Designing 3D Video Games with Models at Run-time

MODELSWARD 2018 keynote

Direct access to the presentation from your laptop/mobile: franckbarbier.com/MODELSWARD2018

Sponsored by the European ECSEL MegaM@Rt² (MegaModelling at Runtime) project

Model Driven Development (MDD), simply...

MDD is a software engineering method whose first phase is the creation of models (from requirements) and the second phase is the derivation of code from models. The discriminating nature of models creates effective properties (opportunities) that lead to greater software quality and greater productivity (rapidity in fact) at development time.

MDD daily practice: success or failure? (1/2)
Roughly speaking, model production is time-consuming and then costly. In this logic, MDD is a software engineering method (i.e., "a way of making software") whose goal is "fighting" code-only approaches, say:

- the no-method approach (a.k.a. the messy method)
- the agile method (test driven...)
- <your method’s name here>
- ...

From an economical viewpoint, one has to demonstrate that the return on investment is "better" with models. In other words, the time "lost" when producing models definitely leads to the reduction of software cost prices in all conditions.

MDD daily practice: success or failure? (2/2)

Considering 1000 (randomly chosen) starting sizeable software projects in any kind of business sector and any kind of technological context (Enterprise (payroll, shipping, supply chain...), Web, safety-critical, HPC...), how many will use MDD?

- 0 (← your answer → hopefully, no...)
- 1 (← your answer → cruel disappointment)
- 10 (← your answer → realism)
- 100 (← your answer → wishful thinking)
- all (← your answer → dream)

Towards “better” models?

Modeling formalisms (or "languages") may (or not) have mathematical bases in the sense that models as usages (or expressions, or instances...) of modeling formalisms have one-to-one counterparts in idealistic worlds (theories) created by mathematics. We may then talk about "formal semantics". Since theories support theorems, a model may be declared "correct" or "incorrect" from the fact that, as a particular usage of the modeling language, it fulfills (or not) some theorems as general rules.

Mathematical theories can be approximated in computer programs leading to some kind of (useful) operationalization to deal with models in size, number, complexity... in large software projects. The common problem is the fact that software engineers/developers do not like, by culture, mathematics!
Towards “executable” models?

Laziness and/or pragmatism lead(s) us to ignore the equipment of modeling languages with mathematical bases (e.g., UML). From the postulate that UML models “seem” readable (i.e., one is able to create some sense from reading), the arising question is: Is this sense unique? In natural language, non-formal semantics is something both common, useful, and powerful for human beings! As an illustration, in French, “Je suis gentil” ("He is nice") both means:

- “Il est gentil” ("He is nice").
- “Il est stupide” ("He is stupid")

So, beyond, language executability is the ability to infer sense from model reading (interpretation) in a systematic way. Systematization creates the opportunity to implement interpretation engines, but, contrary to natural language interpretation, execution must lead to predictable (and thus unique) results. Example:

- StateChart XML formalism (Harel’s Statecharts semantics variant)
- Usage

```xml
<state id="FSC_connected">
  <transition event="PSC_connection_request" target="Crisis_details_exchange"/>
</state>
```
- Operationalization: PauWare (Java), SCIWin (JavaScript),…

As confirmed in [Harel & Runke - Meaningful Modeling: What's the Semantics of "Semantics"? - IEEE Computer, October 2004], formality, executability and semantics are orthogonal visions on models that are intended to be composed to create modeling languages.

The Holy Grail? Models at run-time

As abstractions (and thus approximations), models do not conform to (generated) code while the contrary is true. Inevitable code adaptation at maintenance time breaks the initial mapping created at code generation (a.k.a. derivation) time. This is the main factor of MDD failure.

Models at run-time then aim at embodying design-time models at run-time so that code adaptation is nothing more than model adaptation”. From that postulate, there is an annoying contradiction: models are no longer abstractions (they lose their discriminating nature). Example in PauWare (Java):

```java
AbstractStatechart_Init = new Statechart("Init");
AbstractStatechart_PSC_connected = new Statechart("PSC connected");
AbstractStatechart_PSC_connected = new Statechart("PSC connected");
AbstractStatechart_Crisis_details_exchange = new Statechart("Crisis details exchange");
... // Overall state machine creation:
  _ICMS_state_machine = new 
  Statechart_monitor(_Init, xor(_PSC_connected), xor(_PSC_connected), xor(_Crisis_details_exchange), xor(_...), ...);
... // Transitions’ declaration:
  _ICMS_state_machine._fires(_PSC_connection_request, _PSC_connected, _Crisis_details_exchange);
```

"So, models at run-time have to be executable..."
"This definition is not the common (shared) definition of "models at run-time" coming from the MDD community. In that sense, "embedded models" may be a more appropriate expression.

PauWare demo.

"Embedded models" remain true models...

In this JavaScript code, one may observe that data transformations are expressed in native JavaScript while control is expressed in SCXML/SCION, which itself is empowered by JavaScript.
Code, models, is there really a difference?

• For people who dislike models, “software = code only” and models are (often) only appropriate for a posteriori documentation

• For model purists, hiding as long as possible the (programming) code is the key. In this line of reasoning, ALF is a neural modeling language for avoiding the expression of actions in C++, Java... To what limit, this process is sustainable?

• Architectures (as PSMs, between PIMs and code) are often the tricky issue. Does technology eclecticism actually allow the “drawing” of architectures without references to technology features?

“Reactive Programming” or the event bombing

Intuitively, the Internet/Web of Things moves us to some unencountered asynchronicity by which software components have to re-create unanticipated synchronization through very intensive event exchange, i.e., the event bombing. For software components, this vision imposes a strict internal organization that may be instrumented by “embedded models”.

• Visions (example): The Reactive Manifesto

• Supports (example): ReactiveX

• Frond-end Web frameworks (example): The SAM (State-Action-Model) Pattern aims at revisiting the MVC paradigm for front-end Web dev.

“Embedded models” for the Internet/Web of
class EventSniffer extends android.content.BroadcastReceiver {
    @Override
    public void onReceive(android.content.Context arg0, android.content.Intent arg1) {
        if (arg1 != null && arg1.getAction() != null) {
            android.content.Intent intent = new android.content.Intent(arg0, Android_energy_management_example.class);
            intent.setAction(arg1.getAction());
            arg0.startService(intent);
        }
    }
}

public class Android_energy_management_example extends android.app.Service {
    EventSniffer _event_sniffer;
    AbstractStatechart _Nominal_Energy_Level;
    AbstractStatechart _Critical_Energy_Level;
    ... // Other states here..
}

Android battery management (notification subscription)

Android battery management (notification reaction with PauWare)
AbstractStatechart_monitor._Android_energy_management_example_state_machine;

public void start() throws Statechart_exception {
  _Android_example_state_machine.fires(android.content.Intent.ACTION_BATTERY_LOW, _Nominal_Energy_Level, _Critical_Energy_Level);
  _Android_example_state_machine.fires(android.content.Intent.ACTION_POWER_CONNECTED, _Critical_Energy_Level, _Critical_Energy_Level);
}

3D Video Game: “embedded models” are greatly helpful! Examples:

- Zoom management with right mouse button is parallel to any other kind of management
- Camera filming depends upon precise states of the the overall game state machine
- History states allow to straightforwardly recover game situations
- Guards allow the immediate natural testing of character types in order to deal with collisions in various ways

3D Video Game demo.
Lessons learned from *3D Video Game*

- Business code size is divided by 3-5 depending upon the business logic complexity
- Enhanced maintenance through higher readability in particular
- Code format is strongly close to test expression and thus appropriate: preconditions, postconditions, invariants and variants may properly rely on states or logical state combinations (“embedded models” ≈ “test driven”?)
- A 3D video game is in essence a proof-of-concept for reactivity

The end... Thanks for your trust and attention.
Who is the model?

"Model" synonyms: copy, imitation, abstraction, approximation...

Same question for who prefers men

Towards engineering: model-driven reasoning
From difficulties to better future?

- Miller's law: "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information"
  → models (as "abstraction" = "omission") hide details to allow us reasoning on less information
- Models (as "approximation") take us (temporarily) away from reality, but we cannot ignore a (hard?) return to that reality
  → "system in production"
- The (famous) idea of "separation of concerns" to deal with complexity, but concerns are never independent and all systems are complex (otherwise, they aren't "systems") → multi-physics, multi-scale models
- So, in my opinion, modeling still remains challenging:
  - Ex. of bad research direction: formality
  - Ex. of good research direction: intuitiveness

Nonetheless, what might be engineering without models?