Generative Programming & Component Engineering

Keynote Address Presentation
Application of Model Based Development to Flexible Code Generation

Gary Shubert
Lockheed Martin Space Systems Company

Generative Programming & Component Engineering Conference
October 22-23, 2011
Biography

- Gary Shubert is currently a Senior Software Manager for Lockheed Martin Space Systems Company and leads systems and software model based engineering process development and demonstration activities for the Military Support Program - System Evolution Test Bed and for the Space Vehicle Integration Laboratory.
- Gary joined Lockheed Martin in 1989 and has served in roles as an embedded flight software developer, flight software architect, and flight software manager, for launch vehicle, uncrewed and crewed spacecraft systems. Gary served as the software manager for the Orion proposal activities and as the flight software manager for the Orion development program. Gary has served as an avionics lead, software lead, and software architect, for the Space Based Radar, Reusable Space Transportation System, Evolved Expendable Launch Vehicle (Atlas V), and Space Common Data Link programs, and as a flight software engineer for the Titan IV and Advanced Interceptor Technologies programs.
- Gary has been a practitioner and a champion of model based software engineering and transformational automated code generation since 1989. He is the architect for three generations of Lockheed Martin developed automated code generators used on multiple production programs. Gary is an instructor for Lockheed Martin’s Object Centric Modeling and Ada programming language courses.
- His background prior to Lockheed Martin includes being a flight software engineer in the Control Systems Advanced Development department of Northrop Aircraft, and an independent verification and validation engineer and ground software engineer for the Mission Software department of TRW Defense Systems.
- He holds a bachelor’s degree in Aeronautical/Astronautical Engineering from the University of Illinois, and a master’s degree in Aeronautical/Astronautical Engineering from the University of Southern California.
Abstract

• This address will:
  – Present the authors views and perspectives on the past, present and future use of model based development techniques to enable the automated generation of source code and other forms of programming.
  – Discuss past and present use of model based development and automated code generation at Lockheed Martin, with special emphasis on NASA’s Orion Multi-Purpose Crew Vehicle Program.
  – Discuss the advantages and disadvantages, associated with the current state of the practice techniques and tools used to automatically generate source code from general purpose and domain specific models.
  – Discuss the obstacles and enablers associated with achieving the desired future state of complete and efficient automated generation of programming through transformation of general purpose and domain specific models.
The Engineers Dilemma

• Given a set of requirements:
  – Are they?
    • Correct
    • Complete
    • Consistent
  – How do I?
    • Program a processor to provide the required behavior.
    • Program an FPGA to provide the required behavior.
    • Design an ASIC to provide the required behavior.
The Engineers Dilemma

It Can Be a Long & Difficult Road from a “Shall” to a “Solution”
Evolution of The Analysis & Design Processes

• Analyze the requirements to understand the required functionality and behavior, and to capture the intended design to a level of detail sufficient to initiate implementation
  – Write text descriptions to document the functionality, behavior & design
    • Documents contain “sufficient” information to start implementation
    • Write code to implement the design
  – Create diagrams to depict the functionality, behavior & design
    • Emphasis: Structure, Interfaces, and Logic
    • Evolution: Pictures to Models, Hand Code to Automated Code Extraction
    • Techniques: Structured Analysis & Design, Object Oriented Analysis & Design
  – Create formalized specifications of the functionality, behavior & design
    • Emphasis: Structure, Interfaces, Functionality and Behavior; Completeness, Consistency, Analysis, Execution
    • Evolution: Automated Code Extraction to Automated Code Transformation
    • Techniques: Object Centric Modeling with SysML/UML, Domain Specific Modeling, Requirements Specification Languages

Increasing Requirements Complexity has Necessitated Better Analysis & Design Processes
Evolution of The Implementation Processes

• Interpret the analysis and design products to implement a product that provides the required functionality & behavior
  – Analyze the text descriptions of the functionality & behavior, and create low level programming language statements that “program” the device.
  – Analyze the diagrams / figures that depict the functionality, behavior & design, and create / extract high order programming language statements that “program” the device.
  – Apply rules to transform the formalized specifications of the functionality, behavior & design, to generate high order programming language statements that “program” the device.
    • Processor - Source Code (Ada, C++, JAVA, etc.)
    • FPGA - VHDL/Verilog HDL, ESL/HLS, C/C++, Simulink, SystemVerilog
    • ASIC - VHDL/Verilog HDL
Modeling & Automated Code Generation

• **Advantages:**
  – Productivity Increases up to 40%
  – Defect & Error Reduction up to 50%
  – Improved Consistency
  – Improved Ratio of Knowledge Capture Effort to Coding Effort
  – Improved Documentation and Test Automation

• **Disadvantages:**
  – Source Code Bloat & Execution Performance Impacts
  – Some Code Generators Impose Vendor Specific Architectures
  – Some Code Generators Are Difficult to Configure

The Benefits Have Been Demonstrated, Improvements Are Needed
Abstraction & Programming Effort

Modeling Is the Next Generation Evolution of Programming Abstraction
Abstraction & Automated Code Generation

Automated Code Generation Transforms Modeling Abstractions into Programming

Block Diagram ➔ Stateflow Diagram ➔ Class Diagram ➔ Statechart Diagram ➔ Entity Relationship Diagram ➔ User Interface Display

VHDL ➔ ESL/HLS ➔ Source Code ➔ Scripts ➔ Data Tables ➔ HTML

Ada, C++, Java, Handle C, etc.
Obstacles to Automated Code Generation

- Skepticism
- Risk Aversion
- Resistance to Change
- Skill Currency
- Programmer Lobbyists
- Quality of Generated Programming Products
- Ability to Control Code Generation Results
- Non-Standard Action Language for UML

The Majority of the Obstacles Are Cultural
Sample Historical Use at Lockheed Martin

- Titan IVB Advanced Avionics: LM Space Systems - Flight Software
  - Structured Analysis & Design Models, Ada Source Code
- EELV / Atlas V: LM Space Systems - Flight Software
  - Object Oriented Analysis & Design Models, Ada Source Code
- SCDL, OSP IRAD: LM Space Systems - Flight Software
  - Object Oriented Analysis & Design Models, C++ Source Code
- OSP Pad Abort Demonstrator: LM Space Systems - Flight Software
  - Object Oriented Analysis & Design Models, Ada Source Code
  - Mathematical Models, Ada Source Code
- Joint Strike Fighter: LM Aeronautics - Flight Control Software
  - Object Oriented & Mathematical Models, C/C++ Source Code
- XSS-11: LM Space Systems - Flight Software
  - Mathematical Models, C/C++ Source Code
- Orion Pad Abort Demonstrator: Flight Software
  - Object Oriented & Mathematical Models, C/C++ Source Code
- Orion Crew Exploration Vehicle: Ground, Flight & Simulation Software
  - Object Oriented & Mathematical Models, C/C++ Source Code, Handle C, VHDL
  - GUI Models, C/C++, Java; Data Base Models, Java
- Joint Strike Fighter: LM Aeronautics - Systems Management Software
  - Object Oriented & Mathematical Models, C/C++ Source Code
- HW / SW Co-Design Pilot – LM Electronic Systems – Communications HW
  - SysML & SystemC Models, VHDL, Altera HW / ModelSim
- MBSS IRAD - LM Space Systems: Software
  - Object Oriented Models using ALF, C++ Source Code

Automated Code Generation Has Been Used at LM for Many Years
Auto Code – That Will Never Fly Era

• Titan IVB Advanced Avionics – Booster Flight Software
  - Structured Analysis & Design Models
  - Ada 83 Source Code
    - ~20 K-SLOC
    - Real-Time Embedded
    - Interrupt Driven
    - Flight / Safety Critical
  - 1750A Single String Processor Architecture
Auto Code – That Will Never Fly Era

- EELV / Atlas V – Booster & Upper stage Flight Software
  - Object Oriented Analysis & Design Models
  - Ada 95 Source Code
    - ~30 K-SLOC
    - Real-Time Embedded
    - Deterministically Scheduled
    - Flight / Safety Critical
  - 1750A Fault Tolerant Processor Architecture
    - Atlas V MRO Launch
JSF Family Of Aircraft (F-35 A/B/C)

MBSD is done at 3 different levels
- Architecture/Operational Level
- System Design Level
- Implementation

Modeling & Auto Code Generation
Tools:
- Rhapsody
- MATLAB/Simulink
- Others

Advantages:
- Same code in Simulations and Operational Flight
  - No surprises for pilot
- Reuse of design patterns/common models
  - Boost Affordability
  - 3 Aircrafts designed simultaneously

Conventional Take-Off & Landing (CTOL) F-35A
Larger Wing

Carrier Variant (CV) F-35C
Roll Nozzle
3-Bearing Swivel Duct

Short Take-Off Vertical Landing (STOVL) F-35B
Lift Fan

Larger Wing
3-Bearing Swivel Duct
Roll Nozzle
Lift Fan
Auto Code – Really Does Fly Era

• Joint Strike Fighter – Aircraft Flight & Simulation SW
  – Object Oriented & Mathematical Models
  – C/C++ Source Code
    • Multi M-SLOC
    • Real-Time Embedded
    • Deterministically Scheduled
    • Flight / Safety Critical
  – PPC Fault Tolerant Processors
    • F-35 First Vertical Landing
    • http://www.youtube.com/watch?feature=player_profilepage&v=r-cM3wPpCPw
Auto Code – Really Does Fly Era

• Orion Pad Abort Demonstrator – Spacecraft Flight SW
  – Object Oriented & Mathematical Models
  – C/C++ Source Code
    • ~100 K-SLOC
    • Real-Time Embedded
    • Deterministically Scheduled
    • Flight / Safety Critical
  – PPC Fault Tolerant Processors
    • Orion Pad Abort Test Flight 1 (PA-1)
Orion Pad Abort Test Flight 1

Credit: U.S. Army’s White Sands Missile Range
Fundamental Tenets of Model Based Code Generation

• Models depict the requirements with sufficient detail, such that rules may be applied to automatically transform the modeling products into source code, programming, VHDL, scripts, etc.
  – Fidelity:
    • Completeness
    • Consistency
    • Level of Detail/Accuracy
  – Beneficial Side Effects:
    • Analysis
    • Execution
Fundamental Tenets of Model Based Code Generation

- **Model Driven Architecture** enables practical application of automated code generation by isolating models of platform independent behavior from models of platform specific behavior
  * (Computing) Platform Independent Models
    + Allow the engineer to focus on behavior versus implementation
    + Provide greater support for reuse (modular & platform independent)
    + Provide greater support for test (pre-HW and pre-code)
      - Slight performance hit for interface layer code implementation
  * (Computing) Platform Specific Models
    + Allow the engineer to focus on platform and physical architecture
    + Provide greater support for integration (visual integration of threads & interfaces)
    + Provide greater support for test (pre-application)
      - Reuse limited to like platform / deployed architecture
  * Enabling Features
    - Isolation / Layers / Bridges / Mediators
    - Subject Matter Domains / Domain Specific Languages & Tools

> 90% of Typical Flight SW Can Be Platform Independent
Fundamental Tenets of Model Based Code Generation

• Models provide a higher level of abstraction than high order programming languages
  – Graphical representations capture specific forms of information more efficiently and more concisely than syntactical statements
    • Application Software
      – Structure
      – Interfaces / Connectivity
      – Behavior / Logic / Conditionality / Concurrency
      – Order / Dependency / Relationships
  – Action language is preferred for
    • Atomic Behavior
      – Simple Equations
      – Instance Creation, Deletion, Linking, Assignment

Visual Representations are Preferred to Syntax for Complex Concepts
Fundamental Tenets of Model Based Code Generation

• Models allow the engineer to focus on analysis and design without regard to the platform & target programming language, similar to the way that high order programming languages allow the engineer to focus on implementation of the desired behavior without regard to the target processor
  – Logical Architecture
  – Concept of Operations
  – Functionality
  – Interactions
Auto Code – It’s Expected Era

• NASA’s Orion Multi-Purpose Crew Vehicle Software
  – Ground, Flight & Simulation Software
    • Object Oriented, Mathematical, GUI, DB, Plant Automation Models
    • C/C++ Source Code
      – Multi M-SLOC
      – Real-Time Embedded & Other
      – Deterministically Scheduled & Other
      – Flight / Safety Critical & Other
• PPC Fault Tolerant Processors & Others
NASA’s Orion Multi-Purpose Crew Vehicle Mission

http://www.youtube.com/watch?v=Kvalg9kBA9Y
This Video is from the Lockheed Martin Video Site on You Tube
NASA’s Orion Multi-Purpose Crew Vehicle FSW

Onboard Data Network Switches

Rate Scheduled (Hz) 40 20 10 5 1 As Needed

Lockheed Martin
Enablers of Automated Code Generation

- **Benefits Provided by Modeling**
  - Measurable Productivity Improvements
  - Measurable Defect Reduction

- **Workforce Evolution**
  - Increasing Number of Trained Modelers
  - Increasing Number of Model Based Champions
  - Attrition of Programmers

- **Technology Improvements**
  - Tool Vendor Code Generator Improvements Are Progressing
  - Action Language for Foundational UML Finally Approved

The Obstacles are Being Removed
Converging on Autopia

Changing Paradigms
Modeling Systems, HW and SW
Demonstrating Advantages
Delivering Products

Evolving, Defining, Refining Standards
Collaborating with Users

Training Modelers
Changing Perspectives / Expectations
Exploring Possibilities

Improving Tools
Competition Insures Progress & Evolution
Offering Affordable Tools

We Are Heading in Synergistic Directions
Reaching Autopia

• Challenge to Workforce
  – Formulate and demonstrate practical application of modeling abstractions and automated programming
    • Evolve the abstractions of ALF
  – Continue to raise expectations, resist complacency

• Challenge to Academia
  – Train the future modelers
  – Explore the possibilities

• Challenge to Standards Bodies
  – Provide vision and leadership
  – Resist political influences

• Challenge to Vendors
  – Provide code generators that produce results:
    • as good as those of a skilled programmer
    • that are easy to configure

We Must Work Together to Realize the Next Generation of Automation