Mandatory Properties of Component Concepts

Manfred Broy

Technische Universität München
Institut für Informatik
D-85748 Garching, Germany
Content

• What is a system component concept?
  A concept of a self-contained, independent unit carrying functionality that can be analysed, refined and composed to form larger systems:
  ◊ deployable
  ◊ executable
  ◊ adaptable

• Mandatory ingredients: concept of component
  ◊ specification - implementation independent
  ◊ composition/decomposition - architecture
  ◊ refinement - properties and levels of abstraction
  ◊ implementation - executable system descriptions
  ◊ abstraction
    • relating implementations to specifications
    • relating components at different levels of abstraction
These notions ...

... form a taxonomy

But this is not enough!

We need a semantic modelling theory!
System class: distributed, reactive systems

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

driven by a global, discrete clock
Basic system model

Timed Streams: Semantic Model for Black-Box-Behavior

Message set:

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

Messages transmitted at time \( t \)

Infinite channel history

\[ \langle a, d, a, b \rangle \]
The Basic Behaviour Model: Timed Streams and Channels

\[ C \] set of channels

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

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

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

Channel: Identifier of Type stream

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

Interface behavior

$F : \bar{I} \rightarrow \emptyset(\bar{O})$

Set of interface behaviours with input channels $I$ and output channels $O$:

$IF[I \blacktriangleright O]$}

Set of all interface behaviours: $IF$

A component is a system
A system has a proper time flow

\[ F: \tilde{I} \rightarrow \varnothing(\tilde{O}) \]

**Semantic interface** for \((I \triangleright O)\)

with **timing property addressing strong causality**

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

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

A system shows a **total** behavior

\[
x \downarrow t \quad \text{prefix of history } x \text{ of length } t
\]

\[
I \quad \text{Component interface} \quad O
\]
Example: Component interface specification

A transmission component TMC $x:T$

<table>
<thead>
<tr>
<th>Spec name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>x: T</td>
</tr>
<tr>
<td>in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>y: T</td>
</tr>
<tr>
<td>out</td>
</tr>
</tbody>
</table>

$TMC \ x \sim \ y$

$A \ system \ has \ a \ logical \ specification$

$x \sim y \equiv (\forall m \in T: \{m\} \#x = \{m\} \#y)$

{m} \#x denotes the number of copies of m in stream x
Verification: Proving properties about specified components

From the interface assertions we can prove

- Safety properties

\[ \{m\}\#y > 0 \land y \in \text{TMC}(x) \Rightarrow \{m\}\#x > 0 \]

- Liveness properties

\[ \{m\}\#x > 0 \land y \in \text{TMC}(x) \Rightarrow \{m\}\#y > 0 \]

A system specification
- can be structured by logical properties and
- can be used to prove properties
A state machine \((\Delta, \Lambda)\) consists of

- a set \(\Sigma\) of states - the state space
- a set \(\Lambda \subseteq \Sigma\) of initial states
- a state transition function \(\Delta\)

\(\Diamond\) in case of a state machine with input/output:

- events (inputs \(E\)) trigger the transitions and events (outputs \(A\)) are produced by them respectively:

\[
\Delta : \Sigma \times E \rightarrow \Sigma \times A
\]

- in the case of nondeterministic machines:

\[
\Delta : \Sigma \times E \rightarrow \wp(\Sigma \times A)
\]

- Given a syntactic interface with sets \(I\) and \(O\) of input and output channels:

\[
E = I \rightarrow M^*
\]
\[
A = O \rightarrow M^*
\]
A state machine \((\Delta, \Lambda)\) defines for each initial state 
\[ \sigma_0 \in \Lambda \]
and each sequence of inputs 
\[ e_1, e_2, e_3, \ldots \in E \]
a sequence of states 
\[ \sigma_1, \sigma_2, \sigma_3, \ldots \in \Sigma \]
and a sequence of outputs 
\[ a_1, a_2, a_3, \ldots \in A \]
through 
\[ (\sigma_{i+1}, a_{i+1}) \in \Delta(\sigma_i, e_{i+1}) \]
In this manner we obtain computations of the form

For each initial state $\sigma_0 \in \Sigma$ we define a function

$$F_{\sigma_0} : I \to \mathcal{P}(O)$$

with

$$F_{\sigma_0}(x) = \{y : \exists \sigma_i : \sigma_0 = \sigma_0 \land \forall i \in \mathbb{N} : (\sigma_{i+1}, x_{i+1}) = \Delta(\sigma_i, y_{i+1})\}$$

$F_{\sigma_0}$ denotes the interface behavior of the transition function $\Delta$ for the initial state $\sigma_0$.

Furthermore we define

$$\text{Abs}((\Delta, \Lambda)) = F_\Lambda$$

where:

$$F_\Lambda(x) = \{y \in F_{\sigma_0}(x) : y \in F_{\sigma_0}(x) \land \sigma \in \Lambda\}$$

$F_\Lambda$ is called the interface behavior of the state machine $(\Delta, \Lambda)$.
Moore Machines

- A Mealy machine \((\Delta, \Lambda)\) with
  \[
  \Delta : \Sigma \times E \rightarrow \wp (\Sigma \times A)
  \]
  is called a **Moore machine** if for all states \(\sigma \in \Sigma\) and all inputs \(e \in E\) the set
  \[
  \text{out}(\sigma, e) = \{ a \in A : (\sigma, a) = \Delta(\sigma, e) \}
  \]
  does not depend on the input \(e\) but only on the state \(\sigma\).
- Formally: then for all \(e, e' \in E\) we have
  \[
  \text{out}(\sigma, e) = \text{out}(\sigma, e')
  \]

Theorem: If is \((\Delta, \Lambda)\) a Moore machine the \(F_\Lambda\) is strongly causal.

An interface abstraction of an implementation
- has the required properties
- leads to correct assertions

Moore, Grenoble, ESWEERK October 2009

Manfred Broy

TUM
Sub-Services, Functional Features
Syntactic sub-interfaces and projection

A typed channel set $C'$ is called a sub-type of a typed channel set $C$ if

- $C'$ is a subset of $C$
- The message types of the channels in $C'$ are subsets of the message sets of these channels in $C$

We write then

$$C' \text{ subtype } C$$

Then we denote for the channel history $x \in \mathcal{IH}[C]$ by

$$x|C' \in \mathcal{IH}[C']$$

the restriction of $x$ to the channels and messages in $C'$
Sub-types between interfaces

For syntactic interfaces \((I \to O)\) and \((I' \to O')\) where

\[I' \text{ subtype } I \text{ and } O' \text{ subtype } O\]

we call \((I' \to O')\) An interface behaviour can be structured into
- independent sub-services
- a system offers services

For a behavior \(F \in IF[I \to O]\) we define its projection \(F^\uparrow(I' \to O') \in IF[I' \to O']\)

to the syntactic interface \((I' \to O')\) by (for all \(x \in IH[I']\)):

\[
F^\uparrow(I' \to O')(x') = \{y | O' : \exists x \in IH[I] : x' = x|I' \land y \in F(x)\}
\]

The projection is called \textit{faithful}, if for all \(x \in \text{dom}(F)\)

\[
F(x) | O' = (F^\uparrow(I' \to O'))(x|I')
\]
Composing Systems
Composition

• of system models is compositional
• is hierarchical: a system is a component is a system

\[
\begin{align*}
F_1 \in & \text{IF}[I_1 \triangleright O_1] \\
F_2 \in & \text{IF}[I_2 \triangleright O_2] \\
C_1 = & O_1 \cap I_2 \\
C_2 = & O_2 \cap I_1 \\
I = & I_1 \setminus C_2 \cup I_2 \setminus C_1 \\
O = & O_1 \setminus C_1 \cup O_2 \setminus C_2 \\
F_1 \otimes F_2 \in & \text{IF}[I \triangleright O],
\end{align*}
\]

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

Interface specification is modular

\[ F_1 \otimes F_2 \]

<table>
<thead>
<tr>
<th>in</th>
<th>x1, z21: T</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y1, z12: T</td>
</tr>
<tr>
<td>P1</td>
<td></td>
</tr>
</tbody>
</table>

| out  | y2, z21: T        |
| P2   |                   |

\[ \exists z12, z21: P1 \land P2 \]
Composition of the two state machines

Consider Moore machines \( M_k = (\Delta_k, \Lambda_k) \) \((k = 1, 2)\):

\[
\Delta_k : \Sigma_k \times (I_k \rightarrow M^*) \rightarrow (\Sigma_k \times (O_k \rightarrow M^*))
\]

We define the composed state machine

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

as follows

\[
\Sigma = \Sigma_1 \times \Sigma_2
\]

for \( x \in I \) and

\[
\Delta((s_1, s_2), x) = \{((s_1', s_2'), z|O) : x = z|I \land \forall k : (s_k', z|O_k) = \Delta_k(s_k, z|I_k) \}
\]

This definition is based on the fact that we consider Moore machines.

We write

\[
\Delta = \Delta_1 \parallel \Delta_2
\]

\[
M = M_1 \parallel M_2 = (\Delta_1 \parallel \Delta_2, \Lambda_1 \times \Lambda_2)
\]
An example of an essential property ...

\[
\text{Abs}\left( (\Delta_1, \sigma_1) \ || \ (\Delta_2, \sigma_2) \right) = \text{Abs}(\Delta_1, \sigma_1) \otimes \text{Abs}(\Delta_2, \sigma_2)
\]

Interface abstraction distributes for state machines over composition
Composition of Specifications into Architectures

Input channels

Composed component spec

<table>
<thead>
<tr>
<th>in</th>
<th>x₁: M₁, x₂: M₂, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y₁:</td>
</tr>
<tr>
<td>∃</td>
<td>c₁, c₂,</td>
</tr>
</tbody>
</table>

Output channels

Internal channels

Architectures
• have components with
  - interface specifications
  - implemented by
    • state machines
    • architectures
• are hierarchical forming systems of systems

System composition = logical and

Channel Hiding = existential quantification
Refining Systems
Horizontal Refinement

\[ F : \tilde{I} \rightarrow \tilde{I} \]

is refined by another function

\[ \hat{F} : \tilde{I} \rightarrow \tilde{I} \]

if

\[ \forall x \in \tilde{I} : \hat{F}(x) \subseteq F(x) \]

we write

\[ F \mathrel{\bowtie}_\text{IF} \hat{F} \]

Compositionality of refinement

\[ \forall k : F_k \bowtie \hat{F}_k \]

\[ \bowtie \{ F_k : k \in K \} \bowtie \bowtie \{ \hat{F}_k : k \in K \} \]
Vertical Refinement: Changing Levels of Abstraction

Theorems

- Horizontal refinement implies vertical refinement
- Compositionality of vertical refinement
- Vertical refinement distributes over composition
- Abstractions of vertical refinements of implementations are vertical refinements of abstractions
- Vertical refinement is a Galois connection
Expressive power: time

• The system model can express timing properties

\[ y \in \text{TMC}(x) \Rightarrow \{m\}#x\downarrow t \leq \{m\}#y\downarrow t + \delta \]

• The time granularity can be refined
  ◇ a special case of vertical refinement
Mandatory properties

• Concept of interface - interface abstraction
  ◊ syntactic interface
  ◊ interface specification - behaviour
  ◊ verification of interface properties
  ◊ relating components (compatibility, refinement)
  ◊ behavioural abstraction

• Implementation
  ◊ interface abstraction - correctness
  ◊ verification of properties - testing, model checking and deductive proofs

• Composition
  ◊ architectures
  ◊ compositional
    • behavioural specification
    • implementations
    • architectures
  ◊ hierarchical (system of systems)
  ◊ modular for refinement

• Further aspects
  ◊ sub-functions (function hierarchy)
  ◊ time
  ◊ probability
  ◊ performance
  ◊ ...
Specification, verification, architecture ...

Informal requirements

Requirements Engineering
- Validation
  - Formalized system requirements in terms of service taxonomies

System delivery
- System verification
  - \( R \Rightarrow S \)

Integration
- \( R = R_1 \otimes R_2 \otimes R_3 \otimes R_4 \)

Architecture design
- Architecture verification
  - \( S \leftarrow S_1 \otimes S_2 \otimes S_3 \otimes S_4 \)

Component implementation
- Verification
  - \( R_1 \Rightarrow S_1 \)
  - \( R_2 \Rightarrow S_2 \)
  - \( R_3 \Rightarrow S_3 \)
  - \( R_4 \Rightarrow S_4 \)

Components:
- R1
- R2
- R4
- R3

Deliver

deliver
Open Issues

• Probability
• Non-functional properties
• Modelling of
  ◊ hardware issues
  ◊ mechanical aspects
Refinement layers

- A layer refinement pair are two layers that form the time independent identity

Two layers $L$ and $L'$ are called a refinement pair for if

$$L \otimes L' = \text{Id}(I \triangleright O)$$
Layered protocols

Remember
The comprehensive model

- Usage function hierarchy
- Service taxonomy
- Logical architecture
- Conceptional architecture
- Technical architecture
- Software architecture
- Tasks
  - T1
  - T2
  - T3
  - T4
  - ...
- Deployment
- Hardware architecture