Model Execution: From a Retrospective on Code Generation to a Perspective on Model Compilation

Federico Ciccozzi
federico.ciccozzi@mdu.se
Models
Models
Models

• Models can be found in any scientific discipline
• We need to be precise and specific on WHAT a model is
Models

• Models can be found in any scientific discipline
• We need to be precise and specific on **WHAT a model is**

“A model is an abstract representation of a specific part, problem, solution, or feature of a specific domain”
Software models (some definitions)

“Software models are ways of expressing a software design”

“Software models are representations of software systems made to understand, analyze, and design such systems”

“A software model is a collection of representations whose contents depend on the languages and tools used”

“Software models are formal methods for handling the process of creating software”
Software models in our context

A (software) model

• is a *blue-print* of a software application,
Software models in our context

A (software) model
• is a *blue-print* of a software application,
• can be itself *executable*, and
Software models in our context

A (software) model
• is a blue-print of a software application,
• can be itself executable, and
• is directly usable for automating the development process
Model execution

• Process of partly or fully running a computational model in a software environment
Model execution

• Process of partly or fully running a computational model in a software environment
• Simulation for prediction/monitoring based on specific inputs and configurations
Model execution

• Process of partly or fully running a computational model in a software environment
• Simulation for prediction/monitoring based on specific inputs and configurations
• Pivotal to understand complex systems
Model execution

- Process of partly or fully running a computational model in a software environment
- Simulation for prediction/monitoring based on specific inputs and configurations
- Pivotal to understand complex systems
- Pivotal to forecast outcomes related to criticality aspects (e.g., time, safety, security)
Model execution

• Process of partly or fully running a computational model in a software environment
• Simulation for prediction/monitoring based on specific inputs and configurations
• Pivotal to understand complex systems
• Pivotal to forecast outcomes related to criticality aspects (e.g., time, safety, security)
• Essential in domains like data science, AI, machine learning
Model execution strategies

- Model
- Model translator
- 3GL code
- 3GL compiler
- Low-level code
- Execution

Translational
Model execution strategies

Translation
model → model translator → 3GL compiler → low-level code → execution
Model execution strategies

Translational

model → model translator → 3GL code

3GL interpreter → execution
Model execution strategies

**Translational**

- model
- model translator
- Java
- 3GL interpreter
- execution
Model execution strategies

Translation
- model
- model translator
- 3GL code
- 3GL compiler
- 3GL interpreter
- low-level code
- execution
Model execution strategies

Interpretive

- model
- model interpreter

Translational

- model
- model translator
- 3GL code
- 3GL compiler
- low-level code
- 3GL interpreter

execution
UML execution

• Systematic review of research and practice
  • Research articles
  • Tools
• Investigated >5400 items
• Included 63 articles and 19 tools
• Systematic search and data extraction

Classification of UML execution solutions

UML model execution

- Intended benefits
- Associated process
- Extensibility
- Readiness level
- Supported non-functional properties
- Formal specification languages
- Execution strategy

- UML modeling
- Required UML diagrams
- Used action languages
- Required UML profiles
- Modeling tool
- MDA levels covered
- Based on the fUML standard
- Support for partial models

- UML model execution
- 1..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
- 0..*
About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)

About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)
    • 50% use Java as model transformation/translation language

About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)
    • 50% use Java as model transformation/translation language
    • 17% based on interpretation, only for simulation/analysis purposes
About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)
    • 50% use Java as model transformation/translation language
    • 17% based on interpretation, only for simulation/analysis purposes

• Execution semantics
  • 15% based on fUML

About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)
    • 50% use Java as model transformation/translation language
    • 17% based on interpretation, only for simulation/analysis purposes

• Execution semantics
  • 15% based on fUML

• Action languages
  • >90% use 3GLs
  • Only 10% based on Alf
About executability..

• Execution strategy
  • 85% based on translation to 3GLs (mostly Java and C++)
    • 50% use Java as model transformation/translation language
    • 17% based on interpretation, only for simulation/analysis purposes
• Execution semantics
  • 15% based on fUML
• Action languages
  • >90% use 3GLs
  • Only 10% based on Alf
• Support for simulation
  • 59% provide some simulation feature

Other interesting aspects..

- Extensibility
  - 21% provide some sort of extension mechanism
Other interesting aspects..

- **Extensibility**
  - 21% provide some sort of extension mechanism

- **Traceability (model-code)**
  - 18% provide some support for trace links
Other interesting aspects.

• Extensibility
  • 21% provide some sort of extension mechanism

• Traceability (model-code)
  • 18% provide some support for trace links

• Interactive debuggability
  • 25% provide debugging features at model-level
In summary

• *Translation* outnumbers interpretation
• Interpretation is used for higher-level execution (e.g., simulation)
• Execution semantics from *fUML is neglected* in most cases
• Most solutions employ *3GLs as action languages*
• Almost no model-level interactive debugging
• Little extensibility and customizability
What about "code generation" approach

- Convenient, reuse of existing (trusted) 3GL compilers

What about "code generation" approach

• Convenient, reuse of existing (trusted) 3GL compilers
• Creates discontinuity between model and executable
  • Model debugging can become very difficult
  • Co-debugging and co-simulation nearly impossible

What about ”code generation” approach

• Convenient, reuse of existing (trusted) 3GL compilers
• Creates discontinuity between model and executable
  • Model debugging can become very difficult
  • Co-debugging and co-simulation nearly impossible
• Lack of trust from developers
  • Generated 3GL “inspected” and modified by hand

What about ”code generation” approach

• Convenient, reuse of existing (trusted) 3GL compilers
• Creates discontinuity between model and executable
  • Model debugging can become very difficult
  • Co-debugging and co-simulation nearly impossible
• Lack of trust from developers
  • Generated 3GL “inspected” and modified by hand

What about ”code generation” approach

• Convenient, reuse of existing (trusted) 3GL compilers

• Creates discontinuity between model and executable
  • Model debugging can become very difficult
  • Co-debugging and co-simulation nearly impossible

• Lack of trust from developers
  • Generated 3GL “inspected” and modified by hand
    • Violate source models
    • Violate model-based analysis, optimisation and V&V
    • Get lost if source models change

What about "code generation" approach\textsuperscript{1}

- Convenient, reuse of existing (trusted) 3GL compilers
- Creates discontinuity between model and executable
  - Model debugging can become very difficult
  - Co-debugging and co-simulation nearly impossible
- Lack of trust from developers
  - Generated 3GL “inspected” and \textit{modified} by hand
- 3GL compilers do not understand model semantics
  - Program optimisations may be missed
  - Generated executables may be semantically different from models

What about "code generation" approach

• Convenient, reuse of existing (trusted) 3GL compilers
• Creates discontinuity between model and executable
  • Model debugging can become very difficult
  • Co-debugging and co-simulation nearly impossible
• Lack of trust from developers
  • Generated 3GL “inspected” and modified by hand
• 3GL compilers do not understand model semantics
  • Program optimisations may be missed
  • Generated executables may be semantically different from models
• Not suitable for heterogeneous platforms (multiple 3GLs needed)

Back to code generation for UML

• 26 different code generators from UML to Java
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models

```text
let a : A = null;
f(a); execution stops
k();
```
Back to code generation for UML

• 26 different code generators from UML to Java

• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models

fUML model in textual format

```plaintext
let a : A = null;
f(a); execution stops
k();
```

Java from code generator 1

```java
A a = null;
f(a); unchecked exception
k();
```
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

- 26 different code generators from UML to Java
- No reference semantics (e.g. fUML)
  - Different code generators (even commercial!) produce different codes from same models
Back to code generation for UML

• 26 different code generators from UML to Java
• No reference semantics (e.g. fUML)
  • Different code generators (even commercial!) produce different codes from same models

Some of the issues
• Predictability
• Validity of MB-analysis
• Consistency model-code
• Bidirectional traceability
• Co-simulation
• Co-debugging
Model execution strategies

**Interpretive**
- Model
- Model interpreter
- Execution

**Translational**
- Model
- Model translator
- 3GL code
- 3GL compiler
- 3GL interpreter
- Low-level code
- Execution
Why are we generating 3GL code?!
Why are we generating 3GL code?!
Why are we generating 3GL code?!
We have tried before..
.. And not succeeded
Code generation.. why
We do not have a fear of the unknown. What we fear is giving up the known.

Anthony de Mello
Model execution strategies
Model execution strategies

In H2020 SPACE-10-TEC-2020, new model execution approaches regarded as the most urgent software engineering need in the space industry.
Model compilation

- Model
- Model compiler
- 3GL compiler
- 3GL interpreter
- Model interpreter
- Compilation to low-level code
- Low-level code
- Execution
Model compilation

- 3GL code generators are specific to source DSML and target 3GL
Model compilation

- 3GL code generators are specific to source DSML and target 3GL
Model compilation

• 3GL code generators are specific to source DSML and target 3GL
Model compilation

• Bypass 3GLs and compile models by a flexible compiler framework
Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
Model compilation

• Bypass 3GLs and compile models by a flexible compiler framework
• Semantic anchoring of DSML at hand to compiler IR language
• Preservation of model semantics in generated executables
Model compilation

• Bypass 3GLs and compile models by a flexible compiler framework
• Semantic anchoring of DSML at hand to compiler IR language
• Preservation of model semantics in generated executables
• Coherent model semantics-based analysis and optimisations
Model compilation

- Bypass 3GLs and compile models by a flexible compiler framework
- Semantic anchoring of DSML at hand to compiler IR language
- Preservation of model semantics in generated executables
- Coherent model semantics-based analysis and optimisations
- Reusability of compiler “lowerings” for same front-ends and back-ends
Model compilation

• Bypass 3GLs and compile models by a flexible compiler framework
• Semantic anchoring of DSML at hand to compiler IR language
• Preservation of model semantics in generated executables
• Coherent model semantics-based analysis and optimisations
• Reusability of compiler “lowerings” for same front-ends and back-ends
• Use of AI/ML for compilation purposes (e.g. semantic anchoring)
Model compilation

• Bypass 3GLs and compile models by a flexible compiler framework
• Semantic anchoring of DSML at hand to compiler IR language
• Preservation of model semantics in generated executables
• Coherent model semantics-based analysis and optimisations
• Reusability of compiler “lowerings” for same front-ends and back-ends
• Use of AI/ML for compilation purposes (e.g. semantic anchoring)
• Use of model compilation for AI/ML purposes
Envisioned model compilation approach

- $M_X$: Model in language X
- $DSML_X$: DSML X
- Automated macro-step
- Compiler components
- Compiler theory
- Conforms to
- I/O flow
- Implemented in
Envisioned model compilation approach

- **M**: Model in language X
- **DSML**: DSML X
- **Automated macro-step**
- **Compiler components**
- **Compiler theory**
- **Conforms to**
  - I/O flow
- **Implemented in**

**Compiler framework**

**Compiler**

**Compiler**

**IRL**
Envisioned model compilation approach

\[ M_X : \text{Model in language } X \]
\[ \text{DSML}_X : \text{DSML } X \]

Automated macro-step

Compiler components

Compiler theory

\[ \rightarrow \text{Conforms to} \]

I/O flow

\[ \rightarrow \text{Implemented in} \]

Compiler framework

Compiler_{IRL}
Envisioned model compilation approach

- $M_X$: Model in language X
- $DSML_X$: DSML X
- Automated macro-step
- Compiler components
  - Compiler theory
  - Conforms to
  - I/O flow
  - Implemented in

- $DSML_1$ to $DSML_m$
- HOT
- Compiler IR
- Compiler framework
Envisioned model compilation approach

- $M_X$: Model in language $X$
- $\text{DSML}_X$: DSML $X$
- Automated macro-step
- Compiler components
- Compiler theory
- Conforms to
- I/O flow
- Implemented in

$\text{DSML}_1 \quad \ldots \quad \text{DSML}_m$

$\text{HOT}$

$\text{Compiler}_{\text{IRL}}$

Compiler framework
Envisioned model compilation approach

- $M_X$: Model in language $X$
- $DSML_X$: DSML $X$
- Compiler components
  - Compiler theory
  - Conforms to I/O flow
  - Implemented in Compiler framework

Compiler components:

- Automated macro-step
- Implemented in

Diagram:

- DSML$_1$ → HOT → $T_{DSML_1}$
- $\ldots$
- DSML$_m$ → HOT → $T_{DSML_m}$
- HOT → Compiler$_{IRL}$
Envisioned model compilation approach

\[ M_X: \text{Model in language } X \]
\[ \text{DSML}_X: \text{DSML } X \]

- Automated macro-step
- Compiler components
  - Compiler theory
- Conforms to I/O flow
- Implemented in

\[ \text{DSML}_1 \]
\[ \text{DSML}_m \]

\[ \text{HOT} \]

\[ \text{Compiler}_{\text{IRL}} \]

\[ T_{\text{DSML}_1} \]
\[ T_{\text{DSML}_m} \]

- Compiler framework
Envisioned model compilation approach

- $M_X$: Model in language $X$
- $\text{DSML}_X$: DSML $X$

- Automated macro-step
- Compiler components
- Compiler theory
- Conforms to
- I/O flow
- Implemented in

- Hot

- Compiler framework

- $T_{\text{DSML}_1}$
- $T_{\text{DSML}_m}$

- $\text{DSML}_1$
- $\text{DSML}_m$

- $M_{\text{DSML}_1}$
- $M_{\text{DSML}_m}$
Envisioned model compilation approach

M_x: Model in language X
DSML_x: DSML X
Automated macro-step
Compiler components
Compiler theory
Conforms to
I/O flow
Implemented in

Compiler framework

HOT

T_DSML_1
T_DSML_m

DSML_1

... DSML_m

M_DSML_1

T_DSML_1

M_DSML_m

T_DSML_m

PROG_IRL

IR
Envisioned model compilation approach

- $M_X$: Model in language X
- DSML$_X$: DSML X
- Automated macro-step
- Compiler components
- Compiler theory
- Conforms to
- I/O flow
- Implemented in

Compiler framework

DSML$_1$ --> $M_{DSML}_1$ --> $T_{DSML}_1$ --> PROG$_{IRL}$

DSML$_m$ --> $M_{DSML}_m$ --> $T_{DSML}_m$ --> PROG$_{IRL}$

Compilation steps
Envisioned model compilation approach

- $M_X$: Model in language $X$
- $\text{DSML}_X$: DSML $X$
- Automated macro-step
- Compiler components
  - Compiler theory
- Conforms to
- I/O flow
- Implemented in

Compilation steps

Compiler framework

$\text{PROG}_{\text{IRL}}$

Compilation steps

$\text{HOT}$

$T_{\text{DSML}_1}$ $\ldots$ $T_{\text{DSML}_m}$
ORPHEUS model compilation approach

DSML\_1 \rightarrow \ldots \rightarrow DSML\_m \rightarrow ORPHEUS\_HOT

M\_X: Model in language X
DSML\_X: DSML X

Compiler components

Compiler theory

\longrightarrow Conforms to

\longrightarrow I/O flow

\longrightarrow Implemented in

\longrightarrow Implemented in

DSML\_1 \rightarrow M\_DSML\_1 \rightarrow T\_DSML\_1 \rightarrow \ldots \rightarrow M\_DSML\_m \rightarrow T\_DSML\_m \rightarrow PROG\_IRL

\longrightarrow MLIR lowering steps

https://www.esmdu.se/projects/603-ORPHEUS, funded by the Swedish Research Council (VR)
A lot to do..

• Ability to execute abstract (high-level) and incomplete models
A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution

• Compilation of DSMLs
• Integration of model simulation into heterogeneous multi-paradigm simulation systems
• UML model execution
• Compliance to fUML execution semantics
• Support for UML-compliant action languages
• Support for executing models based on UML profiles
A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution
- Compilation of DSMLs
A lot to do..

• Ability to execute abstract (high-level) and incomplete models
• Observability of executing models
• Control of model execution
• Compilation of DSMLs
• Integration of model simulation into heterogeneous multi-paradigm simulation systems
A lot to do..

- Ability to execute abstract (high-level) and incomplete models
- Observability of executing models
- Control of model execution
- Compilation of DSMLs
- Integration of model simulation into heterogeneous multi-paradigm simulation systems
- UML model execution
  - Compliance to fUML execution semantics
  - Support for UML-compliant action languages
  - Support for executing models based on UML profiles
.. and..
.. and..

Many of us are engineers and..
we tend to bring all back to “programs” and “programming”,
which is what we know (and often value) the most
.. and..

Many of us are engineers and..
we tend to bring all back to “programs” and “programming”,
which is what we know (and often value) the most

Let’s not forget that modelling, for sketching, communication,
and brainstorming purposes, is..

.. fun, useful, and very valuable!
Design adds value faster than it adds cost.

Joel Spolsky
Thank you!