Model-Based Software and Systems Engineering

Elements of Seamless Development

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Modeling in Software and Systems Engineering

When modeling software in systems we have capture following aspects:

- interaction – exchange of information/material
- distribution – structuring systems in architectures with elements related to locations
- context – operational system's environment
- real time
- probability
- physicality
- …

To do that we have to use concepts

- interfaces – scope and interaction
- state – state transition
- architecture – (de-)composition

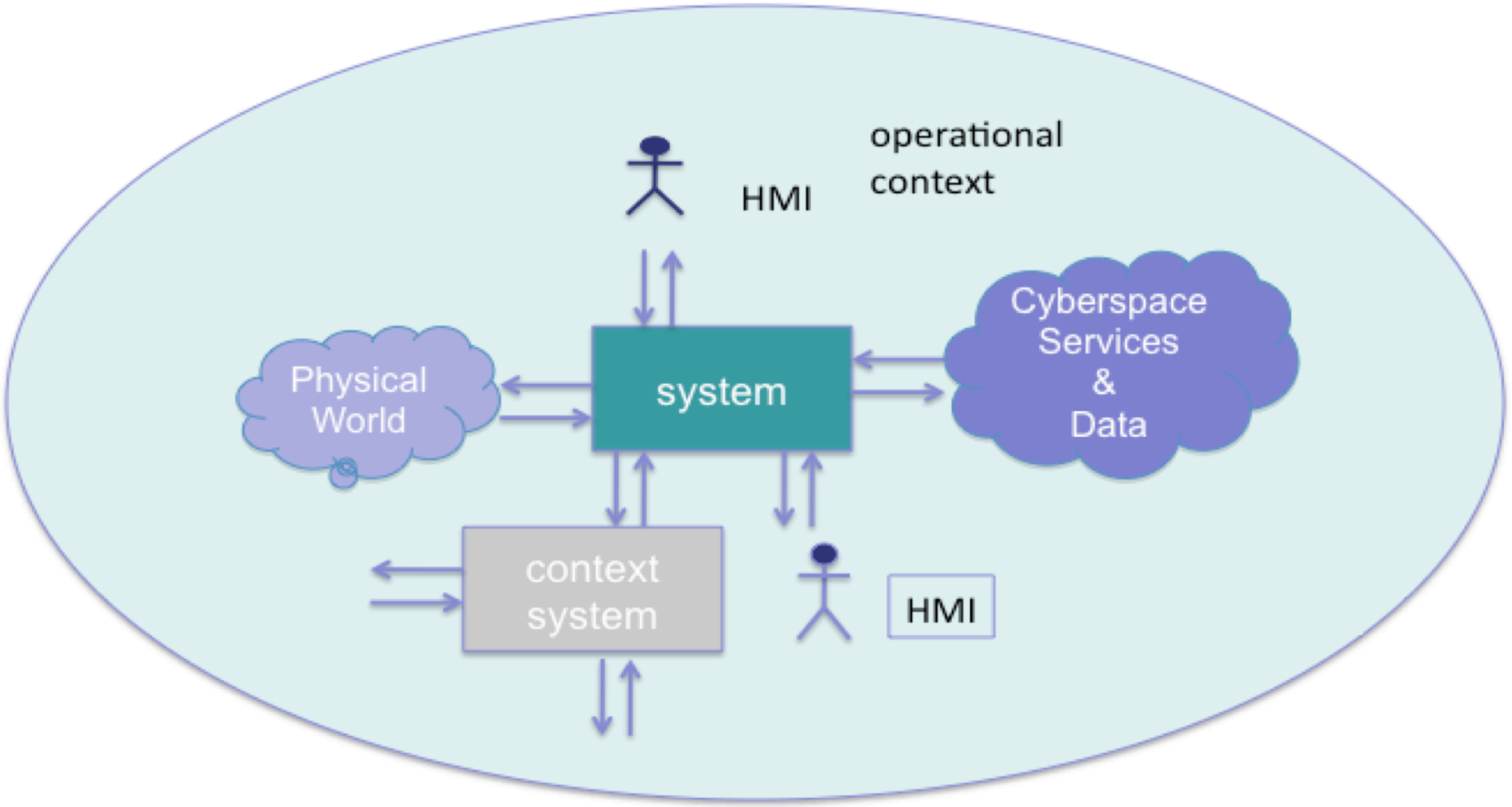
What is seamless development?

- Development by a chain of models
  - High expressive power
  - Clear structure of role of models
  - All aspects in the development captured by models
  - Tight integration in the artifact models
- Development steps by well-defined relationship between modes
  - Refinement
  - Decomposition
  - Change of scope
- Extended tool support
  - High automation
  - All artifacts in tools and comprehensive data base (development back bone)

Basic System Modeling Concepts

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

System and its context



Basic System Notion: What is a discrete system (model)

A system has

- a system boundary that determines
  - what is part of the systems and
  - what lies outside (called its context)
- an interface (determined by the system boundary), which determines,
  - what ways of interaction (actions) between the system und its context are possible (static or syntactic interface)
  - which behavior the system shows from view of the context (interface behavior, dynamic interface, interaction view)
- a structure and distribution addressing internal structure, given
  - by its structuring in sub-systems (sub-system architecture)
  - by its states und state transitions (state view, state machines)
- quality profile
- the views use a data model
- the views may be documented by adequate models

Discrete systems: the modeling theory – interface behaviour as key concept

Sets of typed channels

$I = \{x_1 : T_1, x_2 : T_2, \ldots\}$

$O = \{y_1 : T'_1, y_2 : T'_2, \ldots\}$

syntactic interface

$(I \blacktriangleright O)$

data stream of type T

$STREAM[T] = \{\mathbb{N}\setminus\{0\} \to T^*\}$

valuation of channel set C

$\mathbb{H}[C] = \{C \to STREAM[T]\}$

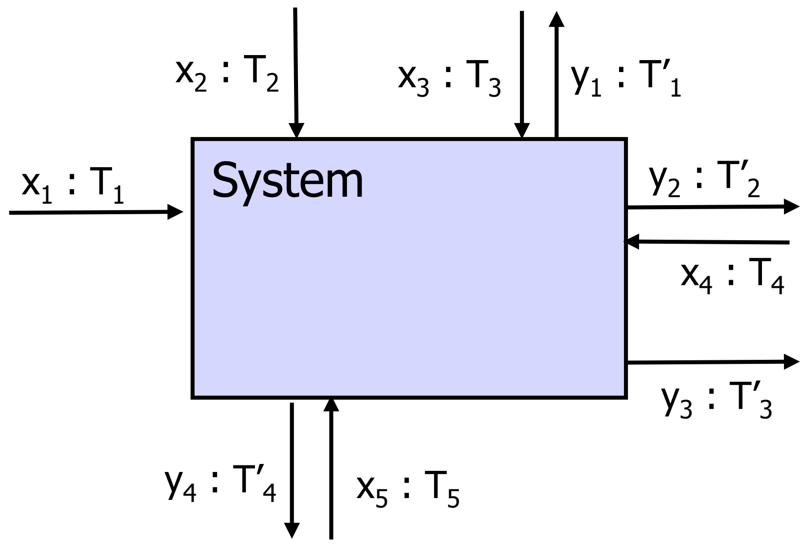
interface behaviour for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{\mathbb{H}[I] \to \wp(\mathbb{H}[O])\}$

interface specification

$p: I \cup O \to \mathbb{B}$

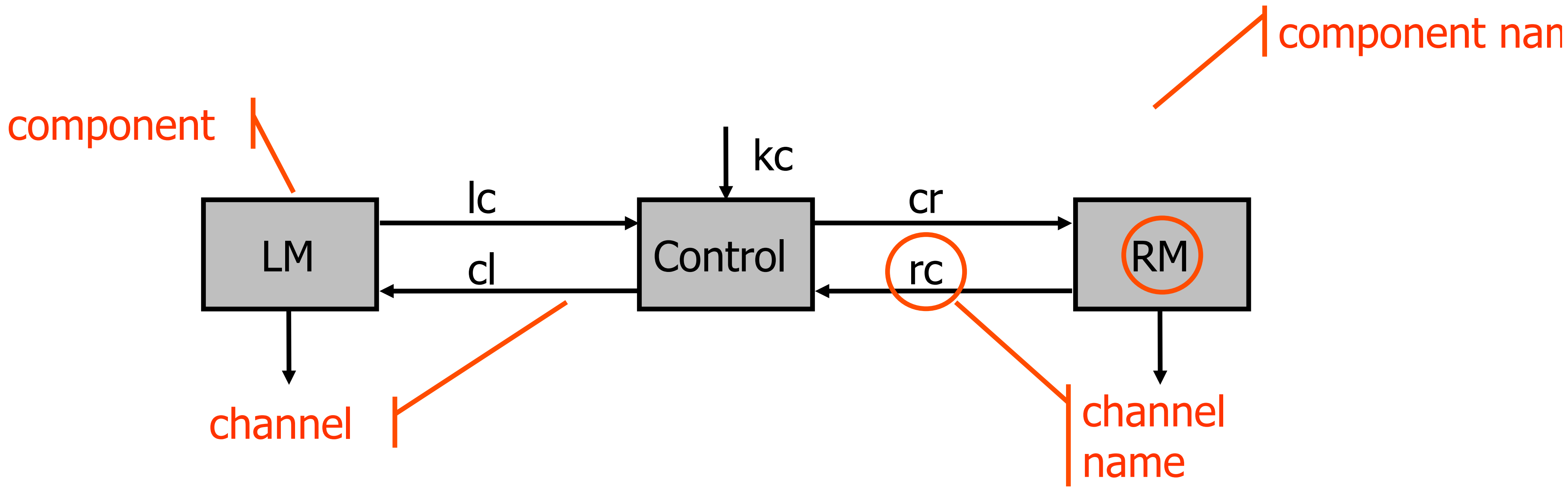
represented as interface assertion S
logical formula with channel names as variables for streams



See: M. Broy: A Logical Basis for Component-Oriented Software and Systems Engineering. The Computer Journal: Vol. 53, No. 10, 2010, S. 1758-1782

Towards a uniform model: Basic system model

System class: distributed, reactive systems

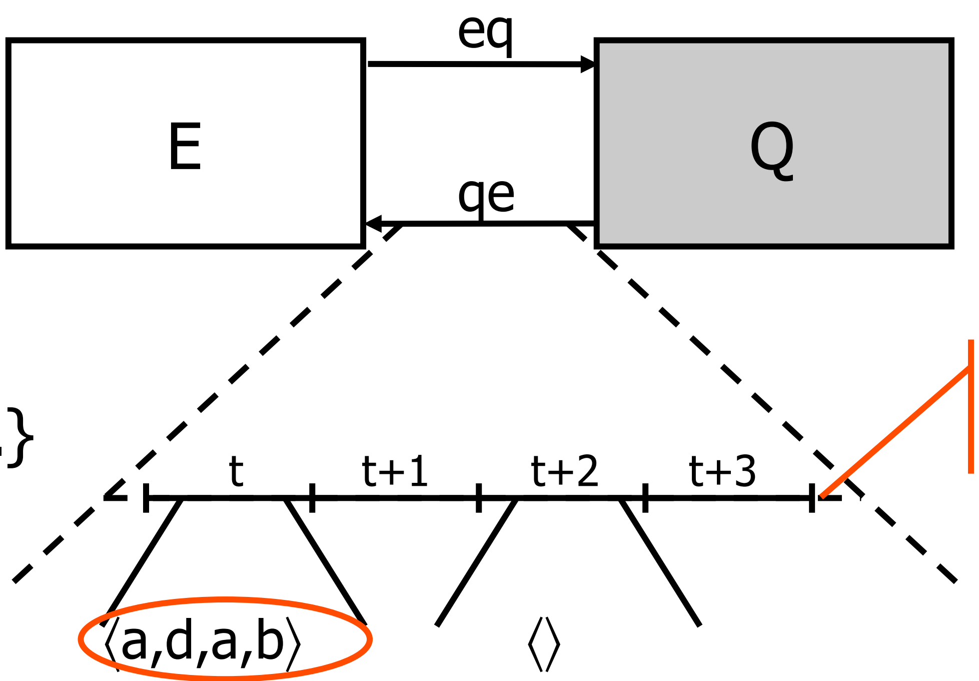


System consists of

- named components (with local state)
- named channels

driven by a global, discrete clock

Timed Streams: Semantic Model for Black-Box-Behavior



Message set:

M = {a, b, c, ...}

infinite channel history

Messages transmitted at time t

Modeling Interface Behavior

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

The Basic Behaviour Model: Timed Streams and Channels

C	set of channels
Type: $C \to \text{TYPE}$	type assignment
$x : C \to (\mathbb{N}\backslash\{0\} \to M^*)$	channel history for messages of type M
$\vec{C}$ or $\mathbb{H}[C]$	set of channel histories for channels in C

System interface model

Channel: Identifier of Type stream

$I = \{ x_1 : D_1, x_2 : D_2, \ldots \}$ set of typed input channels
$O = \{ y_1 : T_1, y_2 : T_2, \ldots \}$ set of typed output channels

Syntactic interface: $(I \blacktriangleright O)$

Interface behavior

$$F : \vec{I} \to \wp(\vec{O})$$

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

$$IF[I \blacktriangleright O]$$

Set of all interface behaviours: IF

System interface behaviour - causality

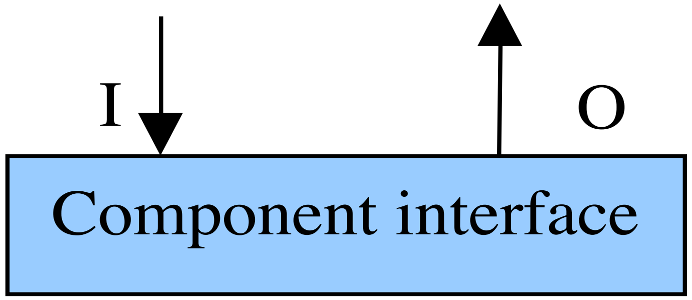
(I ► O) *syntactic interface* with set of input channels I and of output channels O

$F : \vec{I} \to \wp(\vec{O})$ *semantic interface* for (I ► O) with *timing property addressing* *strong causality* (let $x, z \in \vec{I}$, $y \in \vec{O}$, $t \in IN$):

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

$x{\downarrow}t$ prefix of history x of length t

A system shows a total behavior



Continuous systems: the model

Sets of typed channels

$I = \{x_1 : T_1, x_2 : T_2, \dots\}$

$O = \{y_1 : T'_1, y_2 : T'_2, \dots\}$

syntactic interface

$(I \blacktriangleright O)$

continuous data stream of continuous type M

$\mathrm{ConSTREAM}[T] = \{\mathrm{IR}_+ \to M\}$

valuation of channel set C

$\mathrm{CIH}[C] = \{C \to \mathrm{ConSTREAM}[T]\}$

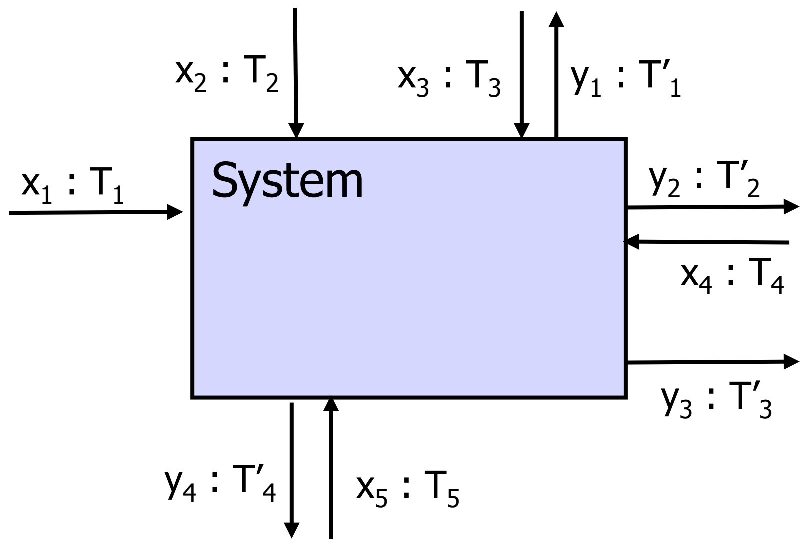
interface behaviour for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{\mathrm{CIH}[I] \to \wp(\mathrm{CIH}[O])\}$

interface specification

$p: I \cup O \to \mathrm{IB}$

represented as interface assertion S
logical formula with channel names as variables for continuous streams



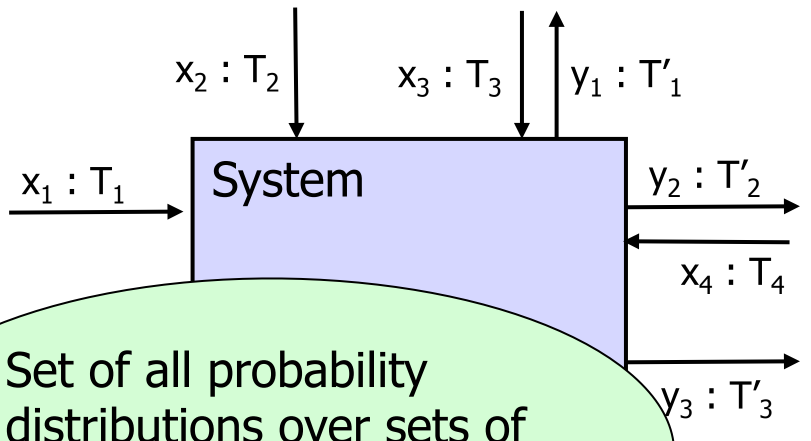
See: M. Broy: System Behavior Models with Discrete and Dense Time. To appear in: Advances in Real-Time Systems. Springer

Discrete systems: the modeling theory - probability

Sets of typed channels

$I = \{x_1 : T_1, x_2 : T_2, \dots \}$

$O = \{y_1 : T'_1, y_2 : T'_2, \dots \}$



syntactic interface

$(I \blacktriangleright O)$

data stream of type T

$STREAM[T] = \{IN\backslash\{0\} \rightarrow T^*\}$

valuation of channel set C

$IH[C] = \{C \rightarrow STREAM[T]\}$

Set of all probability distributions over sets of output histories

interface behaviour for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{IH[I] \rightarrow PD[\wp(IH[O])]\}$

interface specification

$p: I \cup O \rightarrow \mathbb{B}$

See: P. Neubeck: A Probabilitistic Theory of Interactive Systems. PH. D. Dissertation, Technische Universität München, Fakultät für Informatik, December 2012

represented as interface assertion S
logical formula with channel names as variables for streams

Extensions of the model

- Physical Aspects & Properties: Rich Models
  - Space
  - Geometry
  - Temperature
  - …

See: B. Hummel: Integrated Behavior Modeling of Space-Intensive Mechatronic Systems, Technische Universität München, Fakultät für Informatik, December 2010

Evolution and Development: Specification, Refinement, Compatibility

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Interface Specification

- An interface model describes in a particular abstraction the interface behavior of a system
  - ◇ interface behavior for syn. interface $(I \blacktriangleright O)$

$$[I \blacktriangleright O] = \{IH[I] \rightarrow \wp(IH[O])\}$$

  - ◇ interface specification by a predicate

$$p: I \cup O \rightarrow IB$$

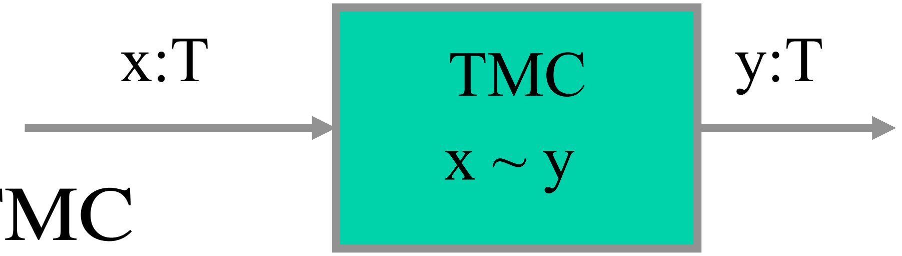
  written by an interface assertion

See: M. Broy: Software and System Modeling: Structured Multi-view Modeling, Specification, Design and Implementation. In: Conquering Complexity, edited by Mike Hinchey and Lorcan Coyle, Springer Verlag, Januar 2012, S. 309-372

Example: System interface specification

Spec name

A transmission component TMC



Input channel

See: M. Broy, K. Stølen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer 2001

TMC

in x: T

out y: T

x ~ y

Output channel

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

Specifying interface assertion

Verification: Proving properties about specified components

From the interface assertions we can prove

- Safety properties

$$m\#y > 0 \wedge y \in TMC(x) \Rightarrow m\#x > 0$$

- Liveness properties

$$m\#x > 0 \wedge y \in TMC(x) \Rightarrow m\#y > 0$$

Structure:
Composition and Decomposition

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Composing Interfaces

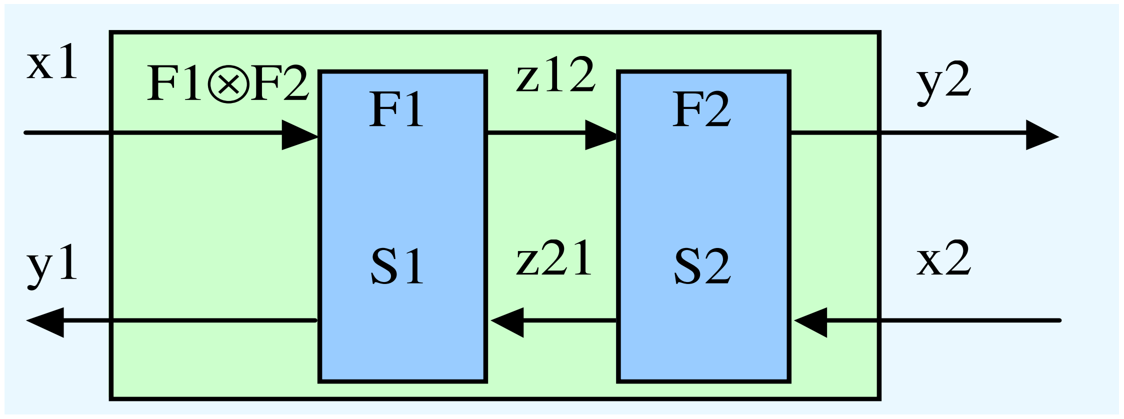
- Composition is an operation on syntactically compatible system interfaces

$$\otimes : [I_1 \blacktriangleright O_1] \times [I_2 \blacktriangleright O_2] \rightarrow [I \blacktriangleright O]$$

- The operation $\otimes$ induces a composition operation on specifications

See: M. Broy: A Theory for Requirements Specification and Architecture Design of Multi-Functional Software Systems. Series on Component-Based Software Development – Vol. 2. Mathematical Frameworks for Component Software. Models for Analysis and Synthesis, 2006, S. 119–154

Modularity: Rules of compositions for interface specs



F1

in	x1, z21: T
out	y1, z12: T
S1	

F2

in	x2, z12: T
out	y2, z21: T
S2	

F1⊗F2

in	x1, x2: T
out	y1, y2: T
$\exists$ z12, z21: S1 $\wedge$ S2	

Refining Interfaces

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Refinement

- The idea of system refinement is that systems are developed
  - ◇ by a sequence of development steps
  - ◇ each step produces a more refined system description
  - ◇ there is a refinement relation between the current system description and the produced system description
- The refinement relation
  - ◇ is a relation between systems descriptions
- The relation can be used as an idealized relationship between
  - ◇ specifications to formalize the steps of gathering requirements in requirements engineering
  - ◇ specifications and architectures to formalize the steps in design of going from requirements to architecture
  - ◇ system specifications and implementations (e.g. by state machines)
  - ◇ levels of abstraction

Refinement

- Given systems/specifications S and S′ we write

  S is_refined_to S′ (and also S ≈> S′)

  to express that S′ is a refinement of S.
- Horizontal Refinement: Property Refinement
  - ◇ Adding properties to a specification – reducing the non-determinism of a system description
- Vertical Refinement: Interaction granularity refinement
  - ◇ Changing the granularity of the interaction
  - ◇ Interesting case: Refinements between continuous and discrete models

See: M. Broy: Compositional Refinement of Interactive Systems. Journal of the ACM, Volume 44, No. 6 (Nov. 1997), 850-891

Remark: Both refinement notions can be applied not only to interface models but to all kinds of models

Horizontal Refinement

$$F: \vec{I} \rightarrow \wp(\vec{O})$$

is refined by a b

$$\hat{F}: \vec{I} \rightarrow \wp$$

if

$$\forall x \in I: F .x \subseteq F.x$$

we write

$$F \approx\!>_{\mathbb{IF}} \hat{F}$$

Compositionality of refinement:
Modularity

$$\frac{\forall\, k: F_k \approx\!>_{\mathbb{IF}} \hat{F}_k}{\otimes\{F_k : k \in \mathbb{K}\} \approx\!>_{\mathbb{IF}} \otimes\{\hat{F}_k : k \in \mathbb{K}\}}$$

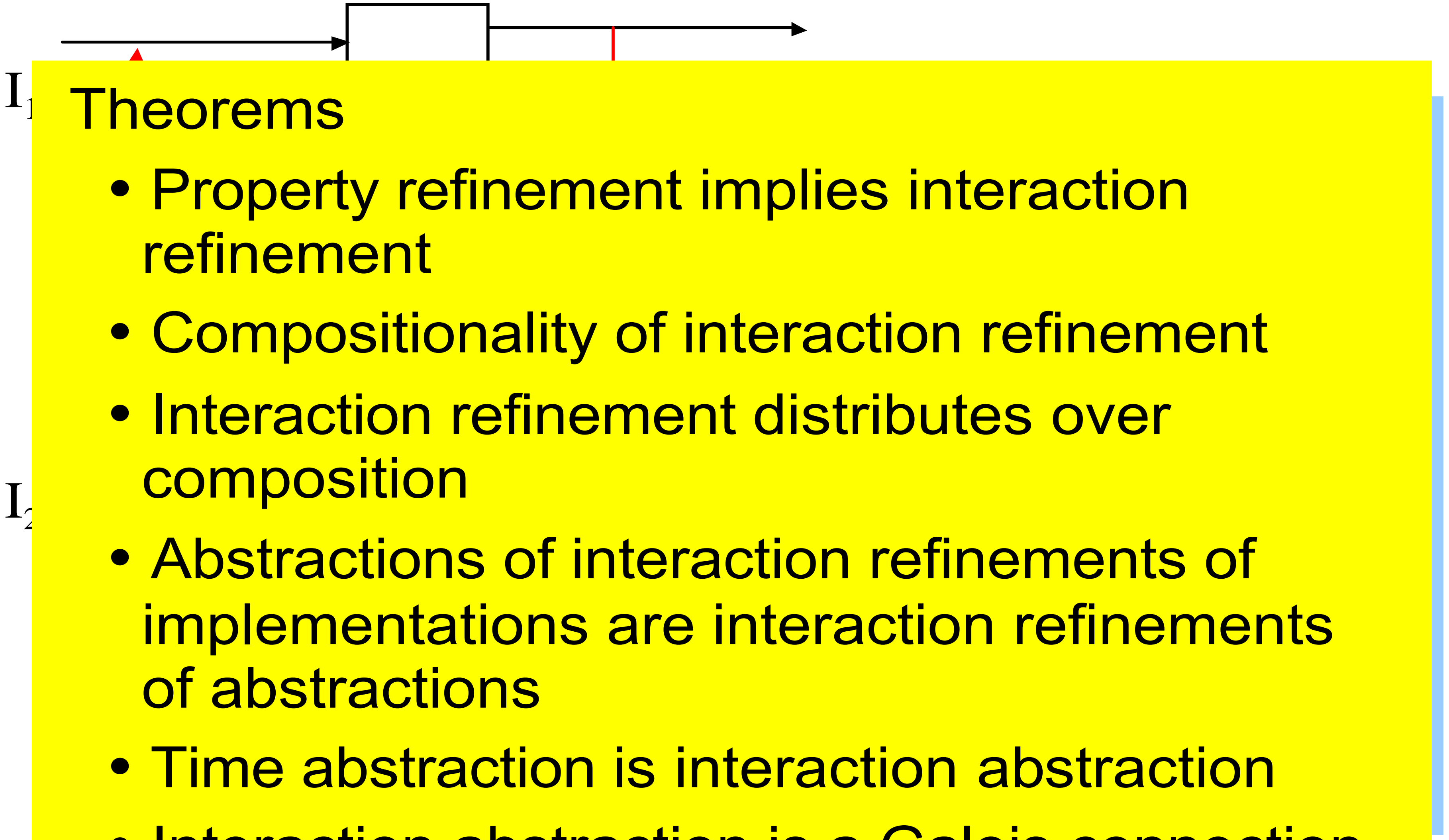
Verification of refinement steps

- A system F with behaviour assertion Q is refined by a system F′ with behaviour assertion Q′ if and only if

  $$Q \Leftarrow Q'$$

  In other words: F′ is a refinement of F if all properties of F are also properties of F′
- The implication $Q \Leftarrow Q'$ shows also how to verify the refinement relation

Vertical refinement: Levels of abstraction



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

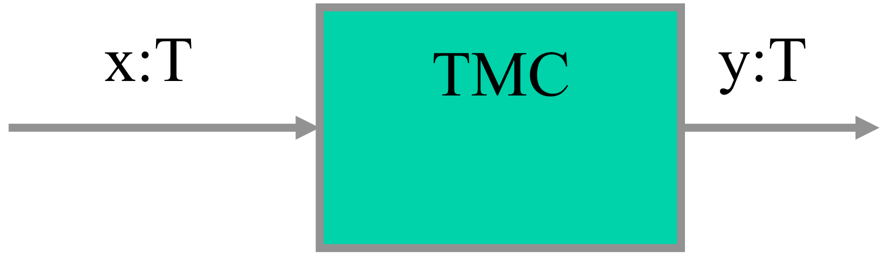
Compatibility

- A system S′ is called (replacement) compatible for system S if S′ can be used instead of S′ in every system M without violating the correctness of the system M
- Compatibility coincides with refinement in case of modular refinement

A flexible model of time

- Time is a key issue in embedded systems:
- Dealing with timing properties
  - ◇ Specification
  - ◇ Analysis
  - ◇ Verification
    - Analysis
    - Testing
    - Model checking
    - Deduction based verification
- Transforming time
- Dedicated models of time
  - ◇ Micro/Macro Step
  - ◇ Perfect synchrony
  - ◇ Scheduling
- Abstractions

Example: TMC with Timing Restrictions



TMC

in x: T **out** y: T
$\forall\ t \in \mathbb{N}: \forall\ m \in T:$ $m\#(x{\downarrow}t) \geq m\#(y{\downarrow}t+delay)$ $m\#(x{\downarrow}t) \leq m\#(y{\downarrow}t+delay+deadline)$

Conclusion Refinement

- Refinement formalises development steps
- Going from
  - an interface specification to an architecture (design step)
  - an interface specification to a state machine (implementation)

  can be understood as special steps of refinement
- Compatibility is defined by refinement, too
- The change of time granularity is a refinement

Implementation: Systems as State Machines

The State View

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

System and States

- Systems have states
- A state is an element of a state space
- We characterize state spaces by
  - a set of state attributes together with their types
- The behaviour of a system with states can be described by its state transitions

State model for systems/components

A system can be implemented by a state Machine

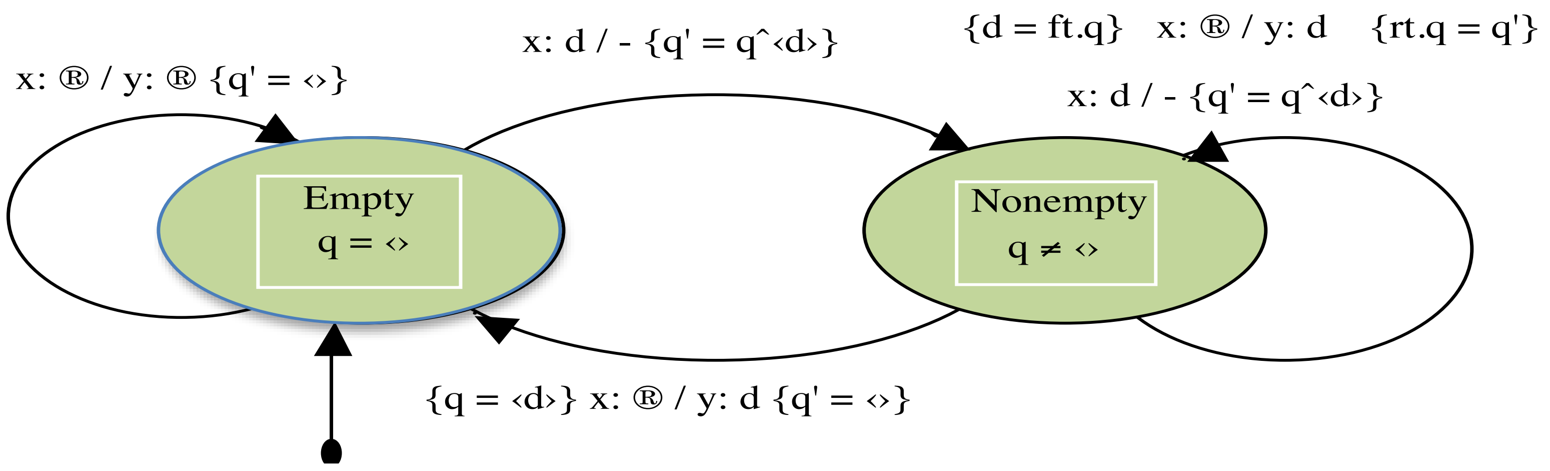
M. Broy: From States to Histories: Relating States and History Views onto Systems. In: T. Hoare, M. Broy, R Steinbrüggen (eds.): Engineering Theories of Software Construction. Springer NATO ASI Series, Series F: Computer and System Sciences, Vol. 180, IOS 2001, 149-186

Σ 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 transition diagram:



State Machines in general

A state machine (Δ, Λ) consists of

- a set Σ of states - the state space
- a set $\Lambda \subseteq \Sigma$ of initial states
- a state transition function or relation Δ
  - ◇ in case of a state machine with input/output:

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

    $$\Delta : \Sigma \times \mathrm{E} \to \Sigma \times \mathrm{A}$$

    in the case of nondeterministic machines:

    $$\Delta : \Sigma \times \mathrm{E} \to \wp(\Sigma \times \mathrm{A})$$

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

$$\mathrm{E} = \mathrm{I} \to \mathrm{M}^*$$

$$\mathrm{A} = \mathrm{O} \to \mathrm{M}^*$$

Computations of a State Machine with Input/Output

A state machine (Δ, Λ) 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

$$(\sigma_{i+1}, a_{i+1}) \in \Delta(\sigma_i, e_{i+1})$$

Computations of a State Machine with Input/Output

In this manner we obtain computations of the form

$$\sigma_0 \xrightarrow{a_1 / b_1} \sigma_1 \xrightarrow{a_2 / b_2} \sigma_2 \xrightarrow{a_3 / b_3} \sigma_3 \quad \ldots$$

For each initial state $\sigma 0 \in \Sigma$ we define a function

$$F_{\sigma 0} : \vec{I} \to \wp(\vec{O})$$

with

$$F_{\sigma 0}(x) = \{y: \exists\ \sigma_i: \sigma 0 = \sigma_0 \wedge \forall\ i \in IN: (\sigma_{i+1}, y_{i+1}) = \Delta(\sigma_i, x_{i+1})\}$$

$F_{\sigma 0}$ denotes the interface behavior of the transition function Δ for the initial state $\sigma 0$.

Furthermore we define

$$Abs((\Delta, \Lambda)) = F_\Lambda$$

where:

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

F_Λ is called the interface behavior of the state machine (Δ, Λ) .

Moore Machines

- A Mealy machine (Δ, Λ) with

$$\Delta : \Sigma \times E \to \wp(\Sigma \times A)$$

  is called Moore machine if for all states $\sigma \in \Sigma$ and 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 state σ.

- Formally: then for all $e, e' \in E$ we have

$$\text{out}(\sigma, e) = \text{out}(\sigma, e')$$

Theorem: If is (Δ, Λ) a Moore machine the F_Λ is strong causal.

Interface Abstraction for State Machines

- For a given state machine with input and output we define the interface through
  - ◇ its syntactical interface (signature)
  - ◇ its interface behavior

- We call the transition of the state machine to its interface the interface abstraction.

Verification/derivation of interface assertions for state machines

- similar to program verification (find an invariant)
- needs sophisticated techniques

Observable Equivalence

- Two systems modelled by state machines

$$(\Delta 1, \Lambda 1) \text{ and } (\Delta 2, \Lambda 2)$$

  are observably equivalent iff they fulfil the equation

$$\text{Abs}((\Delta 1, \Lambda 1)) = \text{Abs}((\Delta 2, \Lambda 2))$$

Composition of the two state machines

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

$\Delta_k: \Sigma_k \times (I_k \rightarrow M^*) \rightarrow \wp(\Sigma_k \times (O_k \rightarrow M^*))$

We define the composed state machine

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

as follows

$\Sigma = \Sigma_1 \times \Sigma_2$

for $x \in I$ and $(s_1, s_2) \in \Sigma$ we define:

$$\Delta((s_1, s_2), x) = \{((s_1', s_2'), z|O): \ x = z|I \wedge \forall k: (s_k', z|O_k) \in \Delta_k(s_k, z|I_k) \}$$

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

$\Delta = \Delta_1 \,||\, \Delta_2$

$M = M_1 \,||\, M_2 = (\Delta_1 \,||\, \Delta_2 , \Lambda_1 \times \Lambda_2)$

An example of an essential property ...

Interface abstraction distributes for state machines over composition

$$\mathrm{Abs}((\Delta 1, \sigma 1) \,||\, (\Delta 2, \sigma 2)) =$$

$$\mathrm{Abs}((\Delta 1, \sigma 1)) \otimes \mathrm{Abs}((\Delta 2, \sigma 2))$$

Conclusion Systems as State Machines

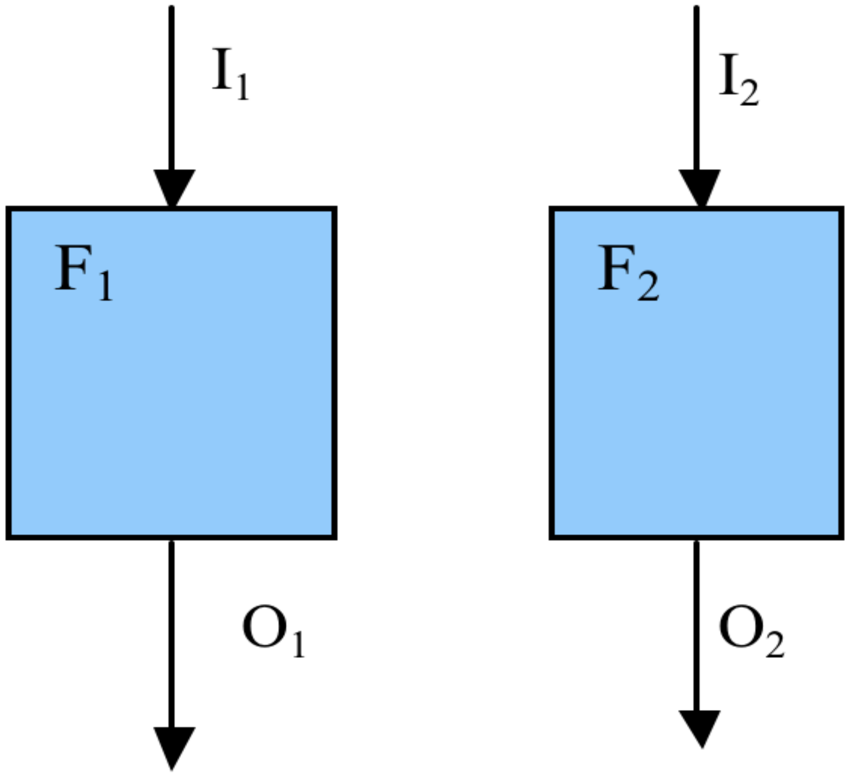
- Each state machines defines an interface behaviour
- Each interface behaviour represents a state machine
- State machines can be described
    - ◇ mathematically by their state transition function
    - ◇ graphically by state machine diagrams
    - ◇ structured by state transition tables
    - ◇ by programs
- State machines define a kind of operational semantics
- Systems given by state machines can be simulated
- From state machines we can generate code
    - ◇ state machines can represent implementations
- From state machines we can generate test cases

Functional View: Functional Decomposition

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Combining Functions

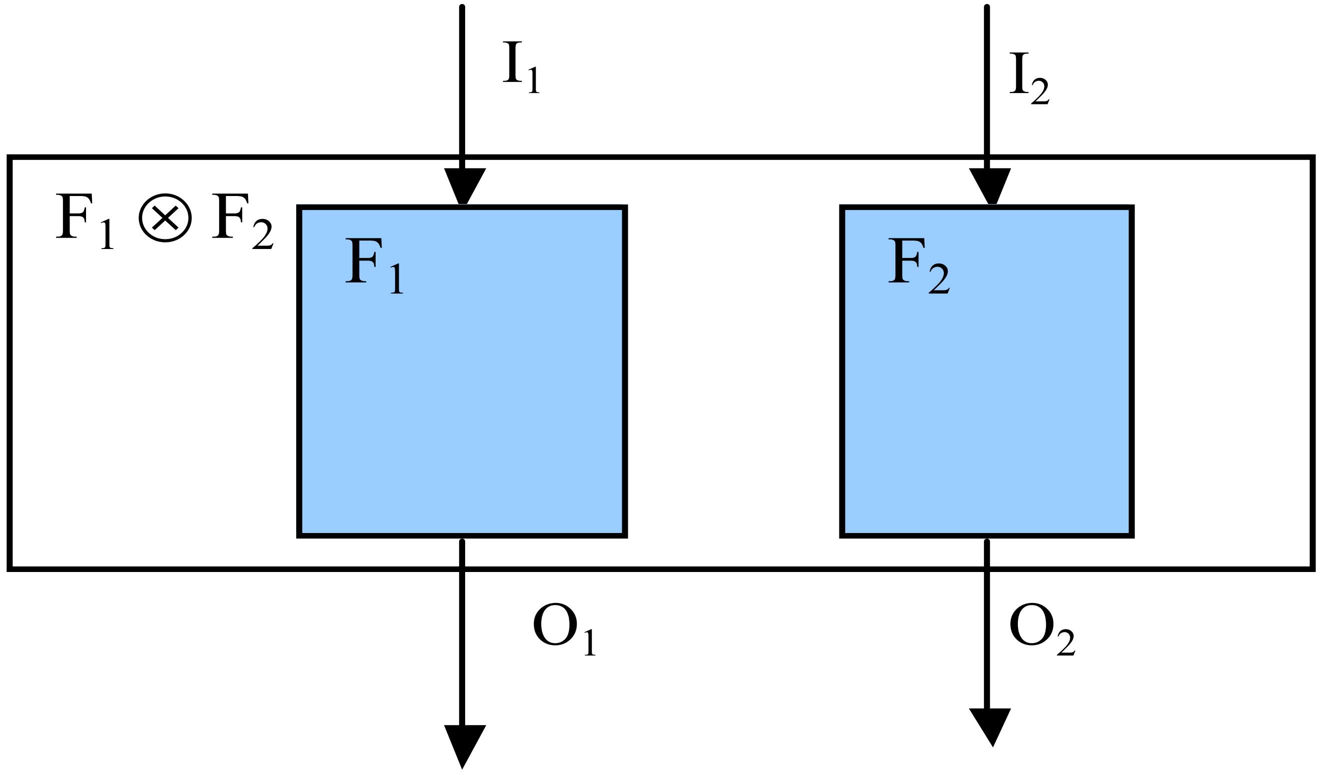
Given two functions F_1 and F_2 in isolation



We want to combine them into a function $F_1 \otimes F_2$

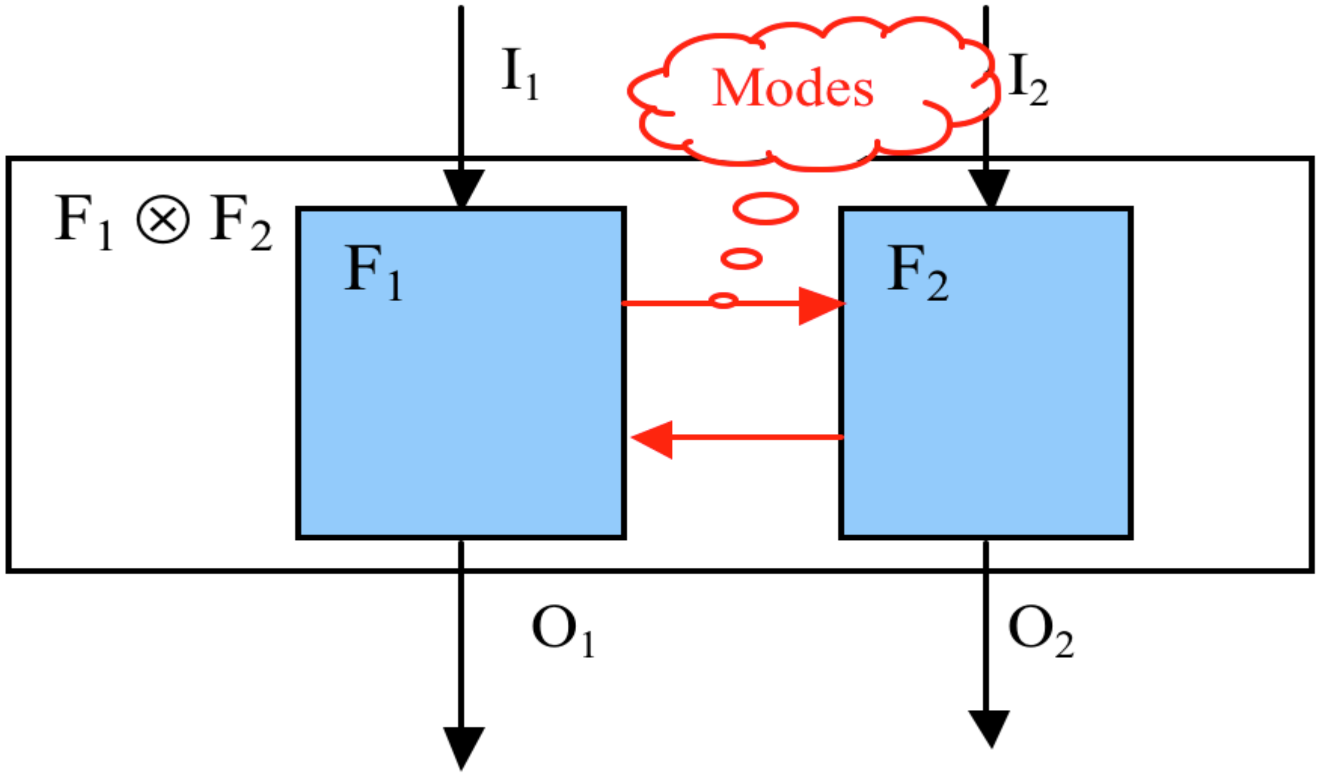
Combining Functions

Their isolated combination



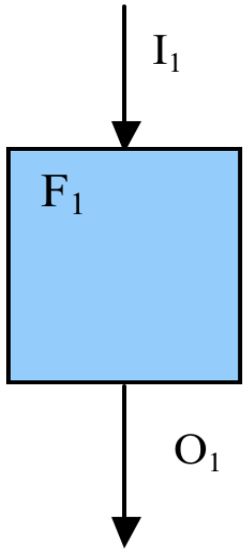
Combining Functions

If services F_1 and F_2 have feature interaction we get:

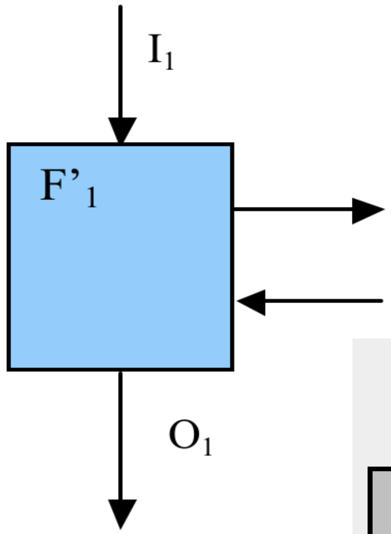


We explain the functional combination $F_1 \otimes F_2$ as a refinement step

The steps of function combination

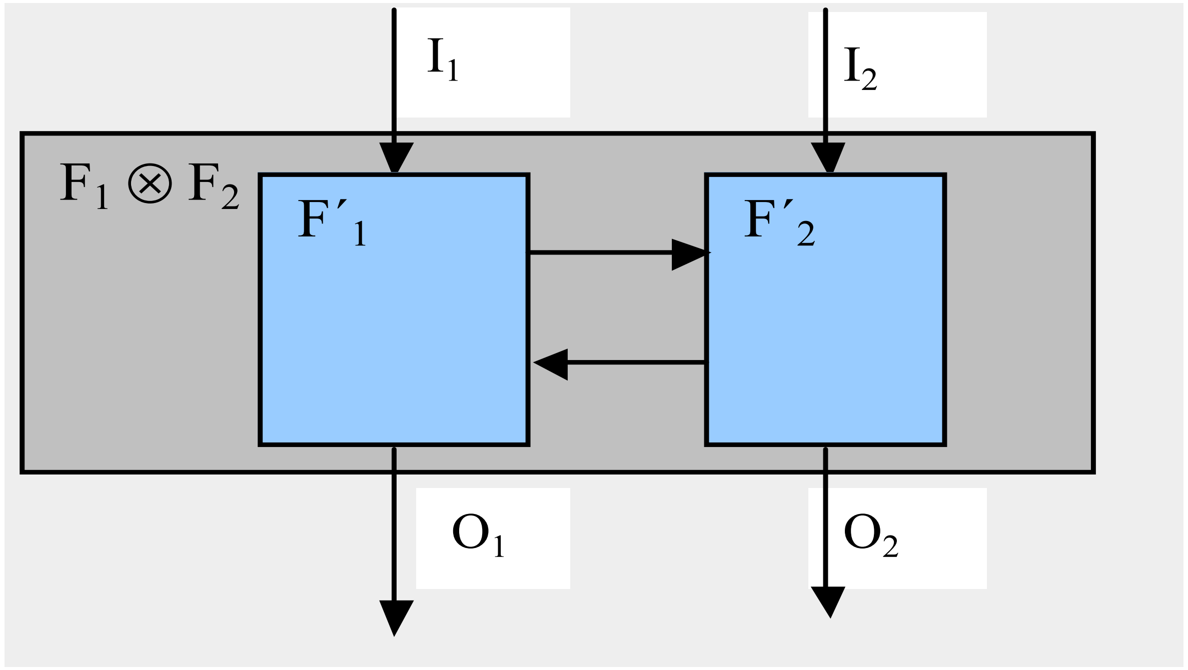


Given the isolated function F_1



We construct a refinement F'_1

And combine F'_1 with a refinement F'_2 of F_2



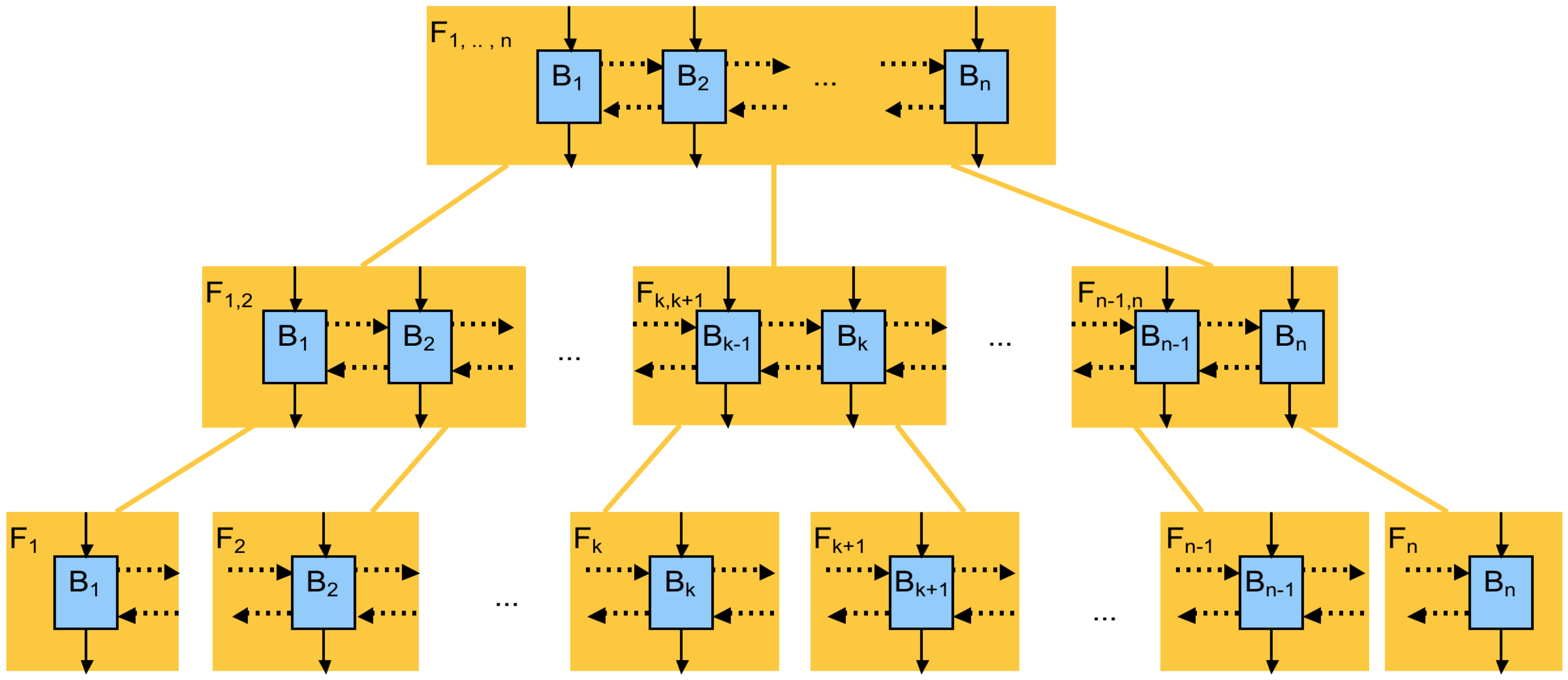
See: M. Broy: Multifunctional Software Systems: Structured Modeling and Specification of Functional Requirements. Science of Computer Programming 75 (2010), S. 1193–1214

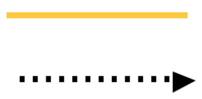
Functional architecture: functional decomposition

- The system interface behaviour F as specified by the system requirements specification $A = \{A_i: 1 \le i \le n\}$ is structured
  - ◇ into a set of sub-interfaces for sub-functions $F_1, \ldots, F_k$
  - ◇ that are specified independently by introducing a number of mode channels to capture their feature interactions
  - ◇ each F_i sub-function is described by
    - a syntactic interface and
    - an interface assertion B_i such that

$$\wedge \{B_i: 1 \le i \le k\} \Rightarrow A$$

Function Hierarchy





$\leftarrow_{sub}$ subservice relation

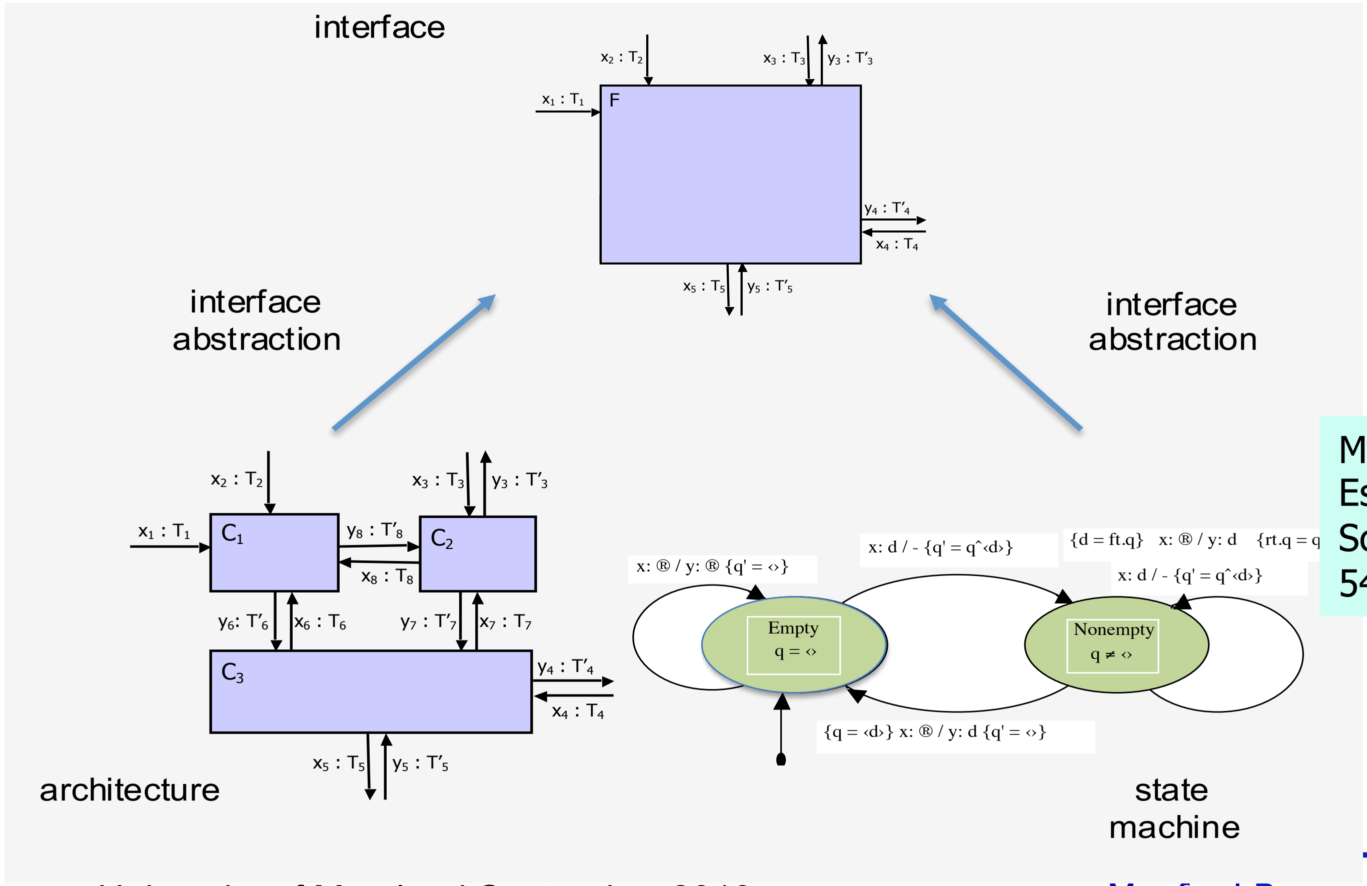
channels of mode types

Model Integration

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Integrating modeling concepts

- An architecture can be abstracted into an interface behavior
  - ◇ Proof techniques for architecture verification
- A state machine can be abstracted into an interface behavior
  - ◇ Proof techniques for implementation verification



M. Broy: The Semantic and Methodological Essence of Message Sequence Charts. Science of Computer Programming, SCP 54:2-3, 2004, 213-256

Extensions of the model: Probability

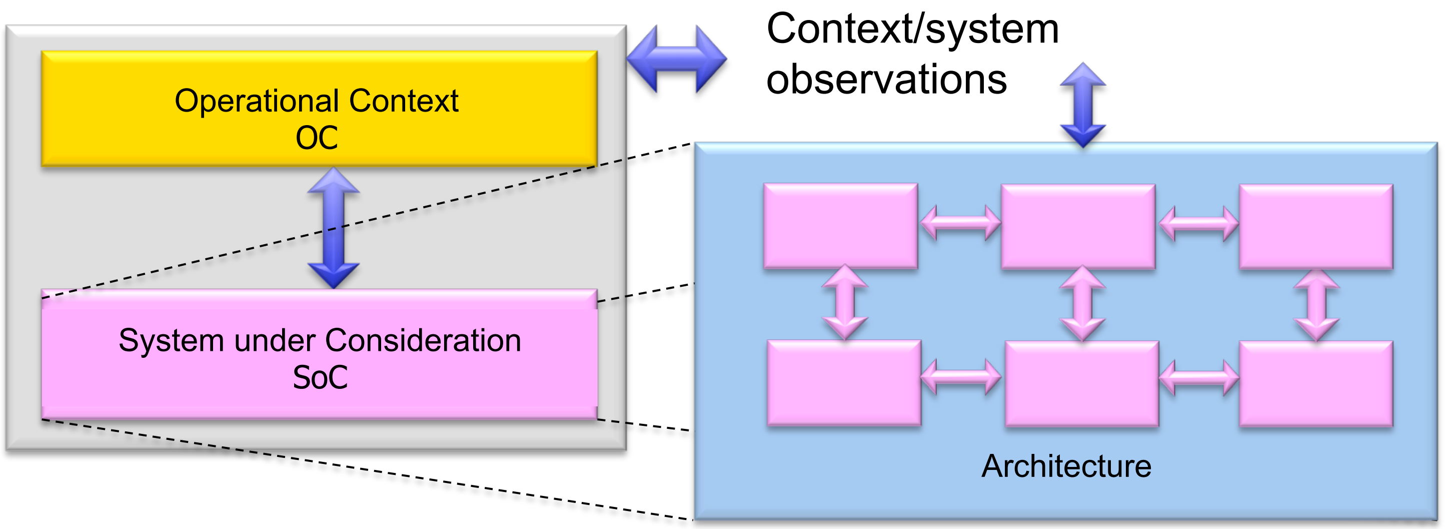
- Probabilistic views
  - ◇ Interface behavior: a probability distribution is given for the set of possible histories
  - ◇ Architectural view: probability distributions for the sub-systems of the architecture
  - ◇ State view: a probability distribution is given for the set of possible state transitions
- Then the model covers
  - ◇ certain "non-functional properties" (safety, reliability, …)
  - ◇ Example: integrated fault trees

See: P. Neubeck: A Probabilitistic Theory of Interactive Systems. PH. D. Dissertation, Technische Universität München, Fakultät für Informatik, December 2012

A full System Perspective

Requirements
- Observations
- Context
  - Properties
- Interface

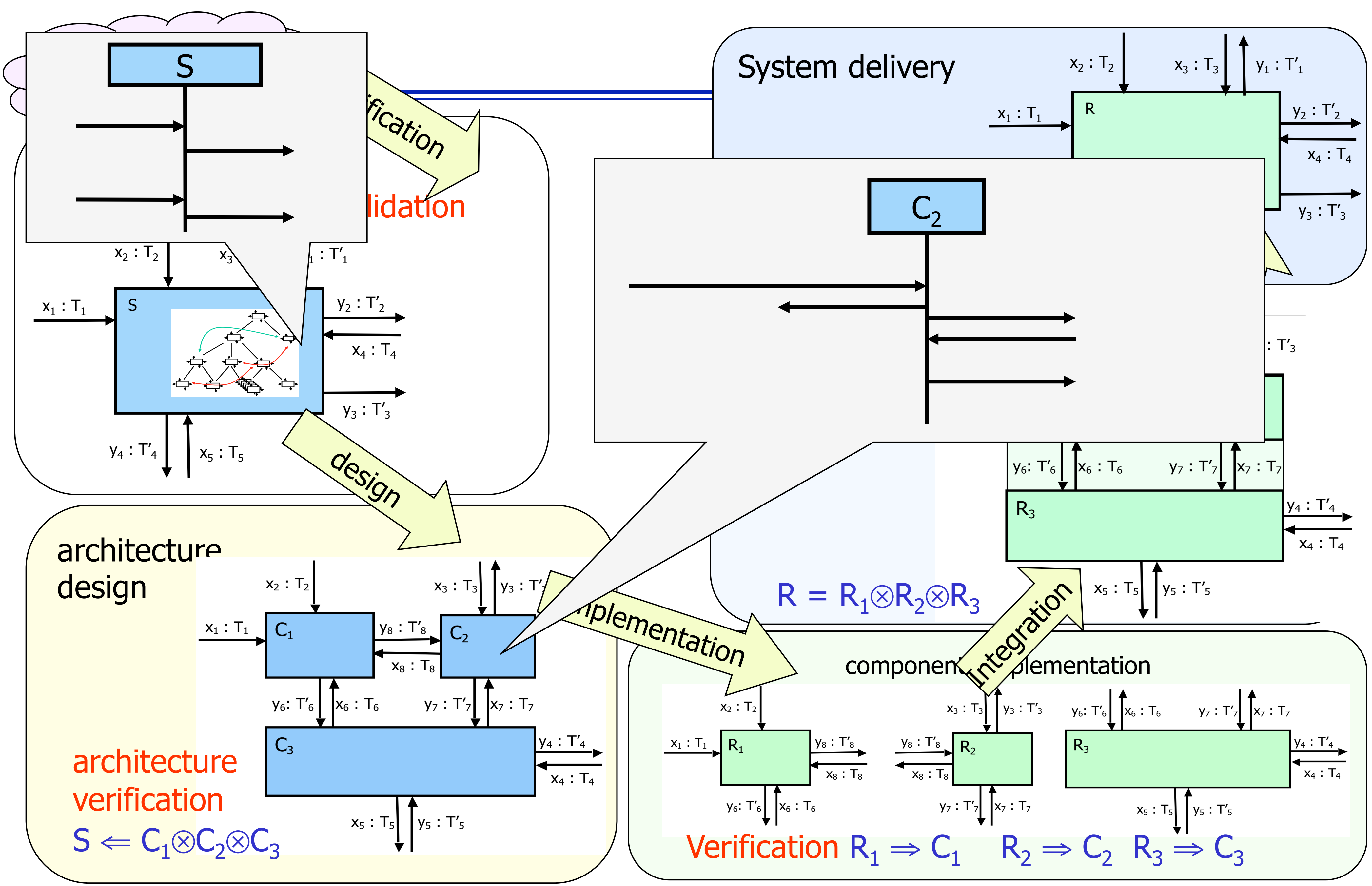
- Architecture
  Sub-systems
  - Connections
  - Interaction
  - Interfaces



System development proceeds in working out a sequence of perspectives at several levels of abstraction that are related
- by refinement
- by decomposition
- by changing the scope/system boundary
- by embedding

Modular Model Based System Development

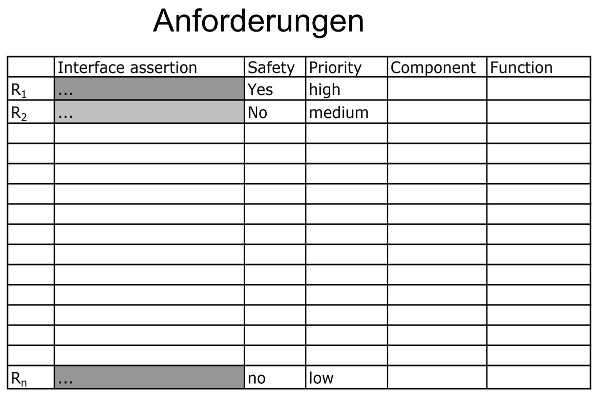
Technische Universität München
Institut für Informatik
D-80290 Munich, Germany


S
ification
lidation
$x_2 : T_2$
x_3
$_1 : T'_1$
$x_1 : T_1$
S
$y_2 : T'_2$
$x_4 : T_4$
$y_3 : T'_3$
$y_4 : T'_4$
$x_5 : T_5$
design
System delivery
$x_2 : T_2$
$x_3 : T_3$
$y_1 : T'_1$
$x_1 : T_1$
R
$y_2 : T'_2$
$x_4 : T_4$
$y_3 : T'_3$
C_2
$: T'_3$
$y_6 : T'_6$
$x_6 : T_6$
$y_7 : T'_7$
$x_7 : T_7$
R_3
$y_4 : T'_4$
$x_4 : T_4$
$x_5 : T_5$
$y_5 : T'_5$
$R = R_1 \otimes R_2 \otimes R_3$
architecture design
$x_2 : T_2$
$x_3 : T_3$
$y_3 : T'_3$
$x_1 : T_1$
C_1
$y_8 : T'_8$
C_2
$x_8 : T_8$
$y_6 : T'_6$
$x_6 : T_6$
$y_7 : T'_7$
$x_7 : T_7$
C_3
$y_4 : T'_4$
$x_4 : T_4$
$x_5 : T_5$
$y_5 : T'_5$
architecture verification
$S \Leftarrow C_1 \otimes C_2 \otimes C_3$
nplementation
Integration
component implementation
$x_2 : T_2$
$x_1 : T_1$
R_1
$y_8 : T'_8$
$x_8 : T_8$
$y_6 : T'_6$
$x_6 : T_6$
$x_3 : T_3$
$y_3 : T'_3$
$y_8 : T'_8$
R_2
$x_8 : T_8$
$y_7 : T'_7$
$x_7 : T_7$
$y_6 : T'_6$
$x_6 : T_6$
$y_7 : T'_7$
$x_7 : T_7$
R_3
$y_4 : T'_4$
$x_4 : T_4$
$x_5 : T_5$
$y_5 : T'_5$
Verification $R_1 \Rightarrow C_1$ $R_2 \Rightarrow C_2$ $R_3 \Rightarrow C_3$

Comprehensive System Architecture
Levels of Abstraction

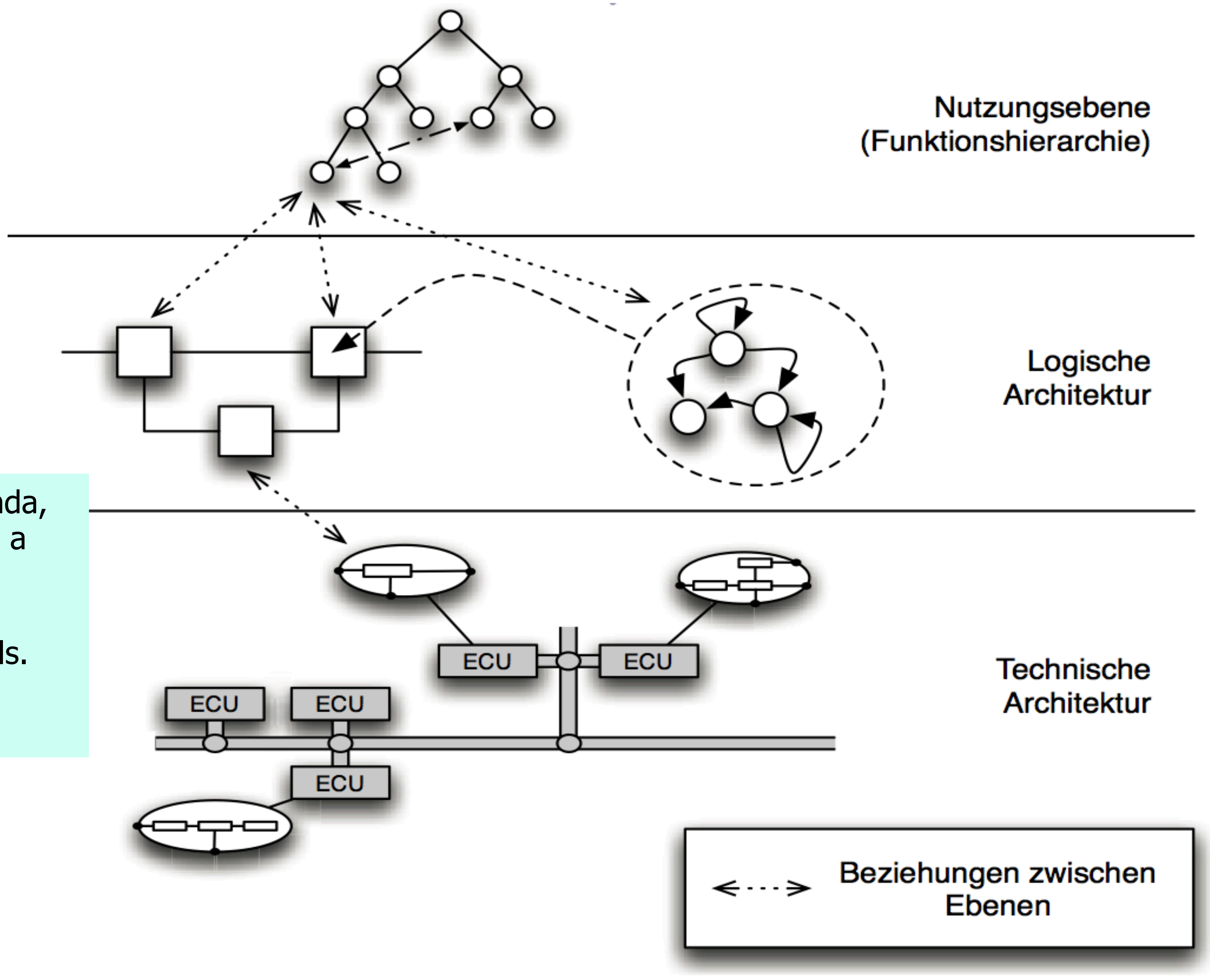
Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Structuring Systems: levels of abstraction



Anforderungen

	Interface assertion	Safety	Priority	Component	Function
R_1	...	Yes	high		
R_2	...	No	medium		
R_n	...	no	low		



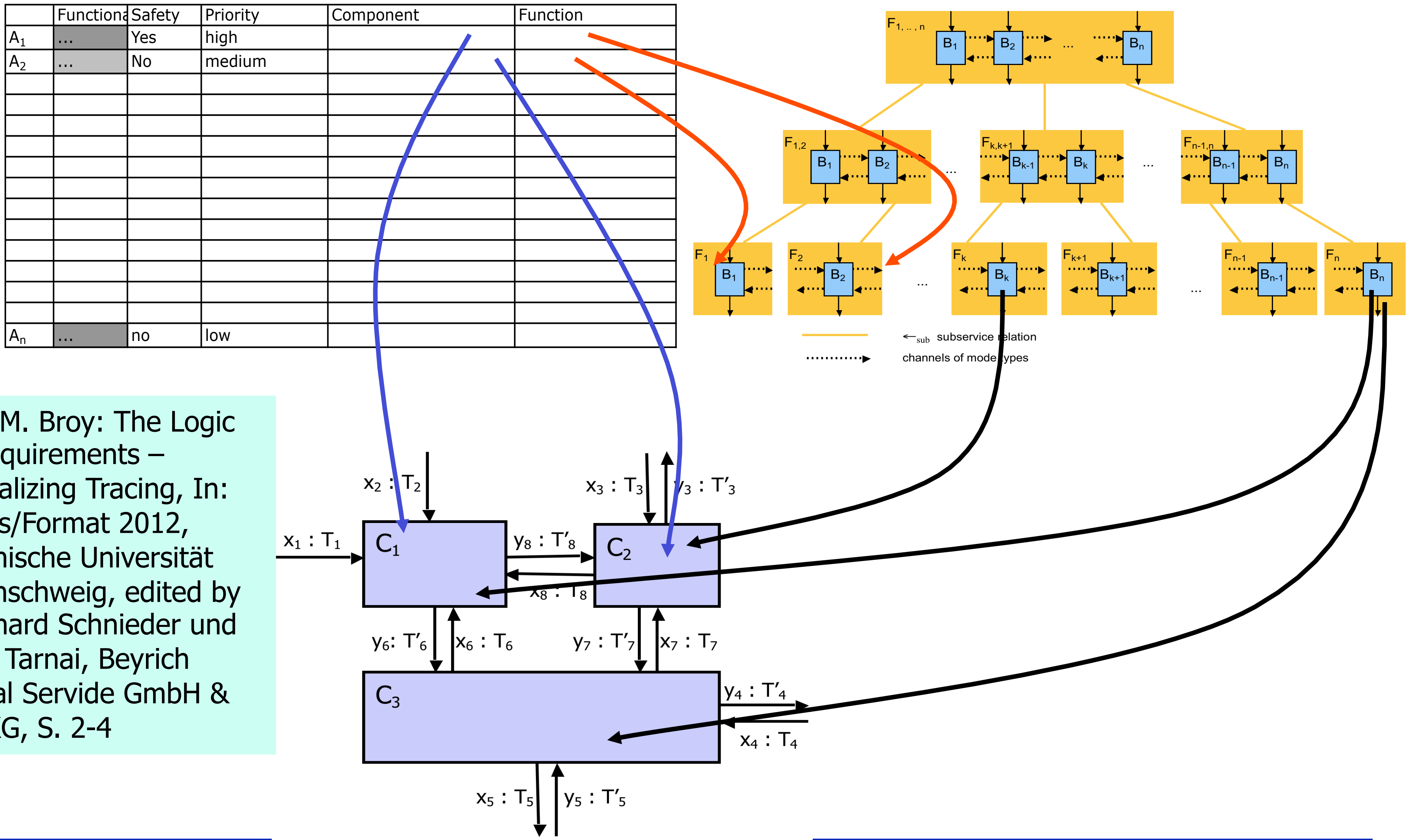
Nutzungsebene (Funktionshierarchie)
Logische Architektur
Technische Architektur
ECU
ECU
ECU
ECU
ECU
Beziehungen zwischen Ebenen

See: M. Broy, M. Gleirscher, St. Merenda, D. Wild, P. Kluge, W. Krenzer: Toward a Holistic and Standardized Automotive Architecture Description. Innovative Technology for Computer Professionals. Computer, IEEE Computer Society. December 2009, S. 98-103

Goals and Requirements and Functional Specification

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

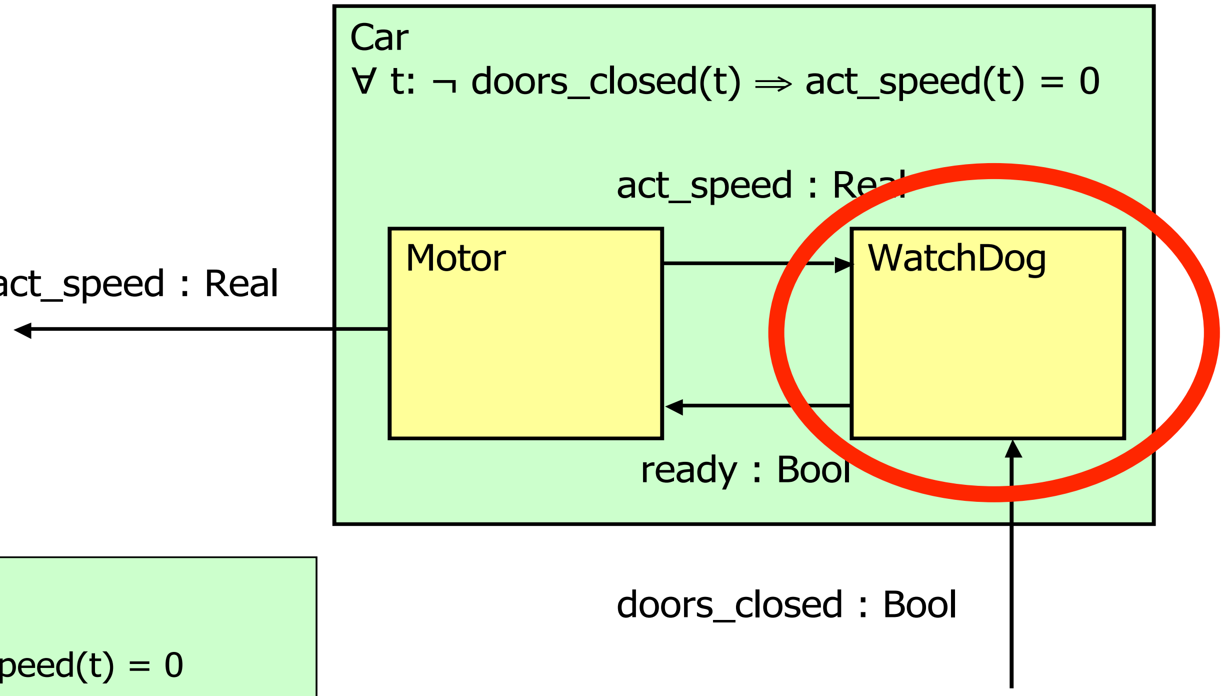
Relational view: Tracing – The power of logics

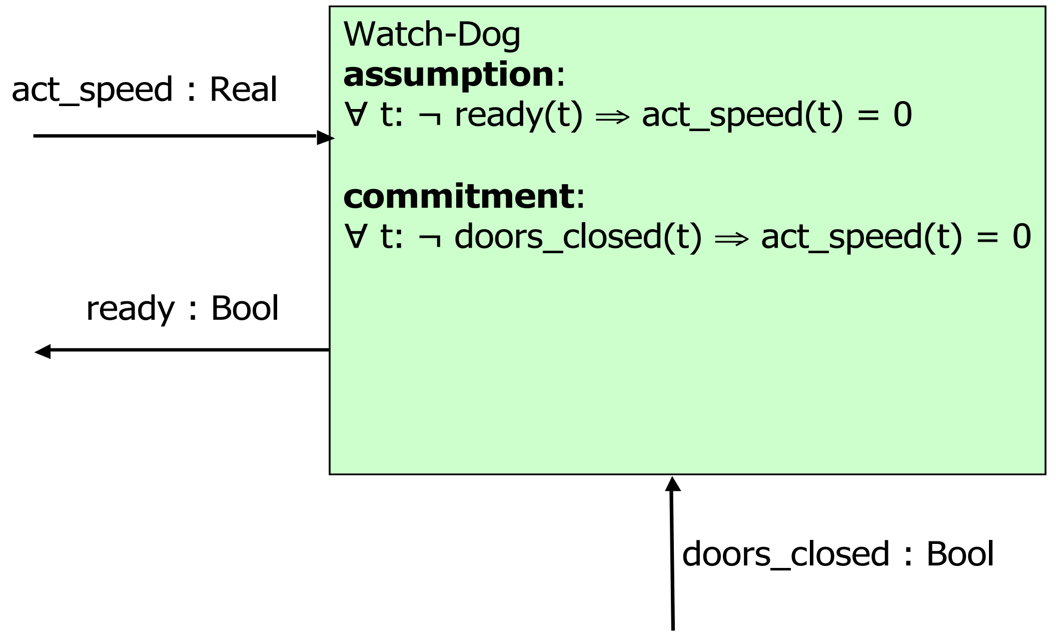


	Functiona	Safety	Priority	Component	Function
A_1	...	Yes	high		
A_2	...	No	medium		
A_n	...	no	low		

See: M. Broy: The Logic of Requirements – Formalizing Tracing, In: Forms/Format 2012, Technische Universität Braunschweig, edited by Eckehard Schnieder und Géza Tarnai, Beyrich Digital Servide GmbH & Co. KG, S. 2-4

Glass Box Specification of a Car´s Architecture





Watch-Dog
assumption:
∀ t: ¬ ready(t) ⇒ act_speed(t) = 0

commitment:
∀ t: ¬ doors_closed(t) ⇒ act_speed(t) = 0

See: M. Broy: Towards a Theory of Architectural Contracts: - Schemes and Patterns of Assumption/ Promise Based System Specification. In: M. Broy, Ch. Leuxner, T. Hoare (Eds.): Software and Systems Safety - Specification and Verification. NATO Science for Peace and Security Series - D: Information and Communication Security 30, IOS Press 2011, 33-87

Example: how A/P-specifications can be formulated

- The specification

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0

  can only be guaranteed if the two inner components work together. This requires

  $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0
- Then the system specification holds if

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ $\neg$ ready(t)
- This is logically equivalent to the A/P-specification for the WatchDog

  assumption: $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0

  commitment: $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0
- In other words,
  - ◇ the overall system specification can be guaranteed by the watchdog
  - ◇ only if the assumption about the behaviour of the component motor holds.

Assumption/Promise to define Architectural Design Patterns

- A/P-specification

  assumption: $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0

  commitment: $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ act_speed(t) = 0

  is logically guaranteed by the simple specification

  $\forall$ t: $\neg$ doors_closed(t) $\Rightarrow$ $\neg$ ready(t)
- This assertion no longer speaks about the specification of the environment, but is a pure interface specification.
- The example shows the simplification of an A/P-specification to a plain interface assertion.

Conclusion A/C

- A/C specs address the logic of the architecture rather than separated interface specifications
- From A/C specs we may derive simplified component specs
- This gives a methodology towards a modular decomposition

Key Principles in Engineering CPSs

- Abstraction
  - Interfaces
  - Changing levels of abstractions
- Modularity
  - Of composition
  - Of refinement
- Semantic coherence
  - From state machines to interfaces
  - From architectures to interfaces
  - From architectures of state machines to state machines of architectures
  - Probabilistic extension of logical (deterministic) models
- Expressiveness
  - Set theoretic and logical: discrete system models
  - Continuous models: control theory
  - Probabilistic: probability distributions on behavior

The Triad of Modeling

- Denotational:
  - mathematical models (of behavior)
- Logical – system properties:
  - specification
  - deduction
  - verification
  - transformation
- Notational:
  - Graphical: diagrams
  - Tables
  - Formulas

Concluding Remarks

- The modelling framework FOCUS
  - originally worked out for model based development
  - specification
  - verification
  - tool support
- Tool: Autofocus 3
- Also useful for semantic foundation following the same approach
  - SDL
  - Bus Systems: CAN, FLEXRAY
  - UML/SysML
  - SOA

See: https://af3.fortiss.org/projects/autofocus3

See: K. Pohl, H. Hönninger, R. Achatz, M. Broy: The SPES 2020 Methodology. Springer Verlag 2012

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 of technical aspects from application aspects
- Technical architectures are modelled along the same theory
- Code and test cases can be generated from the models

The power of generalizing ideas, of drawing comprehensive conclusions from individual observations, is the only acquirement, for an immortal being, that really deserves the name of knowledge.

“Mary Wollstonecraft (1759–1797), British feminist. A Vindication of the Rights of Woman, ch. 4 (1792)