Seamless Model Driven Development and Tool Support for Embedded Software-Intensive Systems

Computer Journal Lecture - 22nd June 2009

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
The role of modelling in software & systems engineering

Software & systems engineering means
• capturing the requirements
  ◊ domain specific
  ◊ functional, logical, technical, methodological
• specification of the system’s overall functionality
• design of a solution in terms of
  ◊ an architecture
  ◊ specifying the components
• implementing the components
• verification of the components and
• integrating them into the system and verifying the integration
• verification of the system
• further evolution

These are complex error prone tasks!
Modelling helps for:

• expressing and documenting the requirements
• specifying the system
• describing the architecture
  ◊ specifying the components
  ◊ their composition and interaction
• modelling the components
• verifying of the components and
• integrating them into the system and verifying the
• verifying the system
• further evolution
The roots of modelling in S&SE

Graphical description:
• Early approaches: SADT, Structured Analysis (SA)
• Later: SDL, ADLs, OOA/D, ROOM
• Today: UML, SysML

Programming and programming languages
• Programming concepts: types (as basis for data models)
• Programming logics
• Object oriented programming concepts

Formal description techniques as modelling concepts
• Predicate Logic Based Specification
• Abstract Data Types
• State Machines (Mealy, ... )
• Temporal Logic
• Process Algebras (CCS, CSP, ... )
• Models of distributed concurrent systems (Unity, TLA, ... )
On models and modelling

What is a model?
◊ An abstraction!

What kind of models?
◊ Informal: language, informal diagrams, ...
◊ Semiformal: formalized graphical or textual presentation languages
◊ Mathematical: in terms of mathematical theories
◊ Formal models: formalized syntax, semantics and logics

How do we use models?
◊ for understanding - Gedankenmodell
◊ for specification, design and documentation
◊ for analysis, validation, simulation, verification, certification
◊ for generation of implementation, tests,
◊ for reuse
What do we model

• Domain specific
  ◊ tautologies, ontologies, data models, ...
  ◊ laws, rules, ...
  ◊ ...

• System specific
  ◊ data
  ◊ interface behaviour
  ◊ architecture
  ◊ state
  ◊ temporal
  ◊ ...

• Technical
  ◊ Protocols
  ◊ CPUs
  ◊ ...

We concentrate on digital (discrete) models in the following
Modelling and engineering

• Support of development processes
  ◊ support in various phases of development
  ◊ integrated support between phases of development

• Support of engineering principles
  ◊ levels of abstraction
  ◊ separation of concerns
  ◊ encapsulation
  ◊ modularity

• Tool support
The five areas of modelling

- Mathematical Models
- Logical Theories
- Description Techniques
- Methodology
- Tools
Informal requirements

Formalized system requirements in terms of service taxonomies

Requirements Engineering Validation

Component implementation verification
R1 ⇒ S1
R2 ⇒ S2
R3 ⇒ S3
R4 ⇒ S4

Architecture design
Architecture verification
S ≜ S1⊗S2⊗S3⊗S4

Integration
R = R1⊗R2⊗R3⊗R4

Realization
Ingredients for Integration

• Coherent Theory
  ◊ Modelling (data/interface/state/interaction/architecture)
  ◊ Refinement
  ◊ Verification
• Consistent Terminology
• Tractable Description Techniques
  ◊ Formulas/Logics
  ◊ Diagrams/ Graphics
  ◊ Tables
• Comprehensive Architecture Structuring
• Flexible Development Process
  ◊ Phases (Requirements/Design/Implementation/Test/Integration)
  ◊ Artefact Model (concept)
  ◊ Methods
  ◊ Process models
• Powerful Tools
  ◊ Artefact Model (tool support)
  ◊ Automation for documentation, analysis, verification, generation
What has to be modelled?

- **Data**
  - states and their attributes
  - messages, events, signals
- **Requirements and specifications**
  - Functional
  - Nonfunctional - Quality Models
- **System Architecture**
  - Structure
  - Components/interfaces
  - Hierarchy
  - Hardware/Software/Deployment
- **Software Architecture**
  - Modules
  - Tasks
- **Test Cases**
- **Development Processes**
- **Development Steps**
  - Refactoring
  - Code generation
- **Quality attributes**
- **Systems**
Towards a comprehensive theory of system modelling: meta model

Feature model
Composition
Refinement
Time

Hierarchy and architecture
Is sub-feature

Interface model: components
Input and output

Process transition model:
Events, actions and causal relations

Abstraction
Uses

Process transition model:
Events, actions and causal relations

Abstraction
Uses

State transition model:
States and state machines

Abstraction
Uses

Data model:
Types/sorts and characteristic functions

Composition
Refinement
Time

Implementation

Composition
Refinement
Time

Implementation

Composition
Refinement
Time

Implementation

Abstraction

Hierarchy
and architecture
Is sub-feature

Abstraction

Hierarchy
and architecture
Is sub-feature

Abstraction

Hierarchy
and architecture
Is sub-feature

Hierarchy
and architecture
Is sub-feature

Abstraction

Hierarchy
and architecture
Is sub-feature

Hierarchy
and architecture
Is sub-feature
What is a (discrete) system?

A system
• has a scope
• a behavior
• a structure and distribution
• a black box view: an interface and an interface behaviour
  ◊ input and output via ports, channels, events, messages, signals
• a glass/white box view:
  ◊ architecture
  ◊ state and state transition
• quality profile
Towards a uniform model: Basic system model

**System class:** distributed, reactive systems

System consists of
- named components (with local state)
- named channels

**driven by global, discrete clock**
Timed Streams: Semantic Model for Black-Box-Behavior

Message set:
\[ M = \{a, b, c, \ldots\} \]

Messages transmitted at time \( t \)
The Basic Behaviour Model: Streams and Functions

\[ C \quad \text{set of channels} \]

Type: \( C \rightarrow \text{TYPE} \quad \text{type assignment} \)

\[ x : C \rightarrow (\mathbb{N}\{0\} \rightarrow M^*) \quad \text{channel history for messages of type } M \]

\( \tilde{C} \) or \( \text{IH}[C] \quad \text{set of channel histories for channels in } C \)
System interface model

Channel: Identifier of Type stream

$I = \{ x_1, x_2, ... \}$ set of typed input channels
$O = \{ y_1, y_2, ... \}$ set of typed output channels

Interface behavior

$f: \vec{I} \rightarrow \vec{O}$

Set of interfaces: $\text{IF}[I \rightarrow O]$
System interfaces

$$(I \times O)$$ syntactic interface with set of input channels $I$ and of output channels $O$

$F : \tilde{I} \rightarrow \wp(\tilde{O})$ semantic interface for $(I \times O)$ with timing property addressing causality

(let $x, z \in \tilde{I}, y \in \tilde{O}, t \in \mathbb{N}$):

$$x\downarrow t = z\downarrow t \Rightarrow \{y\downarrow t+1: y \in F(x)\} = \{y\downarrow t+1: y \in F(z)\}$$

$x\downarrow t$ prefix of history $x$ with $t$ finite sequences

A system is a total behavior

Component interface
Example: Component interface specification

A transmission component TMC

Input channel

```
Spec name
```

```
TMC
```

```
in
```

```
x: T
```

```
out
```

```
y: T
```

```
x ~ y
```

```
x ~ y ≡ (∀ m ∈ T: \{m\} ⊕ x = \{m\} ⊕ y)
```

Output channel
State model for systems/components

\[ \Sigma \quad \text{set of states, initial state } \sigma \subseteq \Sigma \]

State transition function:

\[ \Delta: (\Sigma \times (I \rightarrow M^*)) \rightarrow \wp (\Sigma \times (O \rightarrow M^*)) \]

State machine (infinite Moore automaton):

Interface abstraction

Abs : SM \rightarrow IF

Abs(\(\Sigma, \sigma\)) = F_{\sigma} \quad \text{where}

\[F_{\sigma}(\langle \langle z \rangle \rangle^x) = \{\langle s \rangle^y: y \in F_{\sigma}(x) \wedge (\sigma', s) \in \Delta(\sigma, z)\}\]

Set of all states machines: SM
Composition and Decomposition of Systems

$$F_1 \in IF[I_1 \triangleright O_1]$$
$$F_2 \in IF[I_2 \triangleright O_2]$$

$$C_1 = O_1 \cap I_2$$
$$C_2 = O_2 \cap I_1$$

$$I = I_1 \setminus C_2 \cup I_2 \setminus C_1$$
$$O = O_1 \setminus C_1 \cup O_2 \setminus C_2$$

$$F_1 \otimes F_2 \in IF[I \triangleright O],$$

$$(F_1 \otimes F_2).x = \{z|O: x = z|I \land z|O_1 \in F_1(z|I_1) \land z|O_2 \in F_2(z|I_2)\}$$
Interface specification composition rule

\[ F_1 \otimes F_2 = F_1 \otimes F_2 \]

\[ \begin{align*}
\text{F1} & \quad \text{in} \quad x_1, z_{21}: T \\
& \quad \text{out} \quad y_1, z_{12}: T \\
& \quad \text{P1}
\end{align*} \]

\[ \begin{align*}
\text{F2} & \quad \text{in} \quad x_2, z_{12}: T \\
& \quad \text{out} \quad y_2, z_{21}: T \\
& \quad \text{P2}
\end{align*} \]

\[ \begin{align*}
\text{F1} \otimes \text{F2} & \quad \text{in} \quad x_1, x_2: T \\
& \quad \text{out} \quad y_1, y_2: T \\
& \quad \exists \quad z_{12}, z_{21}: \text{P1} \land \text{P2}
\end{align*} \]
Composition of Specifications into Architectures

Composed component spec

\[
\text{in } x_1: M_1, x_2: M_2, \ldots \\
\text{out } y_1: N_1, y_2: N_2, \ldots \\
\exists c_1, c_2, \ldots : P_1 \land \ldots \land P_n
\]

System composition = logical $\land$

Channel Hiding = existential quantification

Input channels

Output channels

Internal channels
An example of an essential property...

Interface abstraction distributes for state machines over composition

\[
\text{Abs}((\Delta_1, \sigma_1) \parallel (\Delta_2, \sigma_2)) = \text{Abs}((\Delta_1, \sigma_1)) \otimes \text{Abs}((\Delta_2, \sigma_2))
\]
Vertical Refinement

$F : \mathcal{I} \to \mathcal{P}(\mathcal{O})$

is refined by a behavior

$\hat{F} : \mathcal{I} \to \mathcal{P}(\mathcal{O})$

if

$\forall x \in \mathcal{I} : \hat{F}.x \subseteq F.x$

we write

$F \rightarrow_{IF} \hat{F}$

Compositionality of refinement

$\forall k : F_k \rightarrow_{IF} \hat{F}$
Levels of abstraction

Given refinement pairs $\mathcal{A}_I: r_{I_2} \rightarrow \mathcal{P}(r_{I_1}) \mathcal{R}_I$, $r_O \rightarrow \mathcal{P}(r_{O_1}) \mathcal{R}_O$, we call $\mathcal{A}_2$ a refinement of $\mathcal{A}_1$ if there exists a mapping $\text{abs}: \Sigma_2 \rightarrow \Sigma_1$ such that:

$\{ (\text{abs}.\sigma_0, AO.y_0) : (\sigma_0, y_0) \in \mathcal{A}_2 \} \subseteq \mathcal{A}_1$

and for each reachable state $\sigma \in \Sigma_2$ of the state machine $(\mathcal{A}_2, \mathcal{L}_2)$ we have:

$B_{\mathcal{A}_2}(\sigma, y_0) \subseteq \mathcal{A}_1(\text{abs}.\sigma_0, AO.y_0) \subseteq \mathcal{R}_O$

Theorems

- Property refinement implies interaction refinement
- Compositionality of interaction refinement
- Interaction refinement distributes over composition
- Abstractions of interaction refinements of implementations are interaction refinements of abstractions
- Time abstraction is interaction abstraction
- Interaction abstraction is a Galois connection
The comprehensive model

Usage function hierarchy
service taxonomy
Logical architecture

conceptional architecture

Technical architecture

Software architecture

Deployment

Tasks
- T1
- T2
- T3
- T4
...

Deployment

Hardware architecture

T1...

T2...

T3 T4...
The overall goal

Provide a formal model for the comprehensive architecture and all of its views

In this talk: concentration on the conceptional architecture

• The foundation
  ◊ The basic system model: components
  ◊ System specification and verification
  ◊ System composition
• Service taxonomy
• Logical architecture
• Relationship between service taxonomy and logical architecture
A screen shot from AutoFocus
What we got

- **Formal notion of a system**
  - with input and output
  - represented by a relation between input and output histories
  - are specified by history assertions
  - can be used as a component to form a large system
  - can be de-composed into an architecture of components

- **Formal notion of a service/feature/system function**
  - with input and output
  - represented by a relation between input and output histories
  - are specified by history assertions
  - can be used as a sub-service to form a large system
  - can be de-combined into an taxonomy of services

- **Every component**
  - can be de-combined into its taxonomy of its sub-services
  - the sub-services can be related by service dependency relations
Concluding Remarks

• Today software & systems engineering is too much orientated towards the technical architecture and solutions/implementation in the beginning

• We need a comprehensive “architectural” model-based view onto systems including requirements for dealing with complex multi-functional systems

• The models allow for
  - Separation of concerns
  - Separation technical aspects from application aspects

• Technical architectures are modelled along the same theory

• Code and test cases can be generated from the models