Overview

1. Executable Model: What & Why?
2. Executable Modeling Languages
3. Example: PetriNet
4. Model Validation & Verification (V&V)
5. V&V Tools
6. Demos
Software Development Lifecycle

Analysis → Design → Implementation

Model Driven Development (MDD)

Model → Code → Result

Workshop on Model-Based Design, Test, and Verification of Software
Software Development Lifecycle

Model Driven Development (MDD)

Executable Model

Code

Boehm’s curve

errors should be removed ASAP
What is Executable Model?

An **executable model** is a model with a behavioral specification sufficiently **detailed** so it can be systematically implemented/executed in the production environment.
Traditional Modeling is Extra Work

- It is hard to validate the correctness of the models before development.
- The developers may not follow the models, without providing feedback.
- It is hard to keep the models and development artifacts in sync during development (and maintenance).
Executable Modeling Just Works

Developers create fully executable models

Using a standards conforming UML modeling tool

Developers iteratively execute, test and update the models

The models are deployed in a production environment

Using a standards conforming UML execution tool

The models are the source code
System engineers analyze, simulate and validate the system design, and allocate requirements to components. Using a standard conforming SysML modeling tool, system engineers create the models. Models can include both hardware and software components. Execution artifacts could include: System Behavior, Statistics. Agile Development…. with executable models.
Executable Model Ecosystem

Model Development Environment

Execution/Simulation Engine

Other Tools

Enabled by standardized executable modeling.

Target Platform
## What to Model?

### Aspects to be modeled:
- Static (Structural) Modeling
- Dynamic (Behavioral) Modeling
- Actions (Behavior in detail)

<table>
<thead>
<tr>
<th>Concept</th>
<th>Called</th>
<th>Modeled As</th>
<th>Expressed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>The world is full of things</td>
<td>data</td>
<td>Classes, Attributes, Associations, Constraints</td>
<td>UML class diagram</td>
</tr>
<tr>
<td>Things have lifecycles</td>
<td>control</td>
<td>States, Events, Transitions Procedures</td>
<td>UML state chart diagram</td>
</tr>
<tr>
<td>Things do as each stage</td>
<td>algorithm</td>
<td>actions</td>
<td>Action language</td>
</tr>
<tr>
<td>Concepts in UML executable models</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UML Executable Model

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

```
context SpecialMenu inv validDiscount: self.discount >= 10
context SpecialMenu inv atMost3SpecialMenus: SpecialMenu.allInstances().size() <= 3
context Period inv validDates: self.initDate < self.endDate
```
UML Executable Model

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(menu=>m,course=>_courses[i]);
    }
  }
}
```
UML Executable Model

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(menu=>m, course=>_courses[i]);
        }
    }
}
```
UML Executable Model

UML executable model = Structural Model + Detailed Behavioral 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

action-based operations

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(menu=>m,course=>_courses[i]);
    }
  }
}

Workshop on Model-Based Design, Test, and Verification of Software
UML Executable Model

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

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

UML executable models may be specified using several modeling languages.
Executable Modeling Languages

Executable modeling are used to define and execute models.

Examples

Petri Nets

Finite State Automata

UML Activity Diagrams

UML State Machines
Executable Modeling Languages: Components

Components of a modeling language

**Syntax: Concepts provided for defining models**

- Formalism: Metamodeling languages
  - OMG MOF standard
  - EMF Ecore

**Semantics: Meaning of the provided concepts**
Executable Modeling Languages: Design & Implementation

- Concrete Syntax (CS)
- Abstract Syntax (AS)
- Semantics Domain (SD)

Compilation or interpretation to get a full-fledged editor

Compilation or interpretation to get a full-fledged execution environment (debugger, animator, simulator, runtime monitor...)

Workshop on Model-Based Design, Test, and Verification of Software
Executable Modeling Languages: Concrete Syntax

Example: Petri net
Executable Modeling Languages: Abstract Syntax
Executable Modeling Languages: Semantics

```c
int x;
void decr () {
    if ( x > 0 )
        x = x - 1;
}
```

- **Axiomatic**
  ```
  context System::decr() post :
      self.x = if ( self.x@pre > 0 )
                  then self.x@pre - 1
                  else self.x@pre
      endif
  ```

- **Denotational/Translational**
  ```
  x
  ```

- **Operational**
  ```
  operation decr () is do
      if x > 0 then x = x - 1 end
  ```
Operational Semantics

- Defines an interpreter for a modeling language

Components of operational semantics

- **Runtime concepts**: Define the state of an executable model during execution
  - Formalism: Metamodelling languages

- **Steps of computation**: Define the transitions from one state of an executable model to the next state
Operational Semantics: Runtime Concepts

Example: Petri nets

Syntax

Metamodel

Net

Transition

name : String

Transition

places

* transitions

input

* output

Place

name : String

InitialTokens: Int

number of tokens of Place objects changes during execution
Example: Petri nets Execution

Initial marking

Enabled transition

Example: Petri nets Execution
Example: *Petri nets Execution*
Example: Petri nets Execution
Executable Models

- Raise the level of abstraction → productivity
- Platform independent models → early verification
- Allow early verification → costs

Better software at less cost

Timeline:
- 1992: "In the future, most developments will be based on an executable specification" [Harel]
- 1997: UML version 1.0
- 2001: xUML: software development methodology designed to precisely define the semantics of executable models
- 2006: UML version 2.0
- 2013: OMG standards for executable modelling
- 2010: ALF Concrete Syntax for UML Action Language [ALF version 1.0 - Beta 1]
- 2011: Foundational Subset for Executable UML Models [UML version 1.0 - Beta]
This is not a new idea...

Shlaer-Mellor
1988, 1991

Became Executable / Translatable UML (xtUML)

Harel State Charts
1988, 1998

Incorporated into UML as State Machines

Real-Time OO Modeling (ROOM)
1994

Became UML for Real Time (UML/RT)
But it wasn’t standardized before...

- **UML 1.1 (first OMG version)**: 1997
- **Action Semantics for the UML RFP**: 1998
- **UML 1.5 (with action semantics)**: 2003
- **Semantics of a Foundational Subset for Executable UML Models RFP**: 2005
- **UML 2.0**: 2005
- **UML 2.3**: 2010
- **Foundational UML (fUML) 1.0**: 2011
- **UML 2.4.1**: 2011
- **UML 2.5 (current version)**: 2015
- **fUML 1.2.1 (current version)**: 2016
- **fUML 1.3**: 2017
- **fUML 1.4**: 2018
Foundational UML (fuml)

- is an OMG standard
- is an executable sub-language of UML and can be used for defining operational semantics
- Uses operational approach
- Specification of behavioral semantics of foundational UML subset
  - **Class modeling**: class, association, data type, etc.
  - **Activity modeling**: activities, parameters, nodes, flows, etc.
  - **Action language**: communication, object, structural feature, link actions
- **Virtual machine** implemented with **Java** for executing models
Model Execution: Looking ahead…

Model Edition & Consistency
Model/System Integration
Allocation & Deployment Model
Model Execution & Reconfiguration

Model Composition

Design-Space Exploration

Data analysis, Model Discovery, Execution trace analysis

Workshop on Model-Based Design, Test, and Verification of Software
Summary

- **Executable models** provide abstractions and enable early analyses of the behavior of a system.

- **Executable modeling languages** need to have well defined semantics.

- **Operational semantics** define **runtime concepts** (state information) and **steps of computation** (state transitions) for interpreting models.

- **Foundational UML** (fUML) is an executable sub-language of UML and can be used for defining operational semantics.
Dynamic Model Validation & Verification
Software Development Lifecycle

Executable Model

Model Driven Development (MDD)

how can the quality of executable models be improved?
Software Development Lifecycle

Model Driven Development (MDD)

Executable Model

Code

need for methods and tools to evaluate the correctness of executable models

Workshop on Model-Based Design, Test, and Verification of Software
Dynamic Verification & Validation

How can the quality of executable models be improved?

Dynamic Verification and Validation (V&V) techniques are used to ensure the correctness of the modeled behavior.
Dynamic V&V: An example

- **Dynamic analysis** uses execution traces to analyze the properties of a running system and to represent the system's actual behavior.

- **Execution traces** are generated during the execution of a model, and provide information to help reason about the model's execution behavior.
Model Testing

- Validate whether a system meets functional requirements

- **Model-based testing**: Testing a system’s implementation based on models
- **Testing models**: Test a system’s design defined by an executable model
Model Debugging

- Locate defects in a system, e.g., model animation
  - **Model-based debugging**: debugging a running application with the help of a model
  - **Debugging models**: model modification at runtime, back stepping, step-wise execution, ...
Stepwise Debugging (interactive debugging)

- find the cause of a defect by manually observing and controlling execution
Omniscient Debugging

- Stepwise debuggers only go forward

- Omniscient debuggers go forward and backward

- Omniscient debuggers rely on an execution trace storing previous steps
Model Debugging & Model Animation
Model Debugging & Model Animation
Model Debugging & Model Animation
Semantic Model Differencing

Models are subject to change → Change management for models is required

Model differencing

**Goal**: Identify differences among models

**Applications**: Merging, versioning, conflict detection, incremental testing, etc.
Semantic Model Differencing

Syntactic Differencing

Epsilon Comparison Language (ECL)
Dynamic Analysis

- Analyze properties of a running program (model), e.g.,
  - Trace exploration for executable models
  - Trace visualization based on models

- Non-functional property analysis: Analyze emergent properties of a system, e.g.,
  - Performance analysis
Tools for Model Execution, Verification & Validation
Main Current Initiatives

  - xtUML https://xtuml.org/xtuml-days-2018copenhagen
  - Blog post series by Leon Starr on modeling-languages.org: https://modeling-languages.com/author/leon-starr
- BridgePoint, cf. https://xtuml.org
- UML-RT
  - Papyrus-RT: https://www.eclipse.org/papyrus-rts
The Gemoc Studio

Designate and Integrate your Executable DSL

Edit simulate and Animate your heterogeneous models
The Gemoc Studio

Metamodeling Languages

Executable Modeling Languages

Executable Models
The GEMOC Community
Conclusion

- While modeling techniques are widely adopted in various guises (incl. behavioral modeling), model execution is still under used.

- Rethink our model execution tools to support:
  - Immediate feedback (from the runtime state)
  - Direct manipulation (of the runtime state)
  - Reasoning over the time (on the runtime state)

Live modeling and time traveling should be at the core of model execution.
Tools Introduction & Demos
UMPLE

is a technology for model-oriented programming. It merges the concepts of programming and modeling by adding **modeling abstractions directly into programming languages**

https://cruise.eecs.uottawa.ca/umple/
https://cruise.eecs.uottawa.ca/umpleonline/
```java
class Participant {
    String name;
    Integer participantID;
    // -- * Activity registration;
}

class Activity {
    String description;
    Date activityDate;
    // -- Add More --
}
```

```java
/*PLEASE DO NOT EDIT THIS CODE*/
/*This code was generated using the UML 1.10.3.3177 modeling language!*/

import java.util.*;

public class Participant {

    // -- MEMBER VARIABLES
```
Papyrus Industry Consortium

Everybody focused on a common objective
Development of a complete MBE solution
MOKA: Model Execution

is a Papyrus module for execution of UML models, which natively includes an execution engine complying with OMG standards
Fuml Example: A Counter

Running the model using the MOKA engine ➔ see 2min video

Moka is a Papyrus module for execution of UML models

http://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution#Moka_Overview
Cameo Simulation Toolkit

The Cameo Simulation Toolkit provides the first in the industry extendable model execution framework based on OMG fUML and W3C SCXML standards. It extends UPDM plugin to validate system behavior by executing, animating, and debugging SysML Parametric models in the context of realistic mock-ups of the intended user interface.

Cameo Simulation Toolkit: Features

- Activity execution (OMG™ fUML standard)
- State machine execution (W3C SCXML standard)
- SysML parametric execution (OMG SysML standard)
- Multiple action languages support (JSR223 standard)
- Exporting UML state machine to SCXML file format
- Full featured model debugger
- Execution animation
- Quick UI prototyping
- Pluggable engines and evaluators
- Model-driven test cases and scenarios
Feature Presentation
References


THANK YOU!
ANY QUESTIONS?