EXECUTABLE UML AND MBSE

What Executable UML does, how it is totally different from UML, and how it fits with other Executable languages

Leon Starr
leon_starr@modelint.com

MODEL INTEGRATION LLC

www.modelint.com
FUNDAMENTAL IDEA OF MODELING

A model isn’t the thing you are modeling.

To be useful, a model must omit certain aspects of the real world subject to focus on the details of interest.
EXCLUDED DETAILS ARE STILL IMPORTANT

Just because a detail is systematically ignored, doesn’t mean that it is not important

Schematic

Focuses on component properties and connectivity

Excludes layout details

Layout diagram

Focuses on layout geometry

Is applied to schematic
What about UML?

Simulink

control

electronics/optronics

(usage at SAAB)

physical systems

structure
UML PURPOSE AND SKILLS

Requirements modeling
High level code
Lower level code
Just sequences
Use cases
Lot’s of other uses

UML profiles
Executable UML

UML is not a modeling language.
It is a standardized set of object-oriented notations for many purposes.
EXECUTABLE UML

Modeling language

Our purpose is to model requirements imposed by the real world:

- Information
- Rules and constraints
- Real world behavior
- Essential computation

and to separate these from the platform.

Mathematical, executable semantics for each model type

Doesn’t presume object-oriented implementation
AN EXAMPLE XUML MODEL
RULES / REQUIREMENTS

1) A controller can not direct air traffic while off duty.
2) An on duty controller must be logged into a duty station.
3) A duty station may or may not be available.
4) A control zone must have its traffic directed by one air traffic controller at all times.
5) An air traffic controller may not work a shift longer than two hours and fifteen minutes.
Air Traffic Controllers

<table>
<thead>
<tr>
<th>ID {I}</th>
<th>Name</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATC53</td>
<td>Toshiko</td>
<td>A</td>
</tr>
<tr>
<td>ATC67</td>
<td>Gwen</td>
<td>B</td>
</tr>
<tr>
<td>ATC51</td>
<td>Ianto</td>
<td>C</td>
</tr>
</tbody>
</table>

On Duty Controllers

<table>
<thead>
<tr>
<th>ID {I, R1}</th>
<th>Time logged in</th>
<th>Duty Station {R2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATC53</td>
<td>9/27/13 15:00</td>
<td>DS2</td>
</tr>
<tr>
<td>ATC67</td>
<td>9/27/13 11:00</td>
<td>DS1</td>
</tr>
</tbody>
</table>

Off Duty Controllers

<table>
<thead>
<tr>
<th>ID {I, R1}</th>
<th>Last shift ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATC51</td>
<td>9-26-13 17:00</td>
</tr>
</tbody>
</table>

Duty Station

<table>
<thead>
<tr>
<th>Number {I}</th>
<th>Location</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>Front</td>
<td>20</td>
</tr>
<tr>
<td>DS3</td>
<td>Center</td>
<td>30</td>
</tr>
<tr>
<td>DS2</td>
<td>Front</td>
<td>45</td>
</tr>
</tbody>
</table>

Control Zones

<table>
<thead>
<tr>
<th>Name</th>
<th>Traffic</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJC18C</td>
<td>30</td>
<td>ATC53</td>
</tr>
<tr>
<td>SFO37B</td>
<td>25</td>
<td>ATC53</td>
</tr>
<tr>
<td>OAK21C</td>
<td>15</td>
<td>ATC67</td>
</tr>
</tbody>
</table>
Off Duty ATC goes On Duty?
If there is only one On Duty Controller, can he or she go Off Duty?
If every Control Zone is being directed, can another Off Duty Controller log in?
Do we know when a shift should end?
Do we know when a shift should end?

```
Shift Specification
Name {1} : Name
Min break : Duration
Max shift : Duration

Air Traffic Controller
ID {1} : Employee ID
Name : Name
Rating : Experience level

Off Duty Controller
ID {1, R1}
Last shift ended : Date

On Duty Controller
ID {1, R1}
Time logged in : Date
Station {R3}

Control Zone
Name {1} : Czone Name
Traffic : Aircraft quantity
Controller {R2}

Duty Station
Number {1} : Station number
Location : Name
Capacity : Aircraft maximum

has traffic directed by
is directing traffic within
is being used by
is logged into
```

0..1

1

0..*

R1

{ disjoint, complete }

R2

R3
EXCLUDED DETAILS

- States, algorithms, functions are filtered out
- Platform specific features are filtered out
- Implementation choices are filtered out

Lean modeling language – minimal symbols
So you can define and evaluate the application logic without distraction
migrate to On Duty Controller
my station := Duty Station( Number: in.Station )
link /R3/my station
Time logged in = _now.HMS
Logged in -> me
In use -> my station

Actions

State models

Class Model
Verifying Adequate Break

the shift spec. = Shift Specification() // selects singleton
if ( _now - Last shift ended < the shift spec.Min break )
   Log in( in.Station ) -> me
else
   Cannot go on duty -> me

Logging In

migrate to On Duty Controller
my station. = Duty Station( Number: in.Station ) & /R3/my station // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station

ON DUTY

Handing off Control Zone

hoff zone. = /R2/Control Zone( Name: in.Zone )
if in.Controller == ID
   UI.Cannot handoff to self( Controller: in.Controller )
else
   new controller. = On Duty Controller( ID: in.Controller )
   swap hoff zone/R2/On Duty Controller with new controller
   !new missing: UI.Unknown controller( Controller: in.Controller )
   !old missing: UI.Zone not handled by( Controller: ID )
} // swaps controllers and checks for errors
Handoff complete -> me

Verifying Full Handoff

if /R1/On Duty Controller/R2/Control Zone({
   Must handoff zones -> me
   UI.Control Zones Active( ATC: ID )
}) else
   Log out -> me

Logging Out

User leaving -> /R3/Duty Station
migrate to Off Duty Controller
Last shift ended = _now.HMS
Off duty -> me

Log in( Station )

Log out

Ready for duty( Station )

Ready for a break

Air Traffic Controller

ID (I) : Employee ID
Name : Name
Rating : Experience level

Log in( Station )

Logged in

migrate to On Duty Controller
my station. = Duty Station( Number: in.Station ) & /R3/my station // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station

Logging Out

User leaving -> /R3/Duty Station
migrate to Off Duty Controller
Last shift ended = _now.HMS
Off duty -> me

Log in( Station )

Logged in

migrate to On Duty Controller
my station. = Duty Station( Number: in.Station ) & /R3/my station // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station

Handoff complete

Handing off Control Zone

hoff zone. = /R2/Control Zone( Name: in.Zone )
if in.Controller == ID
   UI.Cannot handoff to self( Controller: in.Controller )
else
   new controller. = On Duty Controller( ID: in.Controller )
   swap hoff zone/R2/On Duty Controller with new controller
   !new missing: UI.Unknown controller( Controller: in.Controller )
   !old missing: UI.Zone not handled by( Controller: ID )
} // swaps controllers and checks for errors
Handoff complete -> me
EXCLUDED DETAILS

- How execution is implemented
- How synchronization is implemented
- Distribution across processes/processors
### Logging In

- migrate to On Duty Controller
- my station .= Duty Station( Number: in.Station )
- & /R3/my station // link station
- Time logged in = _now.HMS
- Logged in -> me
- In use -> my station

#### Non-essential ordering of computation

#### Data access implementation
FOUNDATION SEMANTICS of the xUML language
A class is a set of things in the real world such that all things in the set:

- have the same characteristics
- exhibit the same behavior
- are constrained by the same rules
RELATIONAL THEORY

SET THEORY

FUNCTIONS

$y = f(x)$

PREDICATE LOGIC

$\forall x \text{ such that ...}$
A CLASS PREDICATE

- Each class in your model is an n-ary predicate where n is the number of attributes
- An ATC has an ID $i$, a Name $n$ and a Rating $r$
- We can turn this into a proposition (true/false statement) by instantiating it
- ATC53 is named Ianto and has Rating B (true/false statement)
- If the instance does not exist, it is false
STATE MODEL SEMANTICS

Same characteristics, rules, behavior

Common lifecycle

Aircraft
- Tail number (I)
- Altitude
- Airspeed
- Heading
- Position

abstraction

abstraction
THE THEORY

Moore state machine

Leslie Lamport

Time, Clocks, and the Ordering of Events in a Distributed System

Video animation of the platform independent synchronization rules

Also described here
What happens when event occurs while instance is executing a state activity?

Are events prioritized?

What is the duration of an activity?

Can event arrival order be guaranteed?
ACTION SEMANTICS

- Access data from class model
  - Based on relational semantics
- Send/receive event data on state model
  - Follows state machine execution rules
- Perform computations
  - Essential sequence only
  - No data implementation assumptions
THE THEORY

Data access: Relational operations

Sequencing: Data flow semantics

Data types: Type theory
DOMAINS
A domain is a distinct subject matter with its own vocabulary, rules, constraints and behavior.

Example: Linear Algebra

Not a domain: “Stuff that runs on processor X”, “Stuff in library Y”, “Stuff dept Z is coding up”

Domain partition of a system excludes details of deployment onto any particular platform features.
EXAMPLE DOMAIN SEPARATION

Elevator Management

- Floor selection
- Cabin dispatching
- Door open/close timing

Transport

- Safe acceleration
- Precise transport

Elevator uses Transport Bridge between domains

gotoFloor (Cabin 3, Floor 6)

cabinArrived ()

move (Load 14, Position 334.25, Ramp 3B)

moveCompleted ()
A COMPLETE DOMAIN CHART

Elevator Management

Transport

UI

Signal I/O

Model Execution Architecture

xUML

Hand coded

Existing

C
CODE GENERATION
TRANSLATION PRINCIPLES

➤ Models are not modified by translation process

➤ Information is added, usually via some DSL (domain specific language / model markup)

➤ A runtime MX (model execution) platform is supplied

➤ A code generator which populates this platform (from the DSL) is provided

➤ The MX and code generator work for a class of platforms, e.g.
  ➤ Embedded microcontroller
  ➤ Cloud
  ➤ Fault tolerant distributed
**EXAMPLE APPROACH FOR A MICROCONTROLLER**

Pycca Translation Workflow

<table>
<thead>
<tr>
<th>Platform Independent</th>
<th>Platform Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify Domains</td>
<td>Create Pycca Script</td>
</tr>
<tr>
<td>Build &amp; Document Models</td>
<td>Write Bridge Code</td>
</tr>
<tr>
<td>Specify Domain Mapping</td>
<td>Code / Provide Non-Modeled Domains</td>
</tr>
<tr>
<td>Populate Models</td>
<td>Transform and Populate</td>
</tr>
<tr>
<td>Populate Mappings</td>
<td>Text Editor</td>
</tr>
</tbody>
</table>

Use any model draw and text editing tools

Spreadsheets

- Provided
- ST/MX Runtime
- Model Code
- Compile

- Provided
- Model Object Code
- Non-Model Object Code

- Coded by hand
- Non-Model Code

- Pass along
- System Executable
SUMMARY

- Executable UML is a complete modeling language
  - Concrete, unambiguous models of application requirements
  - Executable, platform independent modeling
  - Mathematical foundations
  - Does not presume an OO implementation
  - Domains accommodate other MBSE languages

- Model Translation
  - Models are not modified to generate code
  - So models can be reused and redeployed
  - Models are stable while platform details change