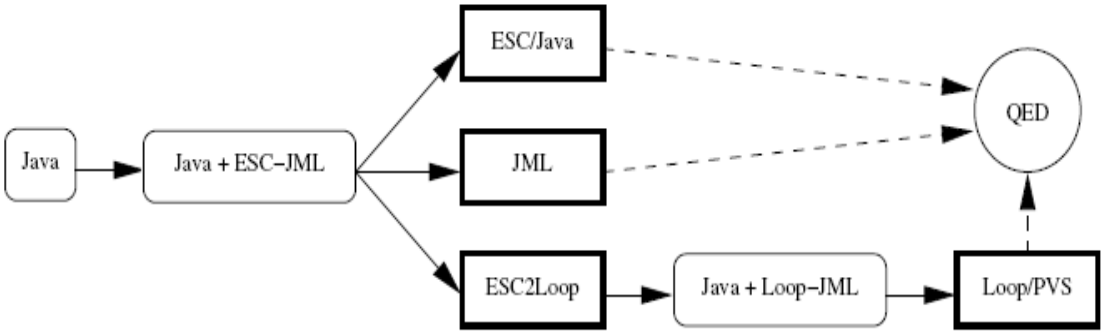
- *Why not use Prolog as an implementation language ?*
- *Can Prolog be used for RAC implementation (at least a subset of it)?*

Current Work

- Extending a Java compiler, already integrated within a modern IDE.
- Maintenance assured by third party developers.
- The actual RAC implementation is done using Wrapper Classes, an integral part of Java.

Problem?

- The existing implementation cannot be proved using Theorem Prover



Objectives

- In this project, a formal semantics would be developed for essentially a part of the sequential Java.

- The J2PL tool would be able to support only JML, so that it can verify JML-annotated Java source code.

Proposed Solution

- J2PL translates source-level code from Java to Prolog
  - It is in a logical theory format that can serve as input for theorem provers.
  - It can be used to prove properties of the Java program, thus achieving a high level of reliability for this program.
- Formal specification language provides tool support.

My Approach : Introduction

- Formalization of Java classes with JML annotations into Prolog syntax.

- Compilation of .java classes into .pl code.
  - Only RAC generated code.

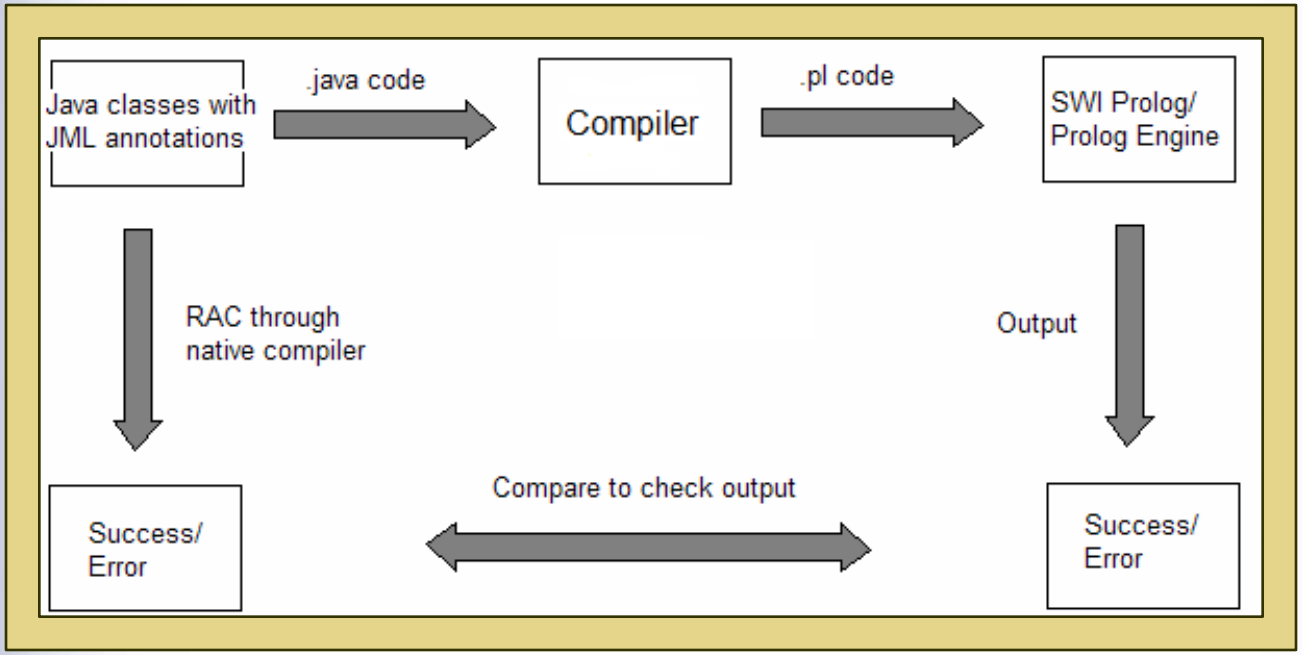
My Approach : Formalization

- PROLOG syntax:

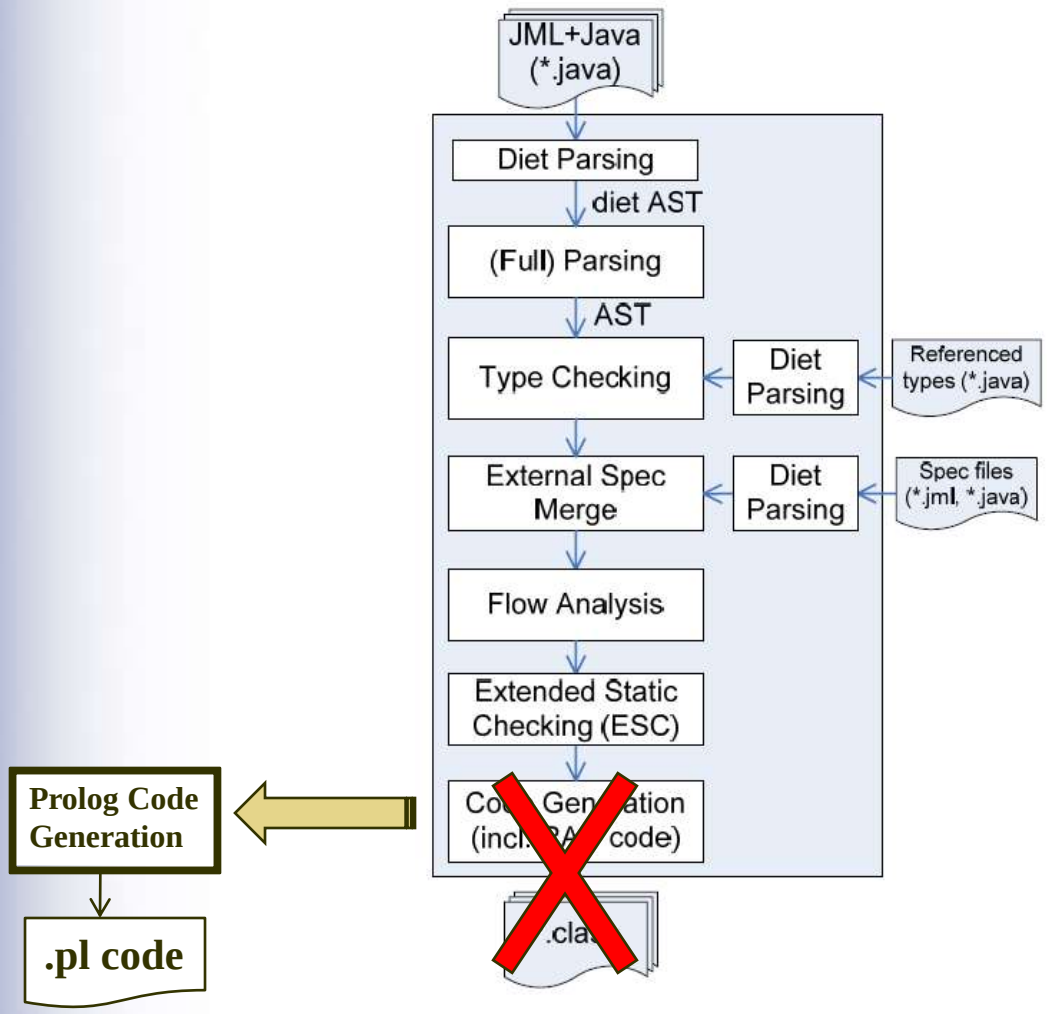
- **specification(*spec-name).***
- **declaration(*spec-name, data-kind, data-name, data-type).***
- **operation(*spec-name, operation-name).***
- **predicate(*spec-name, pred-kind, pred-id, predicate)***

- data-kind: static | variable | input(*op-name)* | *output(op-name)* | *local(op-name)*
- data-type: atom | int| set(data-type) | pair(data- type, data-type)
- pred-kind: static | invariant | initialization | pre(*op-name)* | *post(op-name)*

My Approach : Top Level Design



My Approach :
Integration with Eclipse Architecture



Demo

Template

class_<class name> :- Field Declaration,
Call Main.

Constructor declaration: - Body of the Constructor

Methods:- Pre_Spec(),
Body of the Method (contains sequence of statements),
Post_Spec().

Pre_Spec_<Method Name>:- JML Clause.

Post_Spec_<Method Name> :- JML Clause.

Conclusion

- Exhaustive test cases has to be generated to verify RAC Implementation.
  - The JML Group has generated approx.700 test cases.
  - JUnit test cases can be used.
- From initial results so far obtained, a subset of RAC implementation can be achieved.
- With sufficient knowledge of Eclipse, RAC Implementation using Prolog can be integrated into it.

Future Work

- Judiciously extend the subset of RAC Implementation.
  - Include Level 0 and Level 1 JML annotations.
  - Include floating points.
  - Include concepts of Object Orientation.
    - Eg : Objects, polymorphism, inheritance.
- Execute generated RAC code from within Java.
- The .pl code obtained should be verified using theorem prover.

THANK YOU