Modellbasierte Entwicklung eingebetteter Software: Neue Wege für die Automobilindustrie?

Dr. Bernhard Schätz
Technische Universität München
Institut für Informatik
Lehrstuhl für Software and Systems Engineering/Prof. Dr. Manfred Broy
Heading for an Embedded Software Crisis?

- Software is not the problem
- Software is the answer
- **Bad** software is the problem

- Lesson learned from software crisis:

  Software engineering - like any other engineering discipline - needs suitable models, methods, and tools
Overview

• Transfer Solutions:
  – What’s the issue and how do others go about it

• Model for Automotive Software:
  – Abstraction: Services, Components, Tasks

• Model Based Development:
  – Integrating Abstractions: Consistency and Transformation

• Conclusion:
  – Model Based Development: How does it fit and what is next?
Software Engineering Eras

• Algorithmic, batch-oriented:
  – Domain: Scientific computing
  – Complex algorithms, simple data, single functions, non-reactive
  – Solutions: High-level programming languages

• Process oriented:
  – Domain: Business process automation
  – Simple algorithms, complex data, complex interacting functions, weakly reactive (monolithic)
  – Solutions: High-level architectures, high-level modeling of functional/architectural structure

• Service oriented:
  – Domain: Telecommunication, web-based applications
  – Simple algorithms, complex/distributed data, complex distributed interacting functions, complex reactive behavior
  – Solutions: High-level protocols, high-level modeling of structure /behavior
One size fits all…

- Organizational approaches: new business model, change of organizational structures, introduction of new processes

- Technical approaches:
  - Architectures:
    - Platform abstraction, platform independence
    - Multi tier architectures
  - Development process:
    - Requirements engineering
    - Quality Management (constructive and analytical)
    - Component reuse
  - Model-based development:
    - Abstract models and descriptions of the system under development
    - Model transformations
Modeling: How others do it...

- Business Information Systems:
  - Analyzing Functionality: SA, Function Trees, Use Cases
  - Abstract Structural Models: OOA/OOD, OMT, UML

- Telecommunication:
  - Analyzing Interaction: MSCs, Interaction Diagrams, Chisel
  - Abstract Behavioral Models: SDL, UML-RT

- Avionics:
  - Interface Models: Data Flow Diagrams
  - Completeness and Consistency: Software Cost Reduction
Automotive is different than…

• Business Information Systems:
  – Safety: Hard real-time, high reliability, controlled environment
  – Platform abstraction: low-level intra structure (OS, protocols)

• Telecommunication:
  – Modeling tradition: Interaction-based description (SDL)
  – Distributed Control: Interaction vs. communication

• Avionics:
  – Lot size: High numbers of delivered systems, low costs
  – Resources: Little redundancy (CPUs, busses)
Automotive is like…

- **Business Information Systems:**
  - Increasing functionality: New design spaces for functionality
  - Integration of functionality: Interplay of functions

- **Telecommunication:**
  - Distributed functionality: Highly interactive functionality
  - Shift of paradigm: Interaction vs. procedure call

- **Avionics:**
  - Complexity of models: Restricted models
  - High safety standards: Consistency, completeness
Overview

• Transfer Solutions:
  – What’s the issue and how do others go about it

• Model for Automotive Software:
  – Abstraction: Services, Components, Tasks

• Model Based Development:
  – Integrating Abstractions: Consistency and Transformation

• Conclusion:
  – Model Based Development: How does it fit and what is next?
Hot Spot Models

Some buzzwords:
• Unified Modeling Language
• Model Driven Architecture
• Model-Based Development
• Meta Models
• Design Pattern
• Process Pattern

Some tools:
• Rational Rose
• Rhapsody
• Telelogic Tau
• Together
• Eclipse
• Matlab/Simulink
• ASCET-SD

Model Oriented Development =
Development using (graphical) models?
What makes SE-models good models?

- Abstract: reducing complexity/possibilities for errors
- Integrated: unifying development process
- Supported: enabling mechanization

What we aim at:

- Increased quality of system development process
- Increased efficiency of system development process
Target Area: ECU Development

Integrated Model of SUD

Service Architecture
(Definition of Services by Function Blocks)

(Re)Partition

Component Architecture
(Inter Processor Communication: Processors, Plattform, Busses)

(Re)Schedule

Task Architecture
(Intra Processor Communication: Definition of Tasks)
Model: Abstraction

Integrated Model of SUD

Service Architecture

Component Architecture

Task Architecture
Service Architecture

- Development phase:
  - Requirements Analysis, Logical Design

- Elements:
  - Service, Sub Service, Function, Interface, Signal, Event

- Activities:
  - Definition of system interface
  - Definition of services/functions
Service Architecture

- Notations:
  - Structured Text (requirements)
  - Function Structure Diagrams
  - Data Type Diagrams (signals)
  - System Structure Diagrams (interface)
  - Interaction Diagrams (events)
Component Architecture

- Development Phase:
  - Logical design, technical design

- Elements:
  - Component, subcomponent, channel, signal, state, transition

- Activities:
  - Mapping of functions to components
  - Definition of behavior and communication
Component Architecture

- Notations:
  - Data Type Diagrams (signals)
  - System Structure Diagrams (interface)
  - Interaction Diagrams (events)
  - State Transition Diagrams (behavior)
Task Architecture

- Development Phase:
  - Technical Design, technical implementation

- Elements:
  - Task, schedule, event, signal, state, transition

- Activities:
  - Mapping of communications
  - Definition of tasks
Task Architecture

- Notations:
  - Data Type Diagrams (signals)
  - System Structure Diagrams (interface)
  - Task Structure Diagrams (dependencies)
  - State Transition Diagrams (behavior)
Levels of Abstraction

• Suitable Abstractions:
  – Reducing complexity
    • Different views in different phases
  – Reducing error possibilities
    • Restricting degree of (unnecessary) freedom

• Abstractions on all levels: covering the complete process

• Profit: Increased efficiency, additional quality

• State of the Art: Partially applied (structural aspects)
Overview

• Transfer Solutions:
  – What’s the issue and how do others go about it

• Model for Automotive Software:
  – Abstraction: Services, Components, Tasks

• Model Based Development:
  – Integrating Abstractions: Consistency and Transformation

• Conclusion:
  – Model Based Development: How does it fit and what is next?
Example: Integration

Integrated Model of SUD

Service Architecture

Component Architecture

Task Architecture
Integration of Abstractions
What’s behind: Conceptual Model

At most two channels can be connected to a port:
One to the environment of a component and one to its internal sub-structure

Expression constructed according to the rules for the associated data type, not treated here in detail

Predicates over the component's encapsulated data elements, not treated here in detail
Example: Applying Integrated Models

- Syntactic soundness/consistency of views
  - Invariant (Cannot be violated):
    - All messages have a defined recipient
  - Variant (Ensured when needed)
    - All functions are allocated to processors

- Semantic soundness/consistency of views:
  - All task have deterministic behavior
  - All services are covered by functions
In a Nutshell: Integrated Models

- Principle:
  - All views are abstraction of the model
  - Model covers complete development process

- Soundness/Consistency of views
  - Syntactic: Conceptual Model
  - Semantic: System Model

- Enabling: Change of views

- Profit: Increased quality

- State of the Art: Partially applied (syntactic level)
Applying the Model: Support

Integrated Model of SUD

Service Architecture

Component Architecture

Task Architecture
Example: Mapping Requirements

Diagram:

- Requirement
  - SubRequirements
  - DataElement
    - SubComponents
    - Component
      - ControlState
      - Connector
        - Transition
          - PreCondition
          - PostCondition
  - DataType
    - Port
      - Channel
        - {disjoint}
      - InputPort
      - OutputPort
        - InputPattern
        - OutputPattern
Example: Mapping Requirements
Example: Conflicts
Example: Conflict

- Overlapping transitions
  - Explicit nondeterminism (modeling the environment)
  - Conflicting execution (modeling the system)
Example: Completeness

- Completeness:
  - Detect undefined situations
  - Completions: Add standard behavior (e.g., error handling, ignore)
Example: Transformation
Example: View transformation

- **STD2EET-Generator**
  - Variable instantiation
  - Criterium:
    - Coverage-driven
    - Property-driven

- **EET2STD-Transformation:**
  - Universal/existential
  - Triggered/initial
  - Positive/negative
Example: Support

• Checking soundness of the model
  – Syntactic: All functions/tasks are allocated
  – Semantic: Service descriptions are not incompatible

• Transforming the model:
  – Generating views: Services to Functions
  – Transforming views: Replace abstract communication by implementation signal
Models and CASE

- Preliminary: Integrated Model
  - All views
  - All phases

- Analytical methods:
  - Checking model for soundness
  - Increasing quality

- Generative methods:
  - Transforming model
  - Increasing efficiency

- Profit: Increased efficiency, additional quality

- State of the Art: Weak validation, partial generation (Code, tests)
Overview

• Transfer Solutions:
  – What’s the issue and how do others go about it

• Model for Automotive Software:
  – Abstraction: Services, Components, Tasks

• Model Based Development:
  – Integrating Abstractions: Consistency and Transformation

• Conclusion:
  – Model Based Development: How does it fit and what is next?
Model-Based Development: In a nutshell

- Different views/abstractions, covering complete process:
  - General: Interface, interaction, behavior, etc.
  - Domain specific: Task, bus, schedule, etc.

- Integrated into a single model
  - Design: Invariant consistency
  - Analysis: Variant consistency

- Supported development process:
  - Operations: Refactor, refine communication, etc.
  - Process model: Guidance, support
How does it fit?

- **Automotive Model:**
  - Similar levels of abstraction, e.g.
    - Bosch CARTRONIC
    - ForSoft Automotive
  - Similar views, e.g.
    - Automotive UML (DC)
    - AML Automotive Modeling Language (ForSoft Automotive)

- **Tool Tradition:**
  - Service: DOORS, UML tools
  - Component: UML-RT
  - Task: ASCET-SD, Matlab/Simulink

  - But: No tool chain without an integrated model
To sum up...

• Models are not useful in themselves!

• Models are useful if integrated!

• Models are even more useful if integrated in a model-based development process!