Using Domain Specific Languages for Modeling and Simulation: ScalaTion as a Case Study

John A. Miller

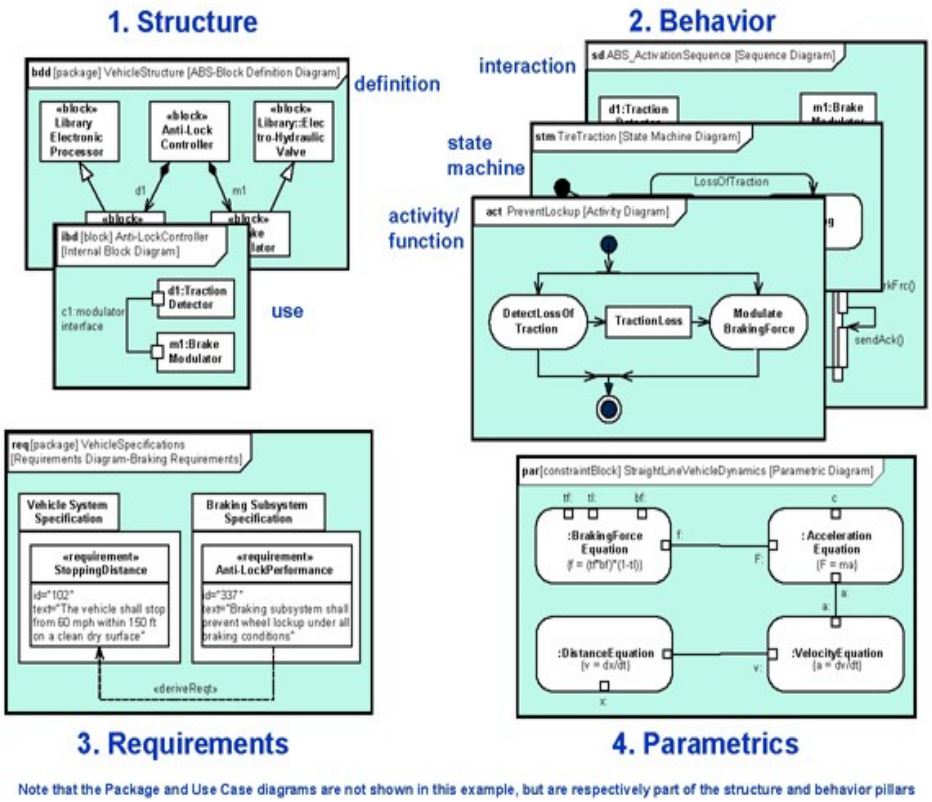
Jun Han

Maria Hybinette

Department of Computer Science
The University of Georgia

Conceptual Model vs. Simulation Program

What is the difference?

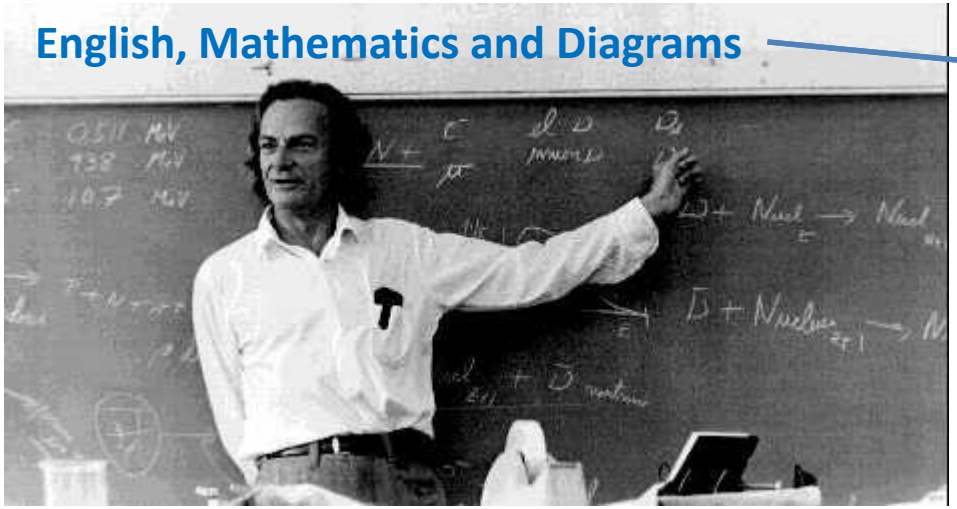


Conceptual Model, e.g., SysML
(http://www.omgsysml.org/)

```
OKOK = .FALSE.
      NRUN = IQ(LHEAD+6)
      NEV = IQ(LHEAD+9)
C
      IF (IB.NE.2) THEN
        CALL ERRMSG('L2_COMPARE','L2_COMP_LETA',
     &   'L2_COMP_LETA called for IB.NE.2 ! Not allowed!!!','F')
        RETURN
      ENDIF
C
C     make sure there is LETA - no mistakes
C
      IF (LLETA(1).LE.0) THEN
        WRITE(LUN,'(" L2_COMP_LETA: Run/Event ",2I7,
     &      " has NO LETA bank for SIMULATION")') NRUN,NEV
        GOTO 999
      ENDIF
      IF (LLETA(2).LE.0) THEN
        WRITE(LUN,'(" L2_COMP_LETA: Run/Event ",2I7,
     &      " has NO LETA bank for DATA")') NRUN,NEV
        GOTO 999
      ENDIF
```

Simulation Program, e.g., Fortran
(http://www-d0.fnal.gov/~hirosky/trigger/l2prod/l2_comp_leta.for)

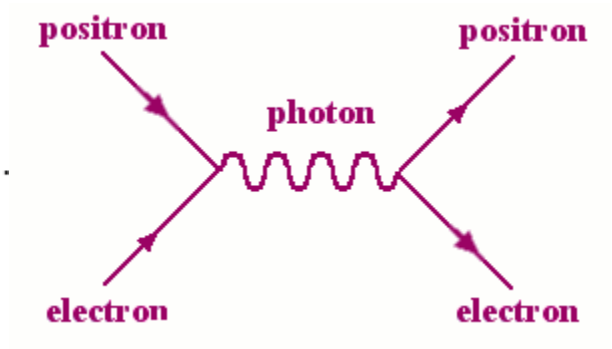
Why the Huge Gulf?



1010101010111000110101 00

"To those who do not know mathematics it is difficult to get across a real feeling as to the beauty, the deepest beauty, of nature"

$$S = \int \partial^\mu \phi \partial_\mu \phi + \frac{\lambda}{4!}\phi^4.$$



http://www.brew-wood.co.uk/physics/feynman3.jpg
http://picasaweb.google.com/lh/photo/Ee3JHToIv9ahUnWVSGxZtg
http://plus.maths.org/content/tying-it-all

Progress from the Right:

Evolution of General-purpose Programming Languages (GPLs)

- First Programming Language
  - **UNIVAC SHORT CODE, 1949**
- First Machine Independent Programming Language
  - **FORTRAN, 1954**
- First Structured Programming Language
  - **ALGOL, 1958**
- First Functional Programming Language
  - **LISP, 1958**
- First Object-oriented Programming Language
  - **SIMULA, 1967**
- First Functional Object-oriented Programming Language
  - **Common Lisp Object System (CLOS), 1988**

Object-oriented Languages with Functional Features

Language	Developer	Manual
OCaml	Remy	Smith 2006
F#	Syme	Syme et al. 2007
Scala	Odersky	Odersky et al. 2008
Python	van Rossum	Watters et al. 1996
Ruby	Matsumoto	Thomas and Hunt 2000
Groovy	Laforge	Koenig et al. 2007
C#	Hejlsberg	Hejlsberg et al. 2003

Progress from the Left:
Enrichment of Conceptual Modeling

- Conceptual Modeling Artifacts
  - **Requirements Document**
  - **Goals and Objectives**
  - **Terminology/Ontology**
  - **Model Design Specification**
- Not meant to be executable, but should facilitate the consistency checking

Progress from the Left (cont.)

- Higher-level of Discourse than Simulation Programs
  - Design Diagrams
    - **Process flow diagrams**
    - **Activity cycle diagrams**
    - **Petri nets**
    - **Event graphs**
    - **UML (Unified Modeling Language)**
  - Component Descriptions
  - Mathematical Models for Elements or Verification
  - Alignment of a Domain Ontology with a Modeling Ontology, e.g., DeMOforge

Progress in the Middle:
Simulation Programming Languages (SPLs)

Language	Developer	Date
GASP	Kiviat	1961
GPSS	Gordon	1961
SIMSCRIPT	Markowitz	1963
SIMULA 67	Nuggard and Dahl	1967
SLAM	Pritsker	1979
SIMAN	Pegden	1985

Later advances: simulation environments, animation and graphical model construction

Creating DSLs for M&S

- Is there a faster way to bridge the gulf?
- **GPL**s and **SPL**s each have their own pros and cons

Why not try using a Domain Specific Language (**DSL**)?

http://www.mageba.ch/user_content/editor/themes/ReferenzenUddevallaBr%FCcke/udevalla_gesamt_577x369.jpg

A Domain Specific Language (DSL)

- Definition:
  - "is a **programming language or executable specification language** that offers, through appropriate notations and abstractions, **expressive power** focused on, and usually restricted to, a **particular problem domain**"
- Key Advantage:
  - "trades **generality for expressiveness** in a limited domain. By providing notations and constructs tailored toward a particular application domain, they offer **substantial gains in expressiveness and ease of use** compared with GPLs for the domain in question, with corresponding **gains in productivity and reduced maintenance costs**"

Domain Specific Language (DSL)

- Types
  - Externally Defined DSLs
    - **Requires pre-processors, parsers and code generators**
    - **This category includes some SPLs**
  - Internally Defined or Embedded DSLs
    - **Definable using the advanced features of the parent language**
    - **Easy to develop such DSLs**
    - **Easy to learn for those familiar with the parent language**

Languages Facilitating DSLs

Languages	Object oriented	Functional	Type checking	Conciseness
Java	Impure	Very little	Static	Low
Python	Lack of encapsulation	Many Features	Dynamic	High
Ruby	Pure	Many Features	Dynamic	High
Scala	Pure	Almost All	Static	High

Static Typing vs. Dynamic Typing?

language	runtime
C GNU gcc	1.08
C++ GNU g++	1.09
Ada 2005 GNAT	1.34
Java 6 -server	1.59
Scala	2.06
Fortran Intel	2.19
Pascal Free Pascal	2.35
Haskell GHC	2.48
C# Mono	2.5
OCaml	3.03

language	runtime
F# Mono	3.22
Lisp SBCL	3.87
Go 6g 8g	4.53
Clojure	10.81
Erlang HiPE	12.86
Ruby JRuby	45.71
Python CPython	46.5
Python 3	49.58
Ruby 1.9	63.78
Perl	64.81

http://shootout.alioth.debian.org/

Language Features for Building DSLs

- Operator Overloading and Infix Notation
- Type Inference
- Type Alias
- First-Class Functions and Closures
- Functional Programming
  - **Immutable variables, iterator methods, higher order functions, currying and partial function applications**
- Default Arguments
- Parser Combinator Library

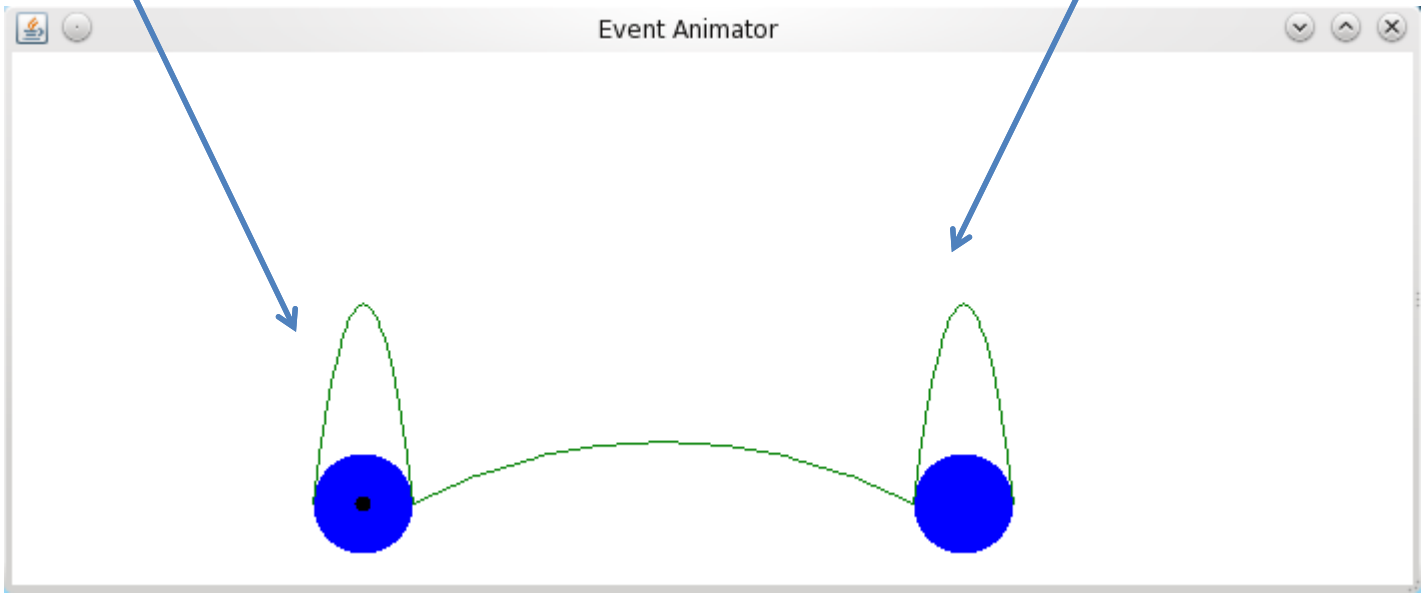
ScalaTion

- Simulation system coded in Scala
  - **Since a design goal of Scala is to facilitate the construction of DSLs**
- Utilizes or recodes some modules of JSIM
  - **Portions of the 50 Kloc JSIM code-base were recoded with approx. 80% reduction in loc**
- Supports the modeling paradigms of the Discrete-event Modeling Ontology (DeMO)
  - **Event, process, activity and state**

scalation.event

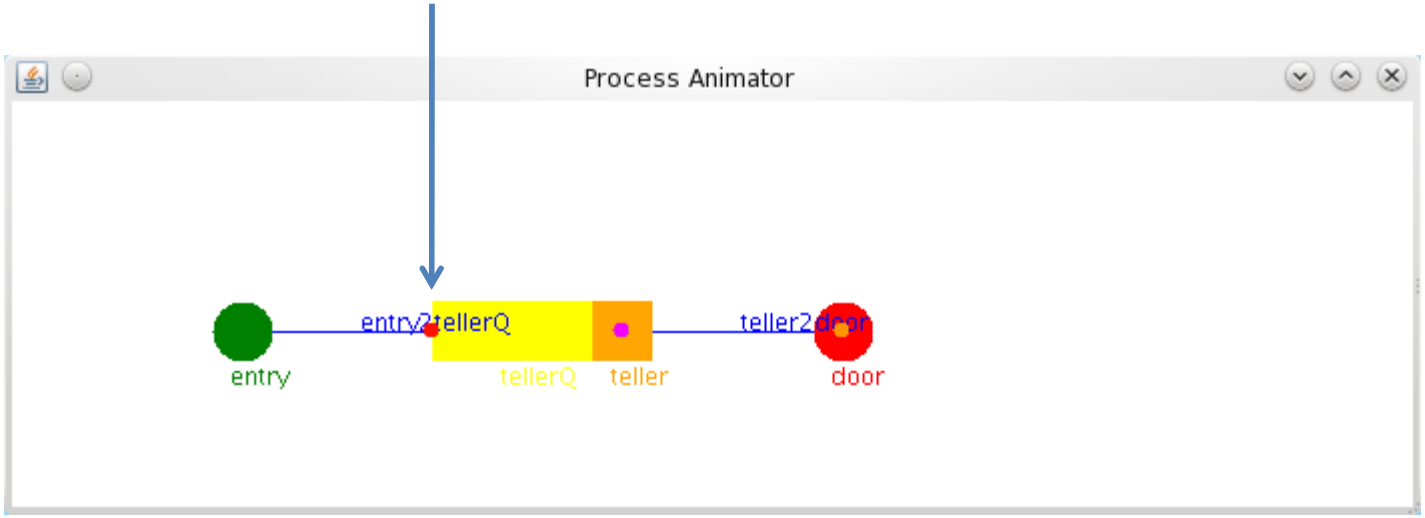
```
case class Arrival (customer: Entity)
  extends Event (protoArr, customer,
aLinks, this, Array (150., 200., 50., 50.)) {
  override def occur () {
    super.occur () // handle casual links
    nArr += 1 // update the current state
    nIn += 1
  } // occur
} // Arrival class
```

```
case class Departure (customer: Entity)
extends Event (protoDep, customer, dLinks,
this, Array (450., 200., 50., 50.)) {
  override def occur () {
    super.occur () // handle casual links
    nIn -= 1 // update the current state
    nOut += 1
  } // occur
} // Departure class
```



scalation.process

```
case class Customer () extends SimActor ("c", this) {
  def act () {
    entry2tellerQ.move ()
    if (teller.busy) tellerQ.waitIn ()
    teller.utilize ()
    teller.release ()
    teller2door.move ()
    door.leave ()
  } // act
} // Customer
```



Conclusions

- Narrowing the gap between model and program
- Using an embedded Domain Specific Language (DSL) rather than a General Purpose Language (GPL) or Simulation Programming Language (SPL)
- ScalaTion prototype looks promising – needs further development and testing
- Other new statically-typed functional object-oriented languages may be suitable as well (e.g., F#)

Future Work

- **scalation.dynamics**: add an integrator more suitable for stiff systems and extend our LinearDiffEq class to handle complex eigenvalues.
- **scalation.optimization**: add simplex method, quadratic programming, steepest descent, conjugate gradient and quasi-newton.
- **scalation.scala3d**: add 3D animation package that interacts with Java OpenGL (JOGL)

Future Work (cont.)

- Adding Unicode Support
  - Use of Greek Symbols

```
case class Normal (μ: Double, σ: Double, ψ: Int)
```

  - Use of Math Symbols

```
def ≤ (y: T): Boolean = x <= y
def · (x: T, y: T): T = x * y
def Σ (v: VectorN [T]): T = v.foldLeft (0) (_ + _)
```