Beyond Mere Logic: A Vision of Computer Languages for the 21st Century
- A discourse on software physics -

Bran Selić
Malina Software Corp. CANADA
Simula Research Laboratory, NORWAY
Zeligsoft Limited (2009), CANADA
University of Sydney, AUSTRALIA

selic@acm.org
Core Thesis of This Talk

- The two solitudes...

Programming Languages Domain

The Physical World
Core Thesis of This Talk

- The two solitudes...must be better integrated
Core Thesis of This Talk

- However...

Programming Languages Domain

The Physical World

Here Be Dragons
The Case of the MARS Climate Orbiter

"The 'root cause' of the loss of the spacecraft was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software..."

-- NASA report, 1999

Q: Why was this not detected automatically by the compiler as a type mismatch?

Because no mainstream programming language has a first-class concept of a "physical" values (or, more generally, the real world)

\[ \text{e.g., force: } \text{Force} = 225; \]
\[ \text{delay(100);} \]

~$650M!
Q: Can't we just define a special library type?

```c
enum LengthUnit {mm, cm, m, km};

type Length {
  real value,
  LengthUnitType unit};
```

No: a compiler would still not catch unit mismatches or know how to compare two or more values of such a type.

In contrast, a first-class language construct has semantics defined by the language that are known and supported by all conforming tools (compilers, validators, interpreters, debuggers, etc.)
But, Why Should We Care?

- Isn’t real-time/embedded software a highly-specialized *niche* domain?
- Really?
An increasing number of software applications interact directly with the physical world.
Sample Applications in This Category

- Control and monitoring systems, communications systems, industrial control systems, automotive systems, etc.
- Financial systems (banking, point of sale terminals, etc.)
- Computer-aided design tools (AutoCAD, CATIA, etc.)
- Simulation software (physics, weather, machinery, etc.)
- Computer games software
- etc.

All of these application types:
1. interact directly with the physical world and/or
2. incorporate a representation of it
Mainstream Programming Languages

None of these languages recognize the concept of time or physical quantity

“Languages of the future for programming techniques of the past” [E. Dijkstra]

http://www.pasteur.fr/formation/infobio/python/ch01s03.html

Source: Tiobe & Jobs
CLAIM: Our current software technologies and design methods are not well suited for the emerging flood of applications that interact with the real world.

Why not?

To understand why things are the way they are, we need to know how they came to be... [Aristotle?]
Where Programming Started...

Original computer applications were devised to mechanize computation of complex algorithms

- Ballistics tables, code breaking, etc.
- ...which is why they are called “computers”

⇒ Strong focus on numerical methods, mathematical logic, and symbol manipulation

A clear algorithmic bias

ENIAC

Colossus
The Response: Software Platonism

- “I see no meaningful difference between programming methodology and mathematical methodology.”
  -- Edsger W. Dijkstra (EWD 1209)

- “Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate.”
  -- Wei-Lung Wang, Comm. of the ACM (45, 5), 2002

This was and still is a highly influential view.
- Functionality ("business logic") comes first
  - Premise: No point in worrying about other concerns (e.g., performance, availability) if that is incorrect

- "Platform-independent" design, "premature optimization", etc.

Unstated assumption:
Non-functional concerns are separable from functionality and, hence, can be easily retrofitted
Those “Other” Concerns

- The “ilities” of software
  - Reliability, scalability, availability, testability, performance/throughput, security, maintainability, stability, controllability, observability, extensibility, interoperability, usability, etc.

Most of these are affected either directly or indirectly by the physical aspects of the system (e.g., platform, communication networks)
Start by Saying “No” to “Non-functional”?

Did someone just say “NON-FUNCTIONAL”!?
What’s Wrong with “Non-functional”?

1. **Negative** specification (does not tell us what they are)
2. Suggests **second-order** concerns (auxiliary, miscellaneous, etc.)
3. Bundles a **very diverse** collection of **often critical** system properties
   - Each of which is achieved by different methods
4. **Separates mutually inter-dependent aspects**
   - Can have a fundamental impact on how functionality is realized
   - NB: They are **non-modular and pervasive**
     ⇒ quality cannot be retrofitted easily
     (e.g., scalability module *(aspect)*)

- **Is “cross-cutting” a better term?**
  - Not much: only deals with points 1 and 2 above
The Wisdom of the Ancients*

* “The ancients stole all our good new ideas” [M. Twain/ R.W. Emerson?]

- “All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament.”
  -- Vitruvius, On Architecture, Book X, 1st Century BC

- Software Application + Computer (Hardware) = Special-purpose machine
  -- Nancy Leveson, “Safeware”, 1995

Q: What impact do the physical characteristics of this have on...

...this?
The Physical World

Here Be Dragons

Software Physics
The essential complexities of the physical world:

- Physical distribution
- Modal behaviour
- Non-determinism (asynchrony)
- Concurrency
- Qualitative diversity
- Quantity can affect quality
It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail.

**Concurrency**

- Difficult to reason about concurrency
Facing the Physical World through Software
Where Software Meets Physics

- Everything that the software senses and performs is mediated by the platform and *is influenced by its physical properties*
Platforms: The Raw Material of Software

- **[Software] Platform**: The full complement of software and hardware required for a given application program to execute correctly.

Mainstream programming and modeling languages lack support for representing platforms and their characteristics!
What About Platform Independence?

- An important and useful notion
  - Helps abstract away irrelevant technological detail
  - Necessary for software portability
- **Platform independence does not mean platform ignorance**
  - There are ways of achieving platform independence that account for the influence of platform characteristics

Any claims of “platform independence” should be accompanied by clear statements of the range of platforms that the application is independent of.
What We Need to Know About Platforms

1. Its relevant quality of service characteristics (size, capacity, performance, bandwidth, etc.)
2. Its computing and communications structure
3. The deployment of application software components across the platform
What is MARTE?

- A domain-specific modeling language (DSML) for the design and analysis of modern cyber-physical systems
  - Modeling and Analysis of Real-Time and Embedded systems
  - Supplements UML (i.e., does not replace it)
  - Realized as a UML profile
What MARTE Adds to UML

1. SUPPORT FOR CONCISE AND SEMANTICALLY MEANINGFUL MODELING OF “CPS” SYSTEMS:
   - A domain-specific modeling language for modeling real-time, embedded, and cyber-physical systems (CPS)
   - Support for precise specifications of quality of service (QoS) characteristics (e.g., delays, memory capacities, CPU speeds, energy consumption)

2. SUPPORT FOR FORMAL ENGINEERING ANALYSES OF MODELS:
   - A generic framework for certain types of (automatable) quantitative analyses of UML models
   - Suited to computer-based automation
Example: "Bare" UML Model

```
ClockApp
«signal» tick()

Display
display(v: string)

Ticker
0..1

0..*

0..*

1

<hardware>
OS timer utility

HW interrupt (frequency?)

loop

sd

ReadWriteLock

0..1

1

How many?

Execution time?

Scheduling delay?

讨论

{(@t2 - @t1) <= 100}

Which units?
```
Annotating a UML Model with MARTE

- **Ticker**
  - «timerResource»
    - isPeriodic=true, duration=(100, us)

- **ClockApp**
  - «signal» tick()
  - «resourceUsage»
    - execTime = ((47*CPUrating), us)

- **Display**
  - «resourceUsage»
    - execTime = (1.5, us)
  - display(v:String)

- «swSchedulableResource»
  - isStaticSchedulingFeature=true, isPreemptable=false

- «hwDevice»
  - description="DSP1455A"
**Core Concept: Resource**

  
  “A source of supply of money, materials, staff and other assets that can be drawn upon...in order to function effectively”

- In MARTE, a platform is viewed as a collection of different types of resources, which can be drawn upon by applications
  
  - The finite nature of resources reflects the physical nature of the underlying hardware platform(s)

```
Platform ---- 1..* ---- Resource
             |
             \
             |
             Memory Resource
```

```
Computing Resource
```

etc.
In MARTE resources are viewed as **service providers**
- Consequently, applications are viewed as **service clients**

**Resource services are characterized by their**
- Functionality
- Quality of service (QoS)

**e.g. (platform services):**
- memory provisioning
- processing power
- bandwidth
- energy
- mutual exclusion
Core Concept: Quality of Service (QoS)

- **Quality of Service (QoS):**
  - A measure of the effectiveness of service provisioning

- **Two complementary perspectives on QoS**
  - **Required QoS:** the demand side (what applications require)
  - **Offered QoS:** the supply side (what platforms provide)

Many engineering analyses consist of calculating whether (QoS) supply can meet (QoS) demand

"Virtually every calculation an engineer performs...is a failure calculation...to provide the limits than cannot be exceeded"

-- Henry Petroski
QoS Compatibility

- We have powerful mechanisms for verifying functional compatibility (e.g., type theory) but relatively little support for verifying QoS compatibility.

Key engineering question:

\(\text{RequiredQoS} \leq \text{OfferedQoS}\)?
Why It is Difficult to Predict Software Properties

- Because platform resources are often shared
  - often by independently designed applications
  - Contention for resources

![Diagram showing resource contention](image-url)
MARTE: Quantitative QoS Values

♦ Expressed as an *amount of some physical measure*

♦ Need a means for specifying physical quantities
  ▪ *Value*: quantity
  ▪ *Dimension*: kind of quantity (e.g., time, length, speed)
  ▪ *Unit*: measurement unit (e.g., second, meter, km/h)

♦ However, additional optional qualifiers can also be attached to these values:
  ▪ *source*: estimated/calculated/required/measured
  ▪ *precision*
  ▪ *direction*: increasing/decreasing (for QoS comparison)
  ▪ *statQ*: maximum/minimum/mean/percentile/distribution
MARTE Predefined Types

- **SourceKind**
  - `est`, `meas`, `calc`, `req`

- **DirectionKind**
  - `incr`, `decr`

- **NFP_CommonType**
  - `expr`: YSL_EXPRESSION
  - `source`: SourceKind
  - `statQ`: StatisticalQualifierKind
  - `dir`: DirectionKind

- **NFP_Boolean**
  - `value`: Boolean

- **NFP_String**
  - `value`: String

- **NFP_Real**
  - `value`: Real

- **NFP_Duration**
  - `unit`: TimeUnitKind
  - `clock`: String
  - `precision`: Real

- **NFP_DataRate**
  - `unit`: DataRateUnitKind

- **NFP_Frequency**
  - `unit`: FrequencyUnitKind

- **NFP_Power**
  - `unit`: PowerUnitKind

- **NFP_Energy**
  - `unit`: EnergyUnitKind

- **NFP_Length**
  - `unit`: LengthUnitKind

- **NFP_Weight**
  - `unit`: WeightUnitKind

- **NFP_Area**
  - `unit`: AreaUnitKind
MARTE Measurement Units

- **Length Unit Kind**
  - `m` [baseUnit = m, convFactor = 1E-2]
  - `cm` [baseUnit = m, convFactor = 1E-2]

- **Weight Unit Kind**
  - `g` [baseUnit = g, convFactor = 1E-3]
  - `mg` [baseUnit = g, convFactor = 1E-3]

- **Time Unit Kind**
  - `s`
  - `ms` [baseUnit = s, convFactor = 0.001]
  - `us` [baseUnit = ms, convFactor = 0.001]
  - `min` [baseUnit = s, convFactor = 60]
  - `hrs` [baseUnit = min, convFactor = 60]
  - `dys` [baseUnit = hrs, convFactor = 24]

- **Power Unit Kind**
  - `W`
  - `mW` [baseUnit = W, convFactor = 1E-3]
  - `KW` [baseUnit = W, convFactor = 1E3]

- **Energy Unit Kind**
  - `J`
  - `KJ` [baseUnit = J, convFactor = 1E3]
  - `Wh` [baseUnit = J, convFactor = 2.778E-4]
  - `kWh` [baseUnit = Wh, convFactor = 1E3]
  - `mWh` [baseUnit = Wh, convFactor = 1E-3]

- **Frequency Unit Kind**
  - `Hz`
  - `KHz` [baseUnit = Hz, convFactor = 1E3]
  - `MHz` [baseUnit = Hz, convFactor = 1E6]
  - `GHz` [baseUnit = Hz, convFactor = 1E9]
  - `rpm` [baseUnit = Hz, convFactor = 0.0167]

- **Data Size Unit Kind**
  - `bit`
  - `Byte` [baseUnit = bit, convFactor = 8]
  - `KB` [baseUnit = Byte, convFactor = 1024]
  - `MB` [baseUnit = KB, convFactor = 1024]
  - `GB` [baseUnit = MB, convFactor = 1024]

- **Area Unit Kind**
  - `mm2`
  - `um2` [baseUnit = mm2, convFactor = 1E-6]

- **Data Rate Unit Kind**
  - `b/s`
  - `Kb/s` [baseUnit = b/s, convFactor = 1024]
  - `Mb/s` [baseUnit = b/s, convFactor = 1024]
MARTE: Representing Time

**Structure of Time**
- time bases
- multiple time bases
- instants
- time relationships

**Access to Time**
- clocks
- logical clocks
- chronometric clocks
- current time

**TB1**

**TB2**

**Using Time**
- timed elements
- timed events
- timed actions
- timed constraints
Example: Time Annotations

Extended duration intervals with bound « [ ] » specification

Jitter constraint between two successive occurrences

Duration expression

Constraint in an observation with condition expression

Instant Interval Constraint
MARTE: Physical Properties Analysis Approach

UML/MARTE MODEL

APPLICATION (ARCHITECTURE) MODEL

ALLOCATION (DEPLOYMENT)

PLATFORM MODEL

EQUIVALENT ANALYSIS MODEL

M2M Xform

COMPUTER-AIDED ANALYSIS

DESIGNER
**Generic Quantitative Analysis Model (GQAM)**

- Captures the pattern common to many different kinds of quantitative analyses (using concepts from GRM)
  - Specialized for each specific analysis kind

**Demand Side**
- Work demand arrivals (Workload intensity)
  - (e.g., event arrivals, time triggers)

**Supply Side**
- Resource1
  - (e.g., disk)
- ResourceN
  - (e.g., CPU)

**Work Characterization (Scenarios)**
- (e.g., application programs, system programs, etc.)

**Analysis Context**
Performance Analysis Example - Context

- An interaction (seq. diagram representation)

```plaintext
<<GaPerformanceContext>> {contextParams= in$Nusers, in$ThinkTime, in$Images, in$R}
```

1: getHomePage

```plaintext
<<GaWorkload Event>> {closed (population=Nusers, extDelay=ThinkTime)}
```

```
<<PaStep>> {hostDemand = (1,ms),
respT={((1,s,percent95),req),
((R,$,percent95),calc)}
```

```
<<PaCommStep>> {msgSize=(2.9, KB)}
```

2: getCustomerData

```
<<PaStep>> {hostDemand = (2,ms)}
```

3:

```
<<PaStep>>
{hostDemand = (2,ms)}
```

Slide courtesy of D. Petriu, M. Woodside (Carleton U.)
Summary

- Software that interacts with the physical world is pervasive and fundamental to everyday life
- Our mainstream programming languages are not well suited for this environment
- Needed: Higher-order languages that are more directly connected to the environment
A shameless plug

Available from a web page/bookstore near you:

Publisher: Morgan Kaufmann
ISBN: 978-0-12-416619-6
THANK YOU
QUESTIONS, COMMENTS, ARGUMENTS...