Implementation-First Approach of Developing Formal Semantics of a Simulation Language in VDM-SL

Tomohiro Oda, Gael Dur, Stephane Ducasse, Hugo Daniel Macedo, Software Research Associates, Inc. Shizuoka University University of Lille Aarhus University
re:mobidyc
the overview
reːmobidyc

A multi-agent simulation platform for population dynamics in biology and ecology.

Major competitors = variants of LOGO

- supports domain-specific language features
- demands less "programming" skills
- serves as a tool for science
The execution of simulation model is not the goal, but the math model is.

The real world phenomina

The simulation model reproduces and explains the real world phenomina.

simulation model

The simulation model implements the math model.

math model
Implementation First
the semantics and its interpreter
development process of reːmobidyc

design rationale
- no imperative programming
- no infinite loops
- GUI-based modeling environment
- type checking with measuring units
- record everything on the heap memory
- reproducible including pseudo-random numbers

implementation
- GUI-based modeller
- GUI-based visualization
- interpreter
- type checker
- memory models

formal specification
- interpretation
- memory model
Why Formal Semantics?

- **grounding to mathematics**
  - The goal of the user is to develop and validate a MATH MODEL in biology.

- **portability**
  - The formal semantics makes it easy to implement a fully compatible interpreters/transpilers.
  - The reːmobidyc modeling language should not depend on our implementation.

- **uncommon memory model**
  - The memory model with synchronous updates for time-series data is not common among existing programming languages and therefore needs concise and unambiguous definition.

with expectations to improve code quality
Why not semantics first?

- GUI-based modeling environment and execution/analysis tools

- while true do
  (Computer_scientists`implement_a_language_feature();
   Biologists`try_it())
Specification
VDM-SL modules

- **AST.vdmsl (ASTTest.vdmsl)**
  - 50 records, 20 unions and 19 constants
  - 7 functions to manipulate AST
- **Evaluation.vdmsl (EvaluationTest.vdmsl)**
  - 9 eval operations, 3 variable access operations and 10 primitive operations
- **Interpreter.vdmsl (InterpreterTest.vdmsl)**
  - 27 operations to manage simulation models, evaluation contexts and random seeds
- **Memory.vdmsl (MemoryTest.vdmsl)**
  - read/write operations, synchronous updates and snapshotting
- **Random.vdmsl (RandomTest.vdmsl)**
  - Fishman-Moore random number generator
- **Unit.vdmsl (UnitTest.vdmsl)**
  - Measurement units
- **MATH.vdmsl**
- **UnitTesting.vdmsl**
state definition of the memory model

state Memory of

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vals</td>
<td>map Address to real</td>
</tr>
<tr>
<td>next</td>
<td>map Address to real</td>
</tr>
<tr>
<td>delta</td>
<td>map Address to real</td>
</tr>
<tr>
<td>nextAvailableSlot</td>
<td>[Address]</td>
</tr>
<tr>
<td>world</td>
<td>Address</td>
</tr>
<tr>
<td>patchBase</td>
<td>[Address]</td>
</tr>
<tr>
<td>xDivisions</td>
<td>[nat1]</td>
</tr>
<tr>
<td>yDivisions</td>
<td>[nat1]</td>
</tr>
<tr>
<td>animats</td>
<td>map Address to (AST`Identifier * nat1)</td>
</tr>
<tr>
<td>newBorns</td>
<td>map Address to (AST`Identifier * nat1)</td>
</tr>
<tr>
<td>deads</td>
<td>set of Address</td>
</tr>
<tr>
<td>valuesStorage</td>
<td>seq of (map Address to real)</td>
</tr>
<tr>
<td>animatsStorage</td>
<td>seq of (map Address to (AST`Identifier * nat1))</td>
</tr>
<tr>
<td>ticks</td>
<td>nat</td>
</tr>
</tbody>
</table>

init s == s = mk_Memory({|->}, {|->}, {|->}, nil, 1, nil, nil, nil, {|->}, {|->}, {}, [], [], 0)
end
Conventional memory model

\[ a = \text{self.x} + \cos(\text{self.heading}) \]

<table>
<thead>
<tr>
<th>agent</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>address</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>vals</td>
<td>12.5</td>
<td>-5.2</td>
</tr>
</tbody>
</table>

\[ \text{self.x} = -2.4 \]
\[ \text{self.heading} += -0.2 \]

The resulting value of \( a \) depends on the order of execution of the scripts.
three cells per address

\[ a = \text{my x} + \cos(\text{my heading}) \]

<table>
<thead>
<tr>
<th>agent</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>x</td>
<td>y</td>
<td>heading</td>
<td>x</td>
</tr>
<tr>
<td>address</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>vals</td>
<td>read</td>
<td>12.5</td>
<td>-5.2</td>
<td>1.23</td>
</tr>
<tr>
<td>delta</td>
<td>write</td>
<td>12.5</td>
<td>-5.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>next</td>
<td>add</td>
<td>-2.4</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

\[ \text{my x}' = -2.4 \text{[m]} \]

\[ \text{my } \Delta \text{heading}' = -0.2 \text{[rad]} \]

write : Address * real ==> ()
write(address, data) ==
  next(address) := data

writeDelta : Address * real ==> ()
writeDelta(address, data) ==
  if address in set dom delta
  then delta(address) := delta(address) + data
  else exit ADDRESS_ERROR;
synchronous update

time 354

<table>
<thead>
<tr>
<th>address</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>vals</td>
<td>read</td>
<td>12.5</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-4.5</td>
<td>34.1</td>
<td>3.0</td>
</tr>
<tr>
<td>next</td>
<td>write</td>
<td>12.5</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-4.5</td>
<td>34.1</td>
<td>-0.2</td>
</tr>
<tr>
<td>delta</td>
<td>write</td>
<td>-2.4</td>
<td>0.0</td>
<td>0.0</td>
<td>-2.4</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

time 355

<table>
<thead>
<tr>
<th>address</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>vals</td>
<td>read</td>
<td>10.1</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-6.9</td>
<td>34.1</td>
<td>-0.2</td>
</tr>
<tr>
<td>next</td>
<td>write</td>
<td>10.1</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-6.9</td>
<td>34.1</td>
<td>-0.2</td>
</tr>
<tr>
<td>delta</td>
<td>write</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

{addr |-&gt; next(addr) + (if addr in set dom delta then delta(addr) else 0) | addr in set dom next \ deads}
time-series memory

<table>
<thead>
<tr>
<th>address</th>
<th>vals</th>
<th>read</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
<td>write</td>
<td>10.1</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-6.9</td>
<td>34.1</td>
<td>-0.2</td>
<td></td>
</tr>
<tr>
<td>delta</td>
<td></td>
<td>write</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

**store**: () ==> ()

store() ==
(valuesStorage := valuesStorage ^ [{a |-> next(a) + (if a in set dom delta then delta(a) else 0) | a in set dom next \ deads}];
animatsStorage := animatsStorage ^ [deads <-: animats munion newBorns];

pre ticks = len valuesStorage and ticks = len animatsStorage;

backend storage (on-memory, file system, null, ...)

<table>
<thead>
<tr>
<th>address</th>
<th>time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>22.1</td>
<td>-5.2</td>
<td>-4.2</td>
<td>5.1</td>
<td>34.1</td>
<td>-1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>351</td>
<td>19.7</td>
<td>-5.2</td>
<td>-4.2</td>
<td>2.7</td>
<td>34.1</td>
<td>2.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>17.3</td>
<td>-5.2</td>
<td>3.2</td>
<td>0.3</td>
<td>34.1</td>
<td>5.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>353</td>
<td>14.9</td>
<td>-5.2</td>
<td>-5.2</td>
<td>-2.1</td>
<td>34.1</td>
<td>-5.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>354</td>
<td>12.5</td>
<td>-5.2</td>
<td>1.23</td>
<td>-4.5</td>
<td>34.1</td>
<td>3.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>355</td>
<td>10.1</td>
<td>-5.2</td>
<td>-0.2</td>
<td>-6.9</td>
<td>34.1</td>
<td>-0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

356...
Memory model for re:mobidyc

The objective of re:mobidyc is to enable the user to analyze
● what happens
● why it happens
● how it happens

We need a memory model with
● lightweight snapshot
  ○ dump heap space at every timestep
  ○ to trace cause and effect
● synchronous update
  ○ delay write to memory to the interval of timestep
  ○ to isolate effects of each action and
  ○ to eliminate intermediate state
Implementation First Revisited
the development process
## Development of reːmobidyc in numbers

<table>
<thead>
<tr>
<th>event</th>
<th>date</th>
<th>Pharo LOC (all)</th>
<th>Pharo LOC (interp)</th>
<th>Pharo tests</th>
<th>VDM LOC</th>
<th>VDM tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation started</td>
<td>Oct 2019</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Jan 2020</td>
<td>1,499</td>
<td>1,150</td>
<td>21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Jan 2021</td>
<td>12,936</td>
<td>7,268</td>
<td>214</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Jan 2022</td>
<td>19,474</td>
<td>9,526</td>
<td>276</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Specification started</td>
<td>Aug 2022</td>
<td>26,330</td>
<td>11,990</td>
<td>320</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Dec 2022</td>
<td>30,114</td>
<td>13,205</td>
<td>338</td>
<td>1,364</td>
<td>113</td>
</tr>
</tbody>
</table>

18
Thoughts on implementation-first Lightweight Formal Method

- The first exploration in Pharo took long time.
- Just writing a formal specification did not cost much.
  - The specification took only 4 months in 3 years development so far.
- The specification in VDM was compressed into 10% of the implementation
  - 13,205 LOC in Pharo → 1,364 LOC in VDM

What if we started with the specification phase first?

... In the development of ViennaVM,

- The first exploration in VDM took long time and just writing C did not cost much.
Explorative Development Process

Front loading effect:

Developing a formal specification reduces the cost of implementation because semantic errors of functionalities are eliminated at the specification phase and the implementers can get focused on implementation issues.

reːmobidyc project:

Developing a prototypical implementation reduced the cost of specification because mis-assumptions on the problem domain were eliminated at the prototyping phase and the specifiers could get focused on semantic issues.
Conclusion

Just writing a formal specification is not costly, but the exploration process is.

- Learning the problem domain
- Finding affordable solution
- Planning for realisation

**Implementation-first approach:**
Models in VDM as a summary of the exploratoriation by implementation languages, and also as a pivot to the next iteration of the development cycle.

- Some of software developers are afraid of cost and risk of adopting formal specification.
- The apparent cost of formal specification can be compressed. (10% in re:mobidyc)
- Even if VDM modeling does not go well, the development can go without a formal specification.
Thank you!