Model-Based Development Tutorial
Mike Whalen
Program Director, UMSEC
University of Minnesota

Outline of Presentation

Introduction
Modeling and Model-Based Development
Why use Model-Based Development?
Requirements
Design
Implementation: Code Generation
Verification and Validation
Pitfalls

How we Develop Software

Test
What is Model-Based Development?

- Properties
- Visualization
- Analysis
- Testing
- Prototyping
- Code/test generation

Model-Based Development Tools

- Commercial Products
  - Esterel Studio and SCADE Studio from Esterel Technologies
  - Rhapsody from I-Logix
  - Simulink and Stateflow from Mathworks Inc.
  - Rose Real-Time from Rational
  - Etc. Etc.

How we Will Develop Software (in theory)
Model-Based Development Examples

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Tools</th>
<th>Benefits Claimed</th>
</tr>
</thead>
</table>
| Airbus           | A340                     | SCADE With Code Generator | • 70% Fly-by-wire Controls  
• 70% Automatic Flight Controls  
• 50% Display Computer  
• 40% Warning & Maint Computer  
• 20X Reduction in Errors  
• Reduced Time to Market |
| UK & Australia   | Electric                  | SCADE With Code Generator | • 90% of Autopilot  
• 50% Reduction in Cycle Time  
• Complexity increased by 4x |
| European Space   | Airbus                    | SCADE With Code Generator | • 30% SLOC Auto Generated  
• 5X Reduction in Errors  
• 60% Reduction in Cycle Time  
• 50-75% Reduction in Cost  
• Reduced Schedule & Risk |
| Honeywell        | Commercial Aviation       | MATLAB Simulink        | • 60% Automatic Flight Controls  
• 5X Increase in Productivity  
• No Coding Errors  
• Received FAA Certification |

Does Model-Based Development Scale?

Airbus A380

- Length: 239 ft 6 in
- Wingspan: 264 ft 10 in
- Maximum Takeoff Weight: 1,235,000 lbs
- Passengers: Up to 840
- Range: 9,383 miles

…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
Process Did not Change Much

A Personal Anecdote
- Part of two large projects using Model-Based Development
  - Same company, similar quality developers
  - One great success
    - Significant cost reductions
    - Improvement in quality
    - Excellent customer satisfaction
  - One great failure
    - Huge cost overruns
    - Models considered less useful than code
    - Group abandoned MBD

Outline of Presentation
- Introduction
  - Modeling and Model-Based Development
    - Why use Model-Based Development?
      - Requirements
      - Design
      - Implementation: Code Generation
      - Verification and Validation
  - Pitfalls
Why Model?

- **Simplify the problem**
  - Consider design alternatives
  - Better communication with customer
  - Better communication with other developers
  - Better documentation for maintenance
- **Analyze the problem**
  - Simulations
  - Testing
  - Proof

What makes a “Good” Model?

- Predictive
- Sound
- Relevant
- Simple
- Focused

Definition of Model [Eykhof]: “a representation of the essential aspects of an existing system (or system to be constructed) which presents knowledge of that system in a usable form”

Kinds of Models

- Statistical models for failure analysis
- Performance models for scaling
- Schedulability models for real-time systems
- CAD models for manufacturing
- Behavioral models of software and systems
  - These are MBD models
  - Source code is also a model (in the mathematical sense)
What is Model-Based Development (Again)?

Model-Based Development is a methodology that uses a set of domain-specific notations (DSNs) to simplify the design, development, and/or verification of software and software-intensive systems.

- DSNs that I will talk about today are Statecharts and block diagrams
  - Basis of: Simulink/Stateflow, SCADE, Rhapsody, MATRIXx,
  - Portion of UML: UML Statecharts

Statecharts

- Popular notation for implementing complex state machines
- Proposed by David Harel (1987)
- Statecharts = state diagrams + depth (hierarchy) + orthogonality (parallelism) + broadcast-communication

Block Diagrams

- Define input/output function
- Use dataflow ordering (arrow direction) to sequence computation
- Very familiar to control engineers; matches how they tend to think
- MBD tools often contain toolboxes for specific control problems
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.
Outline of Presentation

- Introduction
- Modeling and Model-Based Development
- Why use Model-Based Development?
  - Requirements
  - Design
  - Implementation: Code Generation
  - Verification and Validation
- Pitfalls

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

Are MBD Models Requirements?

- Usually: NO

- DSNs in this talk are executable; good at describing how system works
  - That's why you can generate code!
Microwave: Keypad Processing

Microwave: Mode Logic

What are Requirements Models?

MBD models can be used to express requirements, but they are often verbose.
Why Write Requirements as Models?

- Use models as test oracles
  - Run in parallel code to check code compliance to requirements
- Use models for formal analysis
  - Do proof that design model complies with requirements
- More on this later
- However, requirements models are quite verbose
  - Imagine that you had 100s of requirements
    - One project: analyzed 573 requirements for Display Window Manager.
    - Textual notations are much more concise
Design Models

- Central activity of MBD is design
- Often allows for much more natural representation of problem than code
- Choosing your Scope
  - What level of abstraction am I pursuing?
  - Microwave example
- Design using MBD languages is an art, just like any other software design
  - Need constructs for modularity, scoping, just like programming languages

(Lack of) Support for Modularity

Until most recent release of Stateflow, unable to include generic instance of sub-chart!

Outline of Presentation

Introduction
Modeling and Model-Based Development
Why use Model-Based Development?
  - Requirements
  - Design
Implementation: Code Generation
  - Verification and Validation
Pitfalls
**Code Generation**

- Code generation is very important to the value proposition of MBD
- **Pluses:**
  - No need to "write system twice"
  - Ensures that model and code are kept consistent
  - If code generator is trustworthy, means that analysis done on the model will match the code
  - Modulo timing...
- **Minuses**
  - Code generator efficiency becomes critical concern in notation and tool choice
  - Implementation efficiency becomes central to model design
  - Can significantly obscure model meaning
  - Must be careful with test-case generation from model

**Limits of Code Generation**

- "Normal" code generator views execution as input/output function
  - For Rhapsody, these are external events
  - For Simulink, you refresh all inputs each "computational step"
- Something has to perform actual interaction with hardware
  - Can be placed in model, but usually destroys much of simulation/analysis benefit

**Kinds of Code Generation**

- Not necessarily yes/no question
  - Can generate partial implementations
- At Rockwell Collins:
  - Used Simulink/Stateflow with code generation
  - Efficiency important, but not paramount
  - WCET often most important concern
  - Time critical operations and hardware I/O written directly in C
  - Viewed as "black box" to the model
  - In Simulink, these are called S-Functions
  - Can be swapped in/out to optimized/debug builds
  - Additional verification (though compartmentalized)
Microwave Demo

Outline of Presentation

Introduction
- Modeling and Model-Based Development
- Why use Model-Based Development?
  - Requirements
  - Design
  - Implementation: Code Generation
  - Verification and Validation
- Pitfalls

How we Develop Software

- Concept Formation
- Requirements Specification
- Design
- Integration
- Unit Test
- Analysis
- Object Code
Why Do We Test?

- Verification: Conformance to Requirements
- Scalability
- Memory Safety
- Performance
- Reliability
- Documentation
- Numeric Precision
- Code Understanding
- Arithmetic Overflow
- Security
- Deadlock Detection
- Discovery
- Usability
- Software Testing is Hard

Software Testing is Hard
- Inherently

- System

Examples of Formal Methods

The Problem with Testing

- ...is that it only samples a set of possible behaviors
- Unlike physical systems, (where many engineers gained their experience), most software systems are discontinuous
- There is no sound basis for extrapolating from tested to untested cases
- So we need to consider all possible states of the system

Even small systems have trillions and trillions of possible states
MBD Is a V&V-Enabling Technology

- Strong simulation and analysis capabilities built into most tools
  - Demo: Stateflow Elevator
    (Help: Stateflow/Demos/Large-Scale Modeling/Modeling an Elevator System)
  - Even stronger simulation capabilities in external tools
    - Demo: Reactis step simulation with Microwave
- Allows straightforward “Build a little, test a little” philosophy
  - Consistent with spiral development philosophy
Integrating V&V Into MBD Cycle

- Reuse
- Requirements Elicitation
- Autotest
- Modeling
- Simulation
- Autocode
- Automated Analysis

- Reduces Cost of Testing
- Enables More Testing
- 10% Clear Specifications
- Improves Communication
- 10% Easy Validation
- Finds Errors Early
- Cheaper Than Manual Analysis
- Finds the Really Hard Errors

- 10% - 20%

Automation & Rigor

- Many tools automate test execution: jUnit, nUnit, Selenium, SOATest, ...
- We need more:
  - Automate test generation as well as execution
  - Get better coverage than is possible with manual testing
  - Exhaustive coverage!

Model-Driven Test Generation (v1)

- Source Code
- Compiler
- Object Code
- Model must match source code exactly for tests to pass
- Coverage Metric
- Test Case Generator
- Generated Tests
- MBD Model
- Possible to generate test cases that satisfy very rigorous structural coverage metrics
Model-Driven Test Generation (v2)

- MBD Model
- Code Generator + Compiler
- Object Code
- Model should match source code exactly
- Test Case Generator
- Generated Tests
- Coverage Metric

Automated Test Generation

- MBD Model
- Compiler
- Object Code
- Generated Tests
- Oracle
- Where does Oracle come from?
- What is a good oracle?

Demo: Reactis with Microwave
What is a Good Oracle?

Encode Requirements

If the mode is COOKING, then the microwave door shall be CLOSED

SPEC AG(mode = COOKING -> door_closed);

assert (!(mode == COOKING) || door_closed);

...and Use Them as Oracle
Static Analysis and Model Checking

Property

Oracle

True

False:
Test Case

Analysis Tool

Model Checking

Testing Checks Only the Values We Select

Even Small Systems Have Trillions (of Trillions) of Possible Tests!

Finds every exception to the property being checked!

How Can This Work?

- We said that even small programs have trillions of trillions of states
- Impossible to enumerate!
- So, how can we possibly automatically check requirements on programs?
Symbolic Representations

- Big idea: represent sets of system states or paths symbolically as Boolean formulas
  - Compact representation for large sets of states
  - Allows large state spaces to be explored
  - NASA Docking Example: 100 Billion states searched in one minute
  - We have analyzed models with $10^{30}$ states in a few hours
- However, model checking does not scale as well as symbolic analysis
  - Static Analysis: millions of SLOC
  - Model Checking: Hundreds or thousands of SLOC
- For system level, need compositional approaches

A Fun Example…

- Empty Sudoku board has $9^9 = 362,880$ possible states
  - After filling in “given numbers” for a Sudoku board (usually 19-22 squares filled), fewer states: $9^{18} = 9.56 	imes 10^{16}$
  - Valid board with “given numbers” has exactly 1 solution
- What are Sudoku requirements?
  - Must have exactly one of 1-9 in each row, column, and 3x3 square.
  - Equivalently, a “bad” non-solution will have two (or no) elements along one of these vectors
- Can we trick a model checker into solving Sudoku?
  - State that all solutions for a set of givens are “bad”
  - Model checker will demonstrate that this property is false and provide a counterexample (solution)
  - This is how test-case generation using a model checker works

Sudoku encoding in SAL

```
sudoku : CONTEXT =
BEGIN
  BLOCK : TYPE = [ 0 .. 2 ] ;
  BLOCK_RANGE : TYPE = [ 1 .. 3 ] ;
  RANGE : TYPE = [ 1 .. 9 ] ;
  BOARD : TYPE = array RANGE of array RANGE of RANGE ;
sudoku : MODULE =
BEGIN
  INPUT b : BOARD
  OUTPUT good : BOOLEAN
  DEFINITION
  good =
    % good columns
    (FORALL ( x: RANGE, y1: RANGE, y2: RANGE ) :
      ( y1 /= y2 ) => ( b[x][y1] /= b[x][y2] ) ) AND
    % good rows
    (FORALL ( y: RANGE, x1: RANGE, x2: RANGE ) :
      ( x1 /= x2 ) => ( b[x1][y] /= b[x2][y] ) ) AND
    % good blocks
    (FORALL ( block: BLOCK, block_range: BLOCK_RANGE, x1: BLOCK_RANGE, y1: BLOCK_RANGE ) :
      ( NOT (( x1 = x2 ) AND ( y1 = y2 )) ) =>
      b[3*block + x1][3*block_range + y1] /=
      b[3*block + x2][3*block_range + y2] )
END ;
```
**Sudoku Instance**

- Generate a board that meets the Sudoku constraints:
  ```plaintext
test_th: THEOREM sudoku |- (not good);
```

- Solve a specific 'hard' Sudoku according to Wikipedia:
  ```plaintext
test_bf_hard: THEOREM sudoku |- G
```

---

**Model Checking Process**

Does the system have property X?

Yes!

Engineer

Automatic Translation

SMV Spec.

Properties

Automated Check

SMV

Properties

---

No!

Counter Example

Engineer

Automatic Translation

SMV Spec.

Properties

Automatic Translation

SMV Properties
Demo: Microwave Model Checking

6.8 x 10^21 Reachable States

Mode Controller A Modeled in Simulink Translated to NuSMV

Counterexample Found in Less than Two Minutes
Found 27 Errors

Example Requirement
Mode A1 => Mode B1

FCS 5000 Flight Control Mode Logic

Mode Controller B Modeled in Simulink Translated to NuSMV

Over 10^37 Reachable States

Example Requirement:
Drive the Maximum Number of Display Units
Given the Available Graphics Processors

Counterexample Found in 5 Seconds
Checked 573 Properties - Found and Corrected 98 Errors in Early Design Models

ADGS 2100 Adaptive Display and Guidance System

Modeled in Simulink Translated to NuSMV

4,295 Subsystems
16,117 Simulink Blocks

Example Requirement:
Drive the Maximum Number of Display Units
Given the Available Graphics Processors

Counterexample Found in 5 Seconds
Checked 573 Properties - Found and Corrected 98 Errors in Early Design Models

Slide © Rockwell Collins, 2008
### Tech Transfer & Process Improvements

- **Iteration 1**
  - Simulink R14 Model
  - SCADe Model
  - NuSMV Model

- **Iteration 2**
  - Simulink R14 Model
  - Reactis Model
  - NuSMV Model

- **Iteration 3**
  - Simulink R14 Model
  - Reactis Model
  - NuSMV Model

**Translation Time:**
- Iteration 1: 1-4 Hours
- Iteration 2: 10 Minutes
- Iteration 3: 10 Minutes

**Turnaround:**
- Iteration 1: 1 Day to 1 Week
- Iteration 2: 3 Hours to 2 Days
- Iteration 3: 10 Minutes

---

### CerTA FCS Phase I

- **Sponsored by ARL**
  - Wright Patterson VA Directorate
- **Compare FM & Testing**
  - Testing team & FM team
- **Lockheed Martin UAV**
  - Adaptive Flight Control System
  - Redundancy Management Logic
  - Modeled in Simulink
  - Translated to NuSMV model checker

**Subsystems**
- Blocks
- Charts / Transitions / TT Cells
- Reachable State Space

**Properties**
- Syntax checking
- Behavior checking
- Finite state automata
- Model checking

**Testing Results**
- Total: 23/109
- Model-Checking: 40%
- Testing: 60%
- Effort (% total): 60%
- Errors Found: 12

---

### Outline of Presentation

- **Introduction**
- **Modeling and Model-Based Development**
  - Why use Model-Based Development?
  - Requirements
  - Design
  - Implementation: Code Generation
  - Verification and Validation
- **Pitfalls**
Problem 1
Believing Testing Can be Eliminated

Testing will always be a crucial (and costly) component

How we Develop Software

Concept
Formation

Requirements
Specification

Design

Implementation

Analysis

System

Testing Does not go Away

Concept
Formation

Requirements
Specification

Properties

System

Extensive Testing (MC/DC)

Implementation
It Simply Moves

Do it the Right Way

Remedy
- Be honest about the capabilities of model-based development and formal methods
  - Done right, provides outstanding requirements, models, analysis, etc., etc.
  - May greatly reduce the effort spent in testing
Problem 2
Believing the Model is Everything

The model is never enough

Modeling Frenzy

How do we know the model is "right"?

Do it the Right Way
Remedies

- Recognize the Role of Software Requirements
  - The model is not everything
- Development Methods for Model-Based Development Badly Needed
- Model-Based Software Development Process
- Develop Tools and Techniques for Model, Properties, and Requirements Management
- Develop Inspection Checklists and Style Guidelines for Models

Problem 3
Trusting Verification

To really mess things up, you need formal verification

Property or Model: Who is Right?

AG((Is_This_Side_Active & Onside_FD_On) -> Mode_Annunciations_On)

AG!(Onside_FD_On -> Mode_Annunciations_On)

AG((Is_This_Side_Active & Onside_FD_On) -> Mode_Annunciations_On)

AG!(Mode_Annunciations_On -> AX((Is_This_Side_Active & Onside_FD_On) -> Mode_Annunciations_On)))
Remedies

- Develop techniques to determine adequacy of model and property set
  - How do we know they are any “good”
- Techniques for management of invariants
  - How do we validate the assumptions we make
- Methodology and guidance badly needed
  - “Tools with training wheels”
  - “Verification for Dummies”

All we need is one high-profile verified system to fail spectacularly to set us back a decade or more

Problem 4
Believing One Tool Will Be Enough

To be effective, we need a suite of notations and analysis tools (and the ability to continually integrate new ones)
Remedies

- Next generation tools must allow easy extension and modification of notations to meet domain specific needs
- They must allow easy construction of high-quality translations from modeling notations to analysis tools
- They also must enable controlled reuse of tool infrastructure to make tool extensions cost effective

Problem 5
Using Models Where They Don’t Fit

If MBD Notation Doesn’t Provide A Better Representation of Your Problem than Code, You’re Wasting Your Time.
Remedies

- Perform honest assessment of where MBD notations can be used
  - They do not do everything
  - Recursive data structures are especially difficult to model.
  - Do not use models where they do not fit.
- Create a partitioning strategy between models and code for applications that contain both complex mode logic and complex data.

Conclusions: A Personal Anecdote

- Part of two large projects using Model-Based Development
  - Same company, similar quality developers
  - One great success
    - Problem was mix of control behavior and mode machines
    - Strong simulation capability “out of the box”
    - Good requirements
    - Hardware dependencies factored out
  - One great failure
    - Problem had modal behavior + Complex data types (lists, trees)
    - Hardware dependencies built in
    - No Desktop Simulation Possible
    - Requirements were mediocre

Conclusions

- MBD can significantly improve developer productivity, cost, schedule, and quality
- …or it can make your life miserable
- The important thing is to know why you’re doing it!
  - Know the limitations of what can be modeled using the DSNs
  - Know which capabilities you hope to use
  - Design and quality of models depends on this
- V & V receives the largest benefit of the MBD approach
  - Mature tools for test-case generation
  - Starting to see model checking built into commercial tools: SCADE Verifier, Simulink Design Verifier
- There are many other things to discuss! Versioning, diff, semantics, tool costs, training, structuring, vendor “lock in”
Questions?

Medical Cyber-Physical Systems
Improving patient treatment by coordinated systems of medical devices

Research directions:
• Medical device interoperability
• High-confidence development
  - Model-driven design
  - V&V, regulatory approval
Supported by NSF CNS-1035715
http://mg.cis.upenn.edu/MDCPS/

Participants
• University of Pennsylvania
• U Penn Hospital System
• University of Minnesota
• CIMIT/MGH

Improving patient treatment by coordinated systems of medical devices
• Smart alarms and decision support
  - Reduction of irrelevant alarms for CABG patients
  - Based on aggregation of multiple vital signs and fuzzy logic
  - Ongoing research: Prevention of vasospasm in neuro-ICU patients

Model driven development and assurance cases
• Model driven development:
  - Microcontroller, case study
  - Formal verification, regulatory approval
  - Assurance case construction reflects development process structure
  - Applied in pacemaker, ICA pump

Participant: University of Pennsylvania

Smart alarm systems
• Reduction of irrelevant alarms for CABG patients
• Based on aggregation of multiple vital signs and fuzzy logic
• Ongoing research: Prevention of vasospasm in neuro-ICU patients

Networked Blood Glucose Control System
Safety-critical, closed-loop MDS
Research issues:
• Identifying time risks and hazards
• Mitigation strategies
• Validation
• Control design

References

Copyright 2011, University of Pennsylvania
Problem 6
Not Understanding the Semantics of Models

Models can look the same across tools and behave differently. Semantics affect model understanding and analysis.

Figure from: Michelle L. Crane and Juergen Dingel, UML vs. Classical vs. Rhapsody: Not All Models are Created Equal, Proceedings of MoDELS2005, Montego Bay, Jamaica, October, 2005
Infinite Loops (Rhapsody)

- Example of GEN leading to infinite loop
- C1 queues message for C2 which queues message for C1 which ...

Strange Charts (Stateflow)

- Infinite Event Loop
- Early Return Logic
- Infinite Junction Loops

Static Analysis: Finding “Mechanical” Errors, i.e., Fixed Oracle

- Security vulnerabilities
  - Buffer overruns, unvalidated input...
- Memory errors
  - Null dereference, uninitialized data...
- Resource leaks
  - Memory, OS resources...
- Violations of API or framework rules
  - e.g. Windows device drivers; real time libraries; GUI frameworks
- Exceptions
  - Arithmetic/library/user-defined
- Race conditions
  - Two threads access the same data without synchronization
Static Analysis: Approximations

- Static Analysis tools approximate the reachable states of a program with something that is easier to compute
- A sound abstraction includes all of the behaviors of the original program

Sound abstractions can lead to a large number of false alarms:
- errors that can’t occur in the real program
- Most tools use unsound approximations to reduce the number of false alarms, but this means real errors can be missed.

Model Checking: Checking Requirements, i.e., Rich Oracles

- Symbolic representation of program
- Possible to encode rich set of requirements
- Temporal logic (TL) is often used for reasoning
  - Describes the evolution of the model from some current state

For example, in a microwave:

\[ G(\text{mode} = \text{COOKING} \Rightarrow \text{door\_closed}) \]
\[ G(\text{mode} = \text{COOKING} \Rightarrow F(\text{mode} = \text{COOKING})) \]
\[ \text{mode} = \text{SETUP} \]
\[ G(\text{mode} = \text{COOKING} \Rightarrow X(\text{stop\_pressed} \Rightarrow \text{mode} = \text{COOKING})) \]
## Strengths and Weaknesses of Specification Styles

<table>
<thead>
<tr>
<th></th>
<th>Natural Language</th>
<th>Property Based</th>
<th>Constructive Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambiguity</td>
<td>Likely</td>
<td>Eliminated</td>
<td>Eliminated</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>Likely</td>
<td>Possible</td>
<td>Eliminated</td>
</tr>
<tr>
<td>Incompleteness</td>
<td>Likely</td>
<td>Possible</td>
<td>Eliminated</td>
</tr>
<tr>
<td>Implementation Bias</td>
<td>Possible</td>
<td>Possible</td>
<td>Likely</td>
</tr>
</tbody>
</table>

Early ➜ Life Cycle ➜ Late