Model Based Development: some motivation and overview

Bengt Jonsson
Model Based Dev. 2020
Improving Software Development

- Programming is hard
  - Takes a lot of time
  - Takes much effort to get right and efficient
  - Good verification (testing) is hard

- Challenges:
  - Increase productivity (make it more efficient)
  - Increase quality (get it right and efficient)
  - Increase confidence (verify more efficiently)
Doing it the hard way: Waterfall

Major problem: Verification and Testing can be done only after coding →
Expensive correction cycle

Model Based Design: Perform Verification, Coding, Evaluation, directly on Functional Design
⇒ It must have the form of a precise MODEL
Model Based Design [Mathworks]

- **Model is the design**
  - functionality and performance
- **Used for continuous verification and validation**
  - typically, by simulation
- **Automatic code generation guarantees consistency with implementation**
- **Updates in design (model) are immediately reflected in verification and implementation**

*Figure 1 – Elements of model-based design*
Engineering is Based on Models

Filippo Brunelleschi’s design for the dome of the cathedral of Santa Maria del Fiore in Florence remains one of the most towering achievements of Renaissance architecture. Completed in 1436, the dome remains a remarkable feat of design and engineering. Its span of more than 140 feet exceeds St Paul’s in London and St Peter’s in Rome, and even outdoes the Capitol in Washington, D.C., making it the largest dome ever constructed using bricks and mortar. When work on the dome began in 1420 Brunelleschi was virtually unknown. Sixteen years later the dome was built, and its architect was a superstar.
Model Based Design

Why Model Based Design for Embedded Systems?
- Higher level of abstraction => more productive
- Code generation from model
- Platform independent => reuse
- Early Verification and Validation
- Documentation

Development Flow (simplistic)
1. Create a mathematical model of “all” the parts of the system
   - Physical world, hardware platform, network, sensors and actuators
   - Control system, software environment
2. Verify and Validate the model
3. Optimize parameters, for performance, robustness, ..
4. Generate the implementation from the model
   - Most design errors should have been caught in step 2.
Use of models (ctd.)

Models can contain much more detail than text documents
Less ambiguous about, e.g., functionality (run and see)
Models can be shared with other engineers:
- Engineers do not have to generate their own models from text specifications.
- Same model can be used by several engineers for different purposes in the design process.
- Component models can be used in larger systems.
- (ideally) Models supplied by manufacturers accurately describe the behavior and performance of their component implementations.
Use of models throughout design process

- **Model-in-the-Loop (MIL):**
  - Controlled environment represented by model during simulation

- **Software-in-the-Loop (SIL) Simulation**
  - Software realization of embedded control system is simulated with model of controlled environment

- **Hardware-in-the-Loop (HIL) Simulation**
  - Physical Hardware realization of embedded control system is simulated with model of controlled environment
  - Using actual physical hardware interfaces.
Example Simulink Model

Found by typing "sf_car" to MATLAB prompt
MIL Testing

- By recent predictions from the automotive sector, more than 100 million km of road driving would be required for the thorough validation of an automated car. Only with significantly better techniques will we be able to produce software components of sufficiently high quality to be usable in future integrated system.
Tools for Embedded System Design

Simulink: de facto standard

From [Tripakis: Dagstuhl 14]
Different Kinds of UML Models

Different kinds of systems, different views on a system require different forms of models

E.g., in UML

- Class diagrams
- Object Diagram
- Component Diagram
- Deployment Diagram
- Use Case Diagram
- Sequence Diagram
- Interaction Diagram
- Statechart Diagram
- Activity Diagram

The following examples are from

Class Diagrams

model the static view of an application
Deployment Diagrams

visualize the physical components in a system. These components can be libraries, packages, files etc.
Sequence Diagram

Is an interaction diagram that details how operations are carried out -- what messages are sent and when.

If a room is available for each day of the stay, make a reservation and send a confirmation.
Objects have behaviors and state. The state of an object depends on its current activity or condition. A **statechart diagram** shows the possible states of the object and the transitions that cause a change in state.
Activity Diagram

is essentially a fancy flowchart. It focuses on the flow of activities involved in a scenario.
About UML

- UML diagrams do not have a precise formal semantics
- Good: makes them versatile, usable for many purposes
- Bad: Poor basis for simulation, verification, code generation
- In practice, they are mostly used for documentation
- Exist number of UML tools that offer simulation, code generation
  - These must add many constructs, conventions, etc. to “plain UML”
  - E.g., Rational RT,
One View on Model-Based Development

- The following slides were presented by Mike Whalen at ICSE (Int. Conf. on Software Engineering) 2013.
Why We Model: Using MBD Effectively in Critical Domains

Mike Whalen
Program Director, UMSEC
University of Minnesota
How we Develop Software

Concept Formation

Requirements Specification

Design

Implementation

Object Code

Analysis

Test
Model-Based Development Tools

- Esterel Studio and SCADE Studio from Esterel Technologies
- Rhapsody from I-Logix
- Simulink and Stateflow from Mathworks Inc.
- Rose Real-Time from Rational
- I will focus on Statecharts and Dataflow notations.
How we **Will** Develop Software (in theory)
## Model-Based Development Examples

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Tools</th>
<th>Specified &amp; Autocoded</th>
<th>Benefits Claimed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airbus</td>
<td>A340</td>
<td>SCADE With Code Generator</td>
<td>• 70% Fly-by-wire Controls</td>
<td>• 20X Reduction in Errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 70% Automatic Flight Controls</td>
<td>• Reduced Time to Market</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 50% Display Computer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 40% Warning &amp; Maint Computer</td>
<td></td>
</tr>
<tr>
<td>Eurocopter</td>
<td>EC-155/135 Autopilot</td>
<td>SCADE With Code Generator</td>
<td>• 90 % of Autopilot</td>
<td>• 50% Reduction in Cycle Time</td>
</tr>
<tr>
<td>GE &amp; Lockheed Martin</td>
<td>FADEDC Engine Controls</td>
<td>ADI Beacon</td>
<td>• Not Stated</td>
<td>• Reduction in Errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 50% Reduction in Cycle Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Decreased Cost</td>
</tr>
<tr>
<td>Schneider Electric</td>
<td>Nuclear Power Plant Safety Control</td>
<td>SCADE With Code Generator</td>
<td>• 200,000 SLOC Auto Generated from 1,200 Design Views</td>
<td>• 8X Reduction in Errors while Complexity Increased 4x</td>
</tr>
<tr>
<td>US Spaceware</td>
<td>DCX Rocket</td>
<td>MATRIXx</td>
<td>• Not Stated</td>
<td>• 50-75% Reduction in Cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Reduced Schedule &amp; Risk</td>
</tr>
<tr>
<td>PSA</td>
<td>Electrical Management System</td>
<td>SCADE With Code Generator</td>
<td>• 50% SLOC Auto Generated</td>
<td>• 60% Reduction in Cycle Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 5X Reduction in Errors</td>
</tr>
<tr>
<td>CSEE Transport</td>
<td>Subway Signaling System</td>
<td>SCADE With Code Generator</td>
<td>• 80,000 C SLOC Auto Generated</td>
<td>• Improved Productivity from 20 to 300 SLOC/day</td>
</tr>
<tr>
<td>Honeywell Commercial Aviation Systems</td>
<td>Primus Epic Flight Control System</td>
<td>MATLAB Simulink</td>
<td>• 60% Automatic Flight Controls</td>
<td>• 5X Increase in Productivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• No Coding Errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Received FAA Certification</td>
</tr>
</tbody>
</table>

8 Slide courtesy of Steve Miller in “Proving the Shalls” © 2006 Rockwell Collins, Inc. All rights reserved.
Does Model-Based Development Scale?

```plaintext
System Developed Using MBD
- Flight Control
- Auto Pilot
- Fight Warning
- Cockpit Display
- Fuel Management
- Landing Gear
- Braking
- Steering
- Anti-Icing
- Electrical Load Management
```

**Airbus A380**

- **Length**: 239 ft 6 in
- **Wingspan**: 261 ft 10 in
- **Maximum Takeoff Weight**: 1,235,000 lbs
- **Passengers**: Up to 840
- **Range**: 9,383 miles

---

Slide courtesy of Steve Miller in “Proving the Shalls” © 2006 Rockwell Collins, Inc. All rights reserved.
…But it is not all roses

- Many MBD projects fail to meet their original goals of cost, productivity
  - These tend not to get as much publicity!
- Clear eyed understanding of why you model and what you expect is necessary
What are your models for?

- Possible to use MBD for many different purposes:
  - Requirements
  - Design
  - Simulation
  - Visualization
  - Testing
    - Test Generation
    - Test Oracle
  - Formal Verification
  - Code Generation
    - Complete implementation
    - Code skeleton
  - Prototyping
  - Communication with Customer

You must understand, up front, what you expect to do with models in order to successfully adopt MBD.

Major opportunity for improvement in V&V
MBD Models as Requirements

- Are MBD models requirements?

- Notations in this talk are executable; good at describing how system works
• Lots of design detail
• Difficult to see “full system” behavior.
• Straightforward to generate code
The Most Important Issue for Successful Adoption of MBD

Do the Domain-Specific Notations provide a natural representation for your problem?

- Block diagrams are *very natural* for control problems
- Statecharts are *very natural* for description of system modes & mode transitions
- Both block diagrams and statecharts are *very unnatural* for representing complex data structures
- Neither notation naturally supports iteration or recursion
  - It can be “faked”, but not well
Just...No.

Stateflow model of Tetris game (included in the Stateflow Demo models from the Mathworks!).

Diagram is essentially a control-flow graph of a program that implements tetris.

*Much* harder to read and modify than an equivalent program.
Tools Matter

- Often notations are much more cumbersome to use than text
  - No diff / merge capabilities
  - Adding information requires many clicks
- Expressible != Easy
- Anecdote: Simulink vs. SCADE at Rockwell Collins in 2006
  - SCADE had formal pedigree, strong analysis
    - But tools kept crashing on our Windows boxes
  - Simulink had better tools and better salespeople
About SysML

- The following slides are by Marco DiNatale, retis.ssup.it/~marco/teaching/embeddedsystems/lessons
What is SysML

• A graphical modelling language developed in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233a
• Supports the specification, analysis, design, verification, and validation of systems that include *hardware*, *software*, *data*, *personnel*, *procedures*, and *facilities*
• *Is* a visual modeling language that provides
  – *Semantics* = meaning, connected to a metamodel (rules governing the creation and the structure of models)
  – *Notation* = representation of meaning, graphical or textual
• *Is not* a methodology or a tool (SysML is methodology and tool independent)
Metamodel, model and charts/views

Model(s)

System

Views

Built according to the rules of the metamodel
SysML vs UML

• UML is a general-purpose graphical modeling language aimed at Software Engineers
• Diagrams not used
  – Object diagram,
  – Deployment diagram,
  – Component diagram,
  – Communication diagram,
  – Timing diagram and
  – Interaction overview diagram
• Diagrams from UML
  – Class diagram (Block Definition Diagram - Class → Block)
  – Package diagram,
  – Composite Structure diagram (Internal Block Diagram)
  – State Machine Diagram
  – Activity Diagram
  – Use Case Diagram
  – Sequence Diagram
SysML vs UML

• In addition, SysML adds some new diagrams and constructs
  – Parametric diagram,
  – Requirement diagram
  – Flow ports,
  – Flow specifications
  – Item flows.
  – Allocation
SysML vs UML

• Includes UML4SysML: a UML Profile that represents a subset of UML 2 with extensions
• Supports model and data interchange via XML Metadata Interchange (XMI®) and the evolving AP233 standard (in-process)

SysML Extensions

• Blocks
• Item flows
• Value properties
• Allocations
• Requirements
• Parametrics
• Continuous flows
Available diagrams

- SysML Diagram
  - Behavior Diagram
    - Activity Diagrams
      - Sequence Diagram
    - Sequence Diagram
  - Requirement Diagram
    - Block Definition Diagram
    - Internal Block Diagram
  - Structure Diagram
    - Package Diagram
      - Parametric Diagram
Examples of diagrams

1. Structure

2. Behavior

interaction
state
machine
activity/
function

3. Requirements

4. Parametrics
Blocks and Compartments

```
modeBtn : null

<<block>>
DigitalWatch

constraints

operations

parts

+ display : Watch_Display

references
values
properties

+counter : Timestamp

lightBtn : null
```
Blocks and Compartments

Property is a structural feature of a block

Part property
aka. part (typed by a block)
Usage of a block in the context of the enclosing (composite) block
Example - right-front:wheel
The BDD cannot define completely the communication dependencies and the composition structure (no topology)
SysML Ports

• Specify interaction points on blocks and parts
• Integrate behavior with structure
• Syntax: portName:TypeName
• Kinds of ports
  – **Standard (UML) Port**: Operation oriented – for SW components
    • Specifies a set of required or provided operations and/or signals
    • **Typed by a UML interface**
  – **Flow Port**: Used for signals and physical flows
    • Specifies what can flow in or out of block/part
    • **Typed by a block, value type, or flow specification**
    • Atomic, non-atomic, and conjugate variations
• Standard Port and Flow Port Support Different Interface Concepts
Port notation

**Standard Port**

provided interface
(provides the operations)

required interface
(calls the operations)

**Flow Port**

item flow
Parametric Diagrams

- Support for engineering analysis (e.g., performance, reliability) on design models
- Parametric diagram represents the usage of the constraints in an analysis context
- May be used for identification of critical performance properties
- Computational engine is provided by applicable analysis tool and not by SysML
Example of use of a parametric diagram

par [Block] Straight Line Vehicle Dynamics [ Value Bindings ]

- \( v.b.abs.m1.duty\ cycle : \% \)
- \( v.c.t.friction : N \)
- \( v.b.r.braking\ force : N \)
- \( v.mass : Kg \)

**e1: Braking Force Equation**
\[ f = (t_f \times b_f) \times (1 - t_l) \]

**e2: Acceleration Equation**
\[ a = m/\text{sec}^2 \quad \{ f = m \times a \} \]

**e3: Velocity Equation**
\[ v = m/\text{sec} \quad \{ a = d\v/\text{dt} \} \]

**e4: Distance Equation**
\[ x = d/\text{sec} \]

- \( x : m \)
- \( v : m/\text{sec} \)
- \( t : \text{sec} \)

- \( v.position : m \)
- \( \text{clk.time} : \text{sec} \)