Code Generation for Event-B

Krikun Georgiy

Innopolis University

Agenda

Formal Methods

The B Method

The Event-B Methodology

Eiffel Programming Language + Design by Contract

Rodin (the Event-B IDE)

Work plan until the end of the thesis

Formal Methods

Introduction Formal Methods

Formal specification languages are mathematically-based languages whose purpose is to aid the construction of systems and software.

Roughly, formal design can be seen as a three step process:

1. Formal Specification
2. Verification
3. Implementation

Introduction Formal Methods

Benefits:

- Discipline
- Precision

Weaknesses:

- Expense
- Limits Of Computational Models
- Usability

Formal Specification Languages

Model-Based Languages:

Z, VDM, B

Finite State-Based Languages:

FSMs, SDL, Statecharts, X-machines

Process Algebra State-Based Languages:

CSP, CCS, LOTOS

Hybrid Languages:

CHARON

The B Method

The B Method

Developed by Jean-Raymond Abrial

Approach:

Starts from abstract model of a system

Each refinement steps adds more details, provably consistent

Obtain precise model which transform into an implementation

The B Models

Use predicate calculus to model properties

B models are called machines or Abstract Machines
is given by:
- **State** (variable set, state invariant) is the static part
- a set of **Operations**, can modify the State, dynamic part

For each operation must be proved that the specification preserves the invariant (Proof Obligation)

Proof Obligations based on the Substitution Principle

The Event-B Methodology

The Event-B Methodology

Formal method is derived from the B method

Event-B models:

- machines (the dynamic part. e.g. variables, invariants, events)
- contexts (the static part. e.g. carrier sets, constants)

Basic relationships between machines and contexts:

- a machine **sees** a context
- a machine can **refine** another machine
- a context can **extend** another context

General structure

<machine_identifier >
refines
< machine_identifier >
sees
< context_identifier_list >
variables
< variable_identifier_list >
invariants
< label >: < predicate >
variants
< variant >
events
< event_list >

<context_identifier >
extends
< context_identifier_list >
sets
< set_identifier_list >
constants
< constant_identifier_list >
axioms
< label >: < predicate >

Eiffel Programming Language + Design by Contract

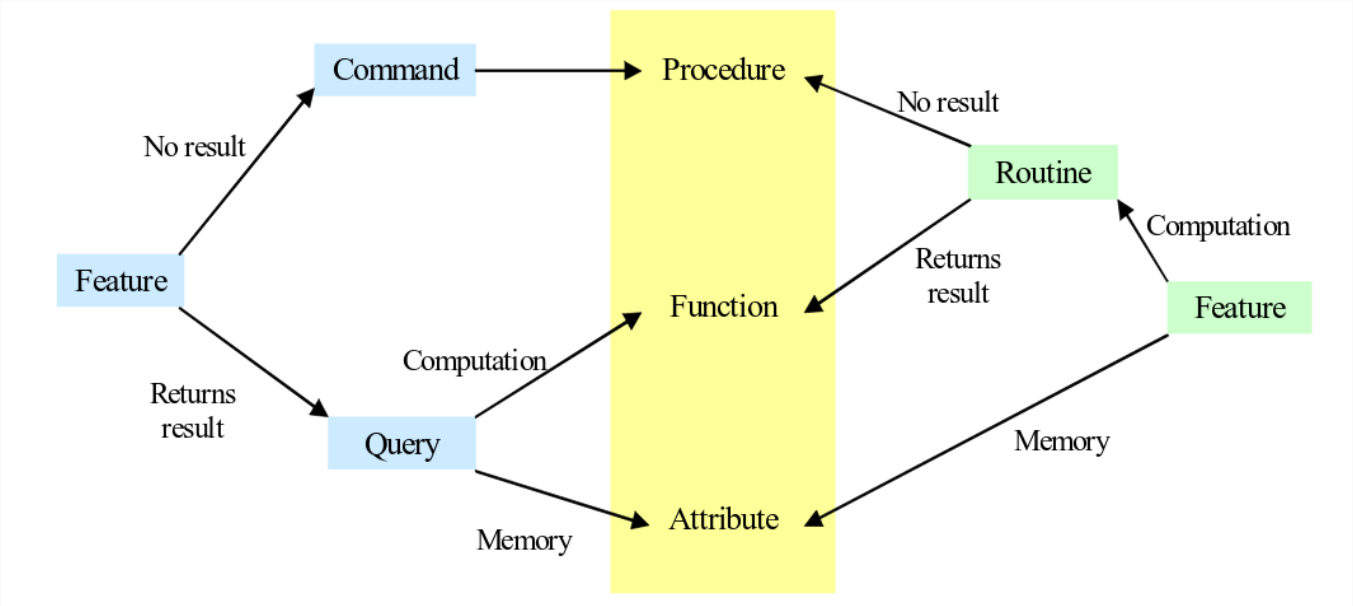
Eiffel Programming Language

OOP language designed by Bertrand Meyer and Eiffel Software

Structure of an Eiffel program:
class → cluster → system → universe

class - a set of features (attribute or routines)

another classification - by role (commands and queries)



Design principles

Method principles:

- Command/Query Separation Principle
- Information Hiding
- Uniform Access

Design by Contract

Design by Contract is a method of software construction, which suggests building software systems that will cooperate on the basis of precisely defined contracts.

Different kinds of contracts:

- Preconditions
- Postconditions
- Class invariants
- Check instructors
- Loop invariants
- Loop variants

Design by Contract

Benefits:

- Software correctness
- Documentation
- Debugging and testing
- Management

Rodin (the Event-B IDE)

Rodin (the Event-B IDE)

The Rodin Platform is an Eclipse-based IDE for Event-B

(The Rigorous Open Development Environment for Complex Systems)

provides a set of tools for Event-B models:

- editor
- proof generator
- provers

plugins provide extended functionality:

- code generators
- model checking
- animation
- visualization
- etc

Existing code generators

EventB2Java

EventB2JML

EventB2Dafny

EventB2SQL

EB2ALL

B2C

EHDL

Work plan until the end of the thesis

Work-plan

1. Modeling code generator in Event-B as case study
2. Generate code with EventB2Java
3. Adapt code as plugin to Rodin