Lightweight Static Verification of [UML] Executable Models

Elena Planas
SET Seminar
03/10/2012
Introducing me

PhD student *
  at Technical University of Catalonia (Spain)
  * under the supervision of Dr. Jordi Cabot and Dra. Cristina Gómez

Currently working
  at Open University of Catalonia (Spain)

Visiting researcher
  at TU/e (during 3 months)

Topics of interest:
  – Software development paradigms: MDD, MDA
  – Conceptual modelling
  – UML, OCL, ...
  – Quality of software models
  – V&V
Why this presentation?

1. To present my work

2. To identify further collaborations
Why this presentation?

1. To present my work

2. To identify further collaborations
Context

MD*

MDE

MDD

MDA
Motivation

- Most errors in software development are introduced during the first steps.
- The later an error is removed, the more expensive the fix is.
Motivation

- Most errors in software development are introduced during the **first steps**.
- The **later** an error is removed, the **more expensive** the fix is.

In MDD, the **quality** of the models directly impact on the quality of the final system derived from them.
Motivation

- Most errors in software development are introduced during the first steps.
- The later an error is removed, the more expensive the fix is.

Boehm’s curve

In MDD, the quality of the models directly impact on the quality of the final system derived from them.

Need for useful methods and tools to check the correctness of models.
Our **goal** is...

To develop a set of **lightweight static** verification methods for checking several **correctness properties** of [UML] executable models
Our **goal** is...

To develop a set of **lightweight static verification methods** for checking several correctness properties of [UML] executable models.
Verification methods classification

Regarding the mode how the analysis is done

Regarding the level of formalization they use
Verification methods classification

Regarding the mode how the analysis is done

Dynamic methods

Static methods

+ Regarding the level of formalization they use
Verification methods classification

Regarding the mode how the analysis is done

Dynamic methods

Static methods

- Non-formal methods
  - Lightweight methods
  - Formal methods

Regarding the level of formalization they use
Verification methods classification

Regarding the mode how the analysis is done

Dynamic methods

Static methods

Non-formal methods

Inspections

Reviews

Lightweight methods

Testing

Our verification methods

Formal methods

Model Checking

Abstract Interpretation

Regarding the level of formalization they use
Verification methods classification

- **Regarding the mode how the analysis is done**
  - **Static methods**
    - Inspections
    - Reviews
  - **Dynamic methods**

- **Regarding the level of formalization they use**
  - **Non-formal methods**
  - **Lightweight methods**
    - Our verification methods
  - **Formal methods**
    - Model Checking
    - Abstract Interpretation

- Static analysis → no execution of the model
- Do not need to translate the model into a mathematical formalization
- They provide quick and useful feedback
- They can be integrated in the development process
- They are only able to perform a partial analysis
Our **goal** is...

To develop a set of **lightweight static verification methods** for checking several **correctness properties** of [UML] **executable models**
An **executable model** is a **model** with a behavioral specification sufficiently **detailed** so it can be systematically implemented/executed in the production environment.
An **executable model** is a **model** with a behavioral specification sufficiently **detailed** so it can be systematically implemented/executed in the production environment.

**Use of executable models in MDD**

1. **Software engineers** create fully executable models.
2. **Software engineers iteratively execute, test and update the models**.
3. **Code generation** and **Model interpretation**.
4. **The models are V&V in a development/test environment**.
5. **The models are deployed in a production environment**.
An **executable model** is a model with a behavioral specification sufficiently **detailed** so it can be systematically implemented/executed in the production environment.

(1) **↑** level of abstraction  →  **↑** productivity
(2) platform independent models  →  **↓** costs
(3) early verification  →  **↑** quality
**executable models** may be specified in several languages
**executable models** may be specified in several languages

**UML executable model** = **Structural Model**
UML Class Diagram + integrity constraints + **Detailed Behavioral Model**
UML Behavioral Diagram precisely defined
Executable models may be specified in several languages.

UML executable model = Structural Model
UML Class Diagram + integrity constraints + Detailed Behavioral Model
UML Behavioral Diagram precisely defined
executable models may be specified in several languages

UML executable model = Structural Model
UML Class Diagram + integrity constraints + Detailed Behavioral Model
UML Behavioral Diagram precisely defined

RestaurantBranch
address: String
phone: String[0..2]

Menu
name: String
price: Real

Course
description: String
category: CourseCategory

SpecialMenu
discount: Real

context SpecialMenu inv validDiscount: self.discount >=10
context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances()->size()<=3

<<enumeration>>
CourseCategory
Starter
MainCourse
Dessert
**executable models** may be specified in several languages

- **UML executable model**
  - Structural Model
    - UML Class Diagram + integrity constraints
  - Detailed **Behavioral Model**
    - UML Behavioral Diagram precisely defined
executable models may be specified in several languages

UML executable model = Structural Model
UML Class Diagram + integrity constraints + Detailed Behavioral Model
UML Behavioral Diagram precisely defined

Using Alf action language (OMG). Alf is a clear, precise yet abstract textual language to specify executable models in the context of UML
**executable models** may be specified in several languages

- **UML executable model**
- **Structural Model**
  - UML Class Diagram + integrity constraints
- **Detailed Behavioral Model**
  - UML Behavioral Diagram precisely defined

Using **Alf action language** (OMG). Alf is a clear, precise yet abstract textual language to specify executable models in the context of UML.

```
activity addMenu (in _name: String, in _price: Real, in _courses:Course[3..*]) {
    if (!Menu.allInstances()->exists(m|m.name=_name) ) {
        Menu m = new Menu();
        m.name = _name;
        m.price = _price;
        for ( i in 1.._courses->size() ) {
            IsComposedOf.createlink(m=>menu,course=>_courses[i]);
        }
    }
}
```
executable models may be specified in several languages

UML executable model = Structural Model UML Class Diagram + integrity constraints + Detailed Behavioral Model UML Behavioral Diagram precisely defined

Using Alf action language (OMG). Alf is a clear, precise yet abstract textual language to specify executable models in the context of UML

activity addMenu (in _name: String, in _price: Real, in _courses:Course[3..*]) {
    if (!Menu.allInstances()->exists(m|m.name=_name)) {
        Menu m = new Menu();
        m.name = _name;
        m.price = _price;
        for (i in 1..<_courses->size()) {
            IsComposedOf.createlink(m=>menu,course=>_courses[i]);
        }
    }
}
Our **goal** is...

To develop a set of **lightweight static verification methods** for checking several **correctness properties** of **[UML]** executable models.
Property #1. **Non-Redundancy**
An action in operation is **redundant** if its effect on the system state is subsumed by the effect of later actions in the same operation.
Property #1. **Non-Redundancy**

An action in operation is **redundant** if its effect on the system state is subsumed by the effect of later actions in the same operation.

```ruby
activity removeCourse () {
  self.description = null;
  self.category = null;
  self.destroy();
}
```

<table>
<thead>
<tr>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>description: String</td>
</tr>
<tr>
<td>category: CourseCategory</td>
</tr>
</tbody>
</table>
An action in operation is **redundant** if its effect on the system state is subsumed by the effect of later actions in the same operation.
Property #1. **Non-Redundancy**

An action in operation is **redundant** if its effect on the system state is subsumed by the effect of later actions in the same operation.

**Course**

<table>
<thead>
<tr>
<th>description: String</th>
</tr>
</thead>
<tbody>
<tr>
<td>category: CourseCategory</td>
</tr>
</tbody>
</table>

**Feedback:** There is no need of clearing the values of the attributes of an object that is going to be removed.

```ruby
activity removeCourse () {
  self.description = null;
  self.category = null;
  self.destroy();
}
```
Property #1. **Non-Redundancy**

An action in operation is **redundant** if its effect on the system state is subsumed by the effect of later actions in the same operation.

```java
activity removeCourse () {
    self.destroy();
}
```
Property #2. **Executability**
The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a **chance** that a user may successfully execute the operation.  

An operation is **strongly executable** when it is **always** successfully executed.
Property #2. **Executability**

The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a **chance** that a user may successfully execute the operation. An operation is **strongly executable** when it is **always** successfully executed.

```plaintext
activity classifyAsSpecialMenu (in _discount:Real) {
    if ( _discount>=10 ) {
        classify self to SpecialMenu;
        self.discount = _discount;
    }
}
```

<table>
<thead>
<tr>
<th>Menu</th>
<th>context SpecialMenu inv validDiscount: self.discount &gt;=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances()-&gt;size()&lt;=3</td>
</tr>
<tr>
<td>price: Real</td>
<td></td>
</tr>
</tbody>
</table>

SpecialMenu

discount: Real

{incomplete}
Property #2. **Executability**

The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a **chance** that a user may successfully execute the operation.

An operation is **strongly executable** when it is **always** successfully executed.

```
Menu

<table>
<thead>
<tr>
<th>name: String</th>
<th>price: Real</th>
</tr>
</thead>
</table>

SpecialMenu

context SpecialMenu inv validDiscount: self.discount >=10
context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances()->size()<=3

activity classifyAsSpecialmenu (in _discount:Real) {
  if ( _discount>=10 ) {
    classify self to SpecialMenu;
    self.discount = _discount;
  }
}
```
The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a chance that a user may successfully execute the operation. An operation is **strongly executable** when it is *always* successfully executed.

```small
context SpecialMenu inv validDiscount: self.discount >= 10
context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances()->size()<=3
```

```small
activity classifyAsSpecialMenu (in _discount:Real) {
    if ( _discount>=10 ) {
        classify self to SpecialMenu;
        self.discount = _discount;
    }
}
```
The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a chance that a user may successfully execute the operation. An operation is **strongly executable** when it is *always* successfully executed.

```java
activity classifyAsSpecialmenu (in _discount)
  if ( _discount>=10 ) {
    classify self to SpecialMenu;
    self.discount = _discount;
  }
}
```

**Feedback**: You must ensure there are less than 3 special menus in the system.
The **executability** of an operation is its ability to be executed without breaking the integrity constraints defined in the structural model.

Two levels of correctness:

An operation is **weakly executable** when there is a chance that a user may successfully execute the operation.  

An operation is **strongly executable** when it is always successfully executed.

---

**Property #2. Executability**

```plaintext
context SpecialMenu inv validDiscount: self.discount >= 10
context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances()->size()<=3

activity classifyAsSpecialmenu (in _discount:Real) {
  if ( _discount>=10 and SpecialMenu.allInstances()->size()<3 ) {
    classify self to SpecialMenu;
    self.discount = _discount;
  }
}
```
Property #3. Completeness
A set of operations is **complete** if all possible changes (inserts/updates/deletes...) on all parts of the system state can be performed through the execution of those operations.
A set of operations is **complete** if all possible changes (inserts/updates/deletes...) on all parts of the system state can be performed through the execution of those operations.

**Property #3. Completeness**

```java
activity addCourse (in _description: String, in _category: CourseCategory) {
    Course c = new Course();
    c.description = _description;
    c.category = _category;
}

activity deleteMenu() {
    self.destroy();
}
```
Property #3. **Completeness**

A set of operations is **complete** if all possible changes (inserts/uploads/deletes...) on all parts of the system state can be performed through the execution of those operations.

```java
activity addCourse (in _description: String, in _category: CourseCategory) {
  Course c = new Course();
  c.description = _description;
  c.category = _category;
}

activity deleteMenu() {
  self.destroy();
}
```

**Feedback:** Actions to destroy courses or to create menus are not specified.
A set of operations is **complete** if all possible changes (inserts/updates/deletes...) on all parts of the system state can be performed through the execution of those operations.

Property #3. **Completeness**

```java
activity addCourse (in _description: String, in _category: CourseCategory) {
    Course c = new Course();
    c.description = _description;
    c.category = _category;
}
```

```java
activity addMenu () {
    Menu m = new Menu();
    ...
}
```

```java
activity deleteMenu() {
    self.destroy();
}
```

```java
activity deleteCourse() {
    self.destroy();
}
```

```java
Menu
- name: String
- price: Real

SpecialMenu
- discount: Real

Course
- description: String
- category: CourseCategory

<<enumeration>>
- CourseCategory
- Starter
- MainCourse
- Dessert

* IsComposedOf 3..*
Summary

Lightweight static methods to verify several correctness properties

- Non-redundancy
- Executability
- Completeness
Summary

UML Executable Models
M2M transformations (ATL, GTR)

Lightweight static methods to verify several correctness properties

- Non-redundancy
- Executability
- Completeness

EXECUTABLE MODEL

FEEDBACK
Summary

UML Executable Models
M2M transformations (ATL, GTR)

Lightweight static methods to verify several correctness properties

Non-redundancy
Executability
Completeness

EXECUTABLE MODEL

FEEDBACK

The feedback (which is expressed in the same language the model) points out how the error may be resolved.
Summary

**UML Executable Models**

M2M transformations *(ATL, GTR)*

**EXECUTABLE MODEL**

**LIMITATIONS**: partial models, partial analysis...

**FEEDBACK**

The feedback (which is expressed in the same language the model) points out how the error may be resolved.

**Lightweight static methods to verify**

several correctness properties

- Non-redundancy
- Executability
- Completeness
Summary

**UML Executable Models**

M2M transformations *(ATL, GTR)*

---

**EXECUTABLE MODEL**

---

**FEEDBACK**

The feedback (which is expressed in the same language the model) points out how the error may be resolved.

---

**Lightweight static methods to verify several correctness properties**

- Non-redundancy
- Executability
- Completeness

---

**Limitations**: partial models, partial analysis...

---

...more work has to be done!
Why this presentation?

1. To present my work

2. To identify further collaborations
Method
- Non-redundancy
- Executability
- Consistency
- Safety
- Deadlock-free
- Livelock-free

Domain
- DSLs
- Other behavioral specifications

Property
- Consistency
- Safety
- Deadlock-free
- Livelock-free

Method
- UML executable models
- ATL M2M Transformations
- GT Rules

Completeness
Method

Lightweight static methods

Model Checking
Event-B
Constraint Programming

UML executable models
ATL M2M Transformations
GT Rules

Non-redundancy
Executability
Completeness

Consistency
Safety
Deadlock-free
Livlock-free

Domain

DSLs
Other behavioral specifications

Property

Consistency
Safety
Deadlock-free
Livlock-free

Method
Method

Model Checking

Constraint Programming

Consistency

Safety

Deadlock-free

Livelock-free

DSLs

Other behavioral specifications

UML executable models

ATL M2M Transformations

GT Rules

Non-redundancy

Executability

Completeness

Lightweight static methods

Model Checking

Event-B

Constraint Programming
Thanks for your attention!

Elena Planas