Towards a Theory of Architectural Contracts:
Schemes and Patterns of Assumption/Promise Based System Specification

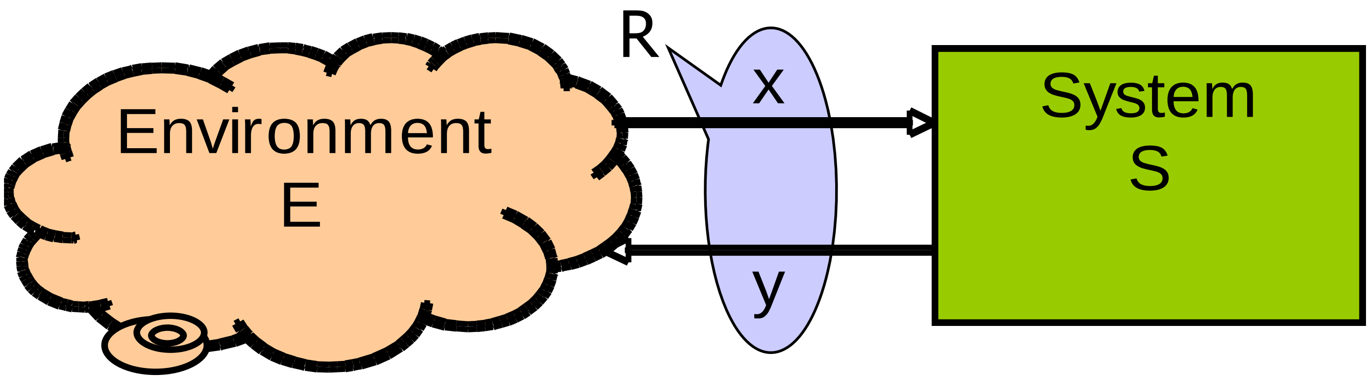
Manfred Broy

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

Contracts and Architectures

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

From interaction assertions to contracts



- Let R(x, y) be an assertion that characterizes the interaction between system S and its environment E, called interaction assertion.
- R provides an observation/specification of the traffic between E and S
- Can we derive of a contract for S from assertion R(x, y) that captures the obligations of system S w.r.t. R?

Exceptional cases

- It is clear that we cannot expect to get a reasonable contract from R(x, y) in every case.
- A simple example would be R(x, y) = false.

Separation

- Given the healthiness condition for the interface assertion

  $\exists$ x, y: R(x, y)

  we can do a separation of R into an assumption and a promise (for the safety properties in R) as follows.
- We specify the responsibilities of the system S that accepts the input history x and issues and output history y such that assertion

  R(x, y)

  holds.
- We are looking for assertions asu(x, y) and pro(x, y) such that

  $\text{asu}(x, y) \wedge \text{pro}(x, y) \Rightarrow R(x, y)$

  and

  assumption asu(x, y)
  promise pro(x, y)

  is a healthy assumption promise specification.
- If R(x, y) is strongly causal in x and fully realizable, then $\text{asu}(x, y) \equiv \text{true}$ is a valid choice.

Separation

- If

  $$\forall\, x: \exists\, y: R(x, y)$$

  does not hold, then we need to construct an assumption $asu(x, y)$ and a promise $pro(x, y)$ such that

(1) $asu(x, y)$ is causal in y and realizable

(2) $asu(x, y) \Rightarrow pro(x, y)$ is strongly causal in x and realizable.

and

$$asu(x, y) \wedge pro(x, y) \Rightarrow R(x, y)$$

Example. From interaction to interface assertions

- Given the specification

$$R(x, y) \equiv (x \approx y \wedge \forall\ t: (\#y{\downarrow}t)+b \geq \#x{\downarrow}t \wedge \#x{\downarrow}t \geq \#y{\downarrow}t)$$

  where x and y are streams of data and b is a given number and $x \approx y$ specifies that x and y carry the same stream of messages (eliminating empty slots "-")
- We choose the assumption

$$asu(x, y) \equiv \forall\ t: (\#y{\downarrow}t)+b \geq \#x{\downarrow}t$$

- $asu(x, y)$ is causal in y and realizable.
- We choose the promise

$$pro(x, y) \equiv (x \approx y \wedge \forall\ t: \#x{\downarrow}t \geq \#y{\downarrow}t)$$

- Assertion $pro(x, y)$ is strongly causal and fully realizable.
- We get

$$pro(x, y) \wedge asu(x, y) \Rightarrow R(x, y)$$

Example. Non-realizable Specification

- Consider a system with input channel x and output channel y both carrying boolean messages:

$$R(x, y) = [\ (\text{true}\#x < \infty \Rightarrow \text{true}\#y = \infty) \wedge (\text{true}\#x = \infty \Rightarrow \text{true}\#y < \infty)\ ]$$

- All the involved assertions are liveness properties. We get

$$\forall\ x: \exists\ y: R(x, y)$$

- However, there does not exist a causal function f with

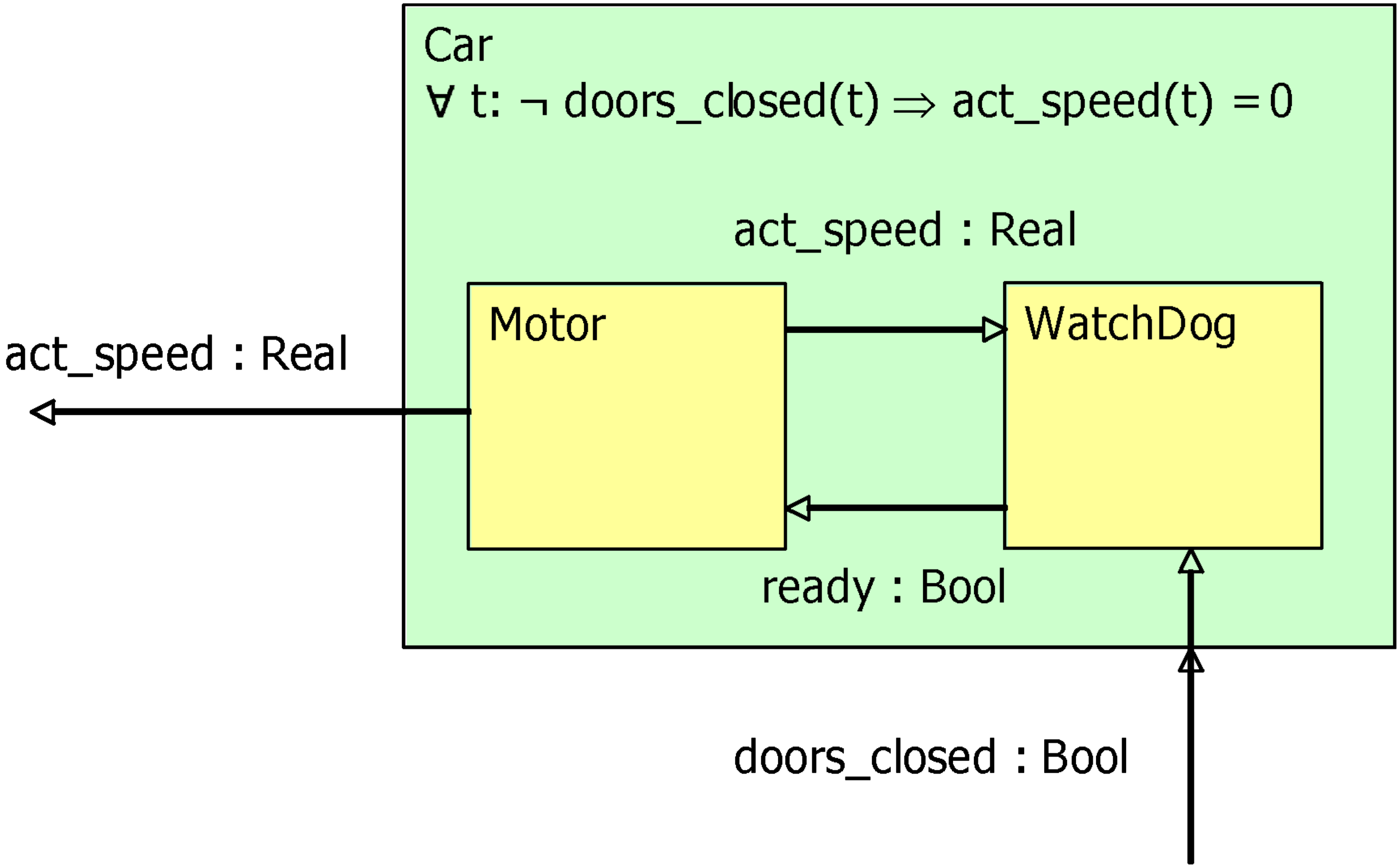
$$\forall\ x: R(x, f(x))$$

- Otherwise, there would exist a strongly causal function f with a fixpoint $y = f(y)$ such that $R(x, y)$ holds which delivers a contradiction.
- The example suggests that non-realizable specifications include liveness properties that cannot be realized.

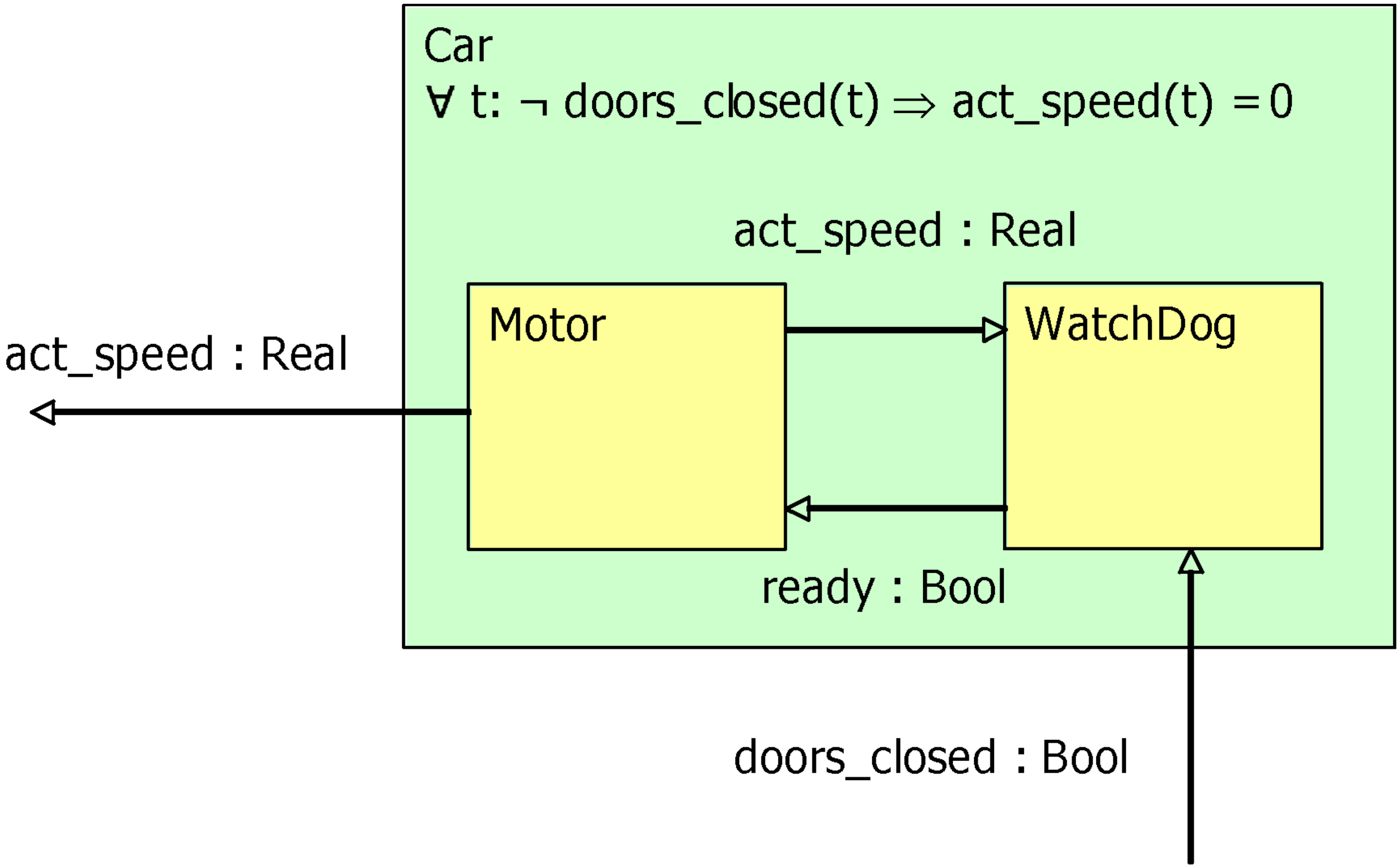
Assumptions in Architectural Modelling

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

Example: Simple Watch Guard in a Car



Glass Box Specification of a Car´s Architecture



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

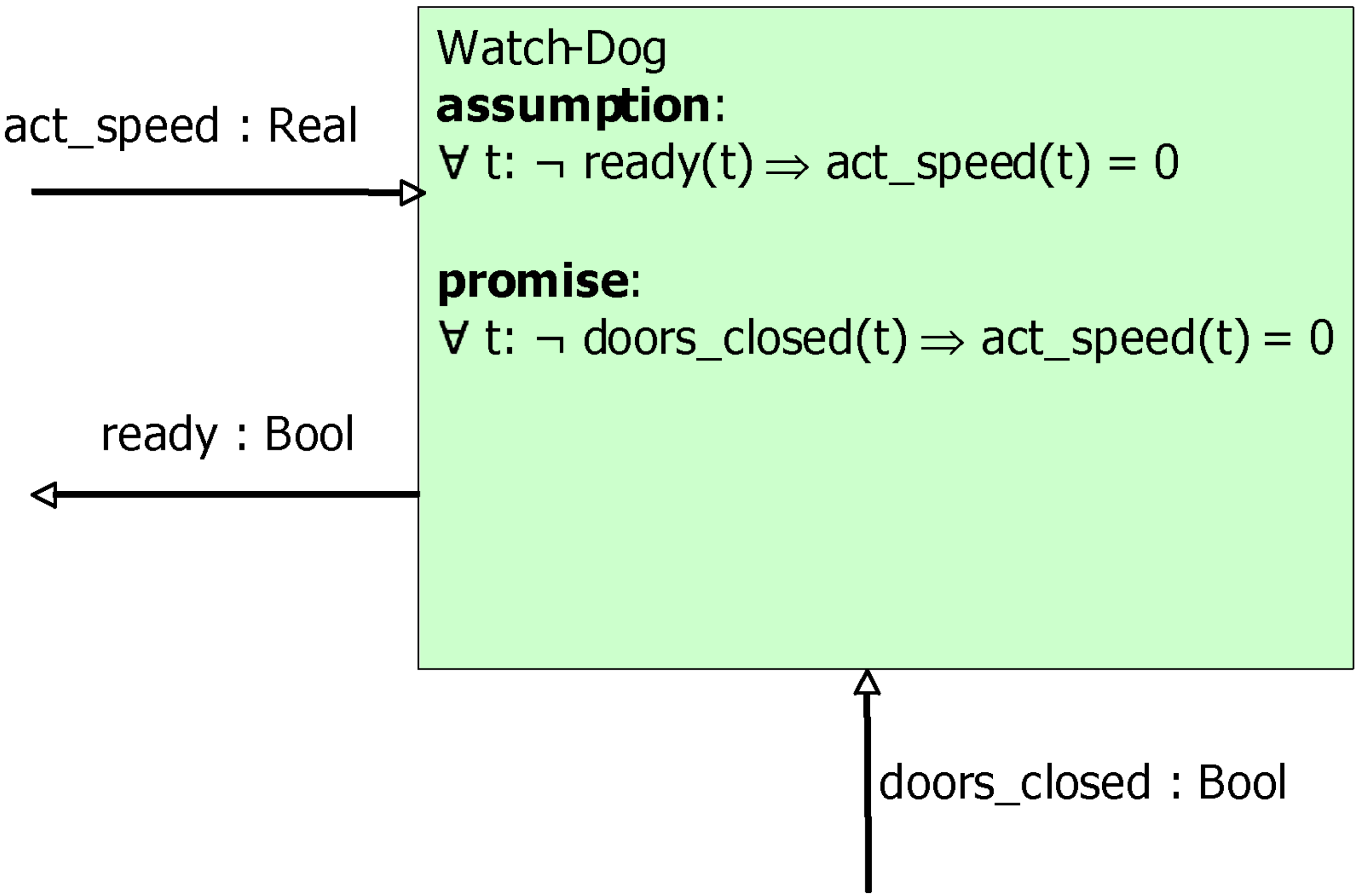
  $\forall$ t: $\neg$ ready(t) $\Rightarrow$ act_speed(t) = 0
- Then the system specification holds if

  $\forall$ 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

  promise: $\forall$ 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.

Simple Watch Guard in a Car (Continued)



Assumption/Promise to define Architectural Design Patterns

- A/P-specification

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

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

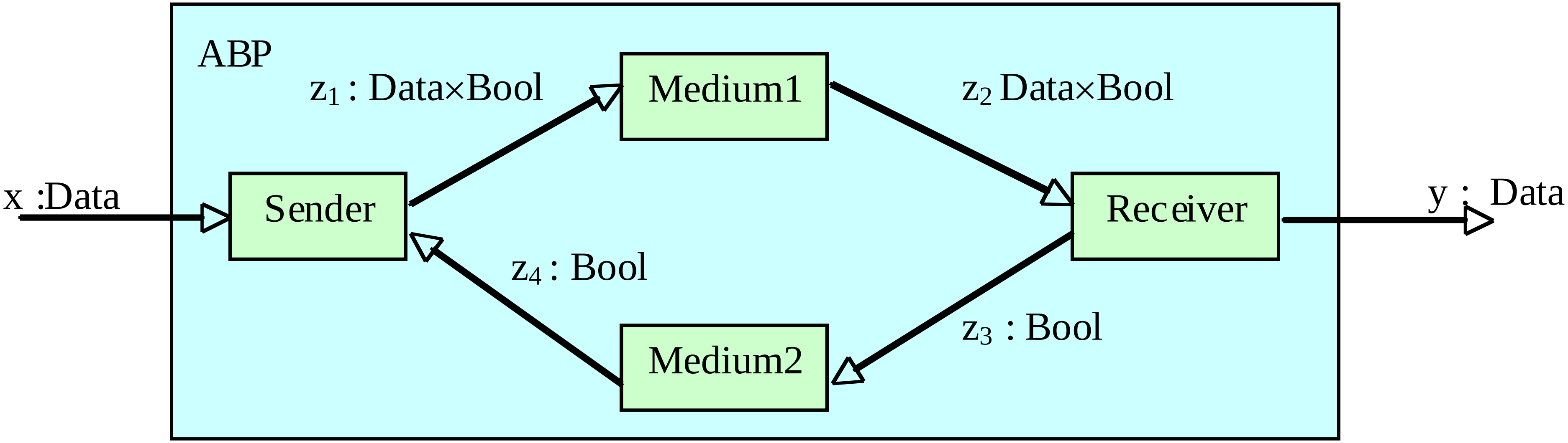
  is logically guaranteed by the simple specification

  $\forall$ 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.

Designing Architectures

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

"Alternating-Bit"-Protocol



Alternating Bit Protocol

- We specify interface behaviour of system ABP by the interface assertion

$$x \approx y$$

and furthermore its architecture by the assertion

$$\begin{aligned}
abp(x, z1, z2, z3, z4, y) \equiv \\
& abs(z1) \approx x \\
\wedge\ & abs(z2) \approx x \\
\wedge\ & Data\#y = \#abs(z2) \\
\wedge\ & Data\#x = \#abs(z3) \\
\wedge\ & \#abs(z3) = \#abs(z4) \\
\wedge\ & abs(z2) \approx y
\end{aligned}$$

Auxiliary functions

- The auxiliary function abs eliminates all repeated elements and empty slots in a stream.
- The auxiliary function abs is described by following equations (using the auxiliary function del):

$$\mathrm{abs}(\langle - \rangle \hat{\ } z) = \mathrm{abs}(z)$$

$$\mathrm{abs}(\langle e \rangle \hat{\ } z) = \langle e \rangle \hat{\ } \mathrm{del}(z, e)$$

$$\mathrm{del}(\langle e \rangle \hat{\ } z, e) = \mathrm{del}(z, e)$$

$$e \neq d \Rightarrow \mathrm{del}(\langle d \rangle \hat{\ } z, e) = \mathrm{abs}(\langle d \rangle \hat{\ } z)$$

$$\mathrm{del}(\langle - \rangle \hat{\ } z, e) = \mathrm{del}(z, e)$$

- Now we can look for specifications of the sub-systems that fulfil the requirements included in the architecture.

Deriving specifications for sub-interfaces

- We derive specifications for sub-interfaces as given by

  $\{z1, z4\}$

- From

  $\exists\ x, y, z2, z3: abp(x, z1, z2, z3, z4, y)$

  we derive

  $\exists\ x: abs(z1) \approx x \wedge Data\#x = \#abs(z4)$

  from which we can derive

  $\exists\ x: \#abs(z1) = \#abs(z4)$

  This expresses a condition for the two streams on channels z1 and z4.
- This condition does not indicate who is responsible for the liveness property included in the specification.
- The example shows a way to design architectures by designing first the specifications for the channel histories and then the specification for the components.

Conclusion

- A/C specs address the logic of the architecture rather than separated interface specifications for the components
- From A/C specs we may derive separated interface specifications for the components by simplified assertions
- This gives a methodology towards a modular decomposition in architecture design

Outlook:
Assumption/Promise for Non-functional Reqs

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

Future work

- Perspectives of the systematic application of assumption/promise patterns in system development for non-functional system properties.
- Dealing with non-functional requirements in requirements engineering.

Rich Contracts: Specifying Non-Functional System Properties

- Non-functional properties deal with aspects of systems that do not address the system's behaviour.
- The formula (for all environments E):

  $$\forall\ E: Asu(E) \Rightarrow Pro(S \otimes E)$$

  specifies a contract $Con(S)$ for system S.
- An example would be the weight of system $S \otimes E$.
- Assuming the simple rule

  $$weight(S \otimes E) = weight(S) + weight(E)$$

  being interested in the system specification (for given k)

  $$weight(S \otimes E) \leq k$$

- that requires a limit for the system's weight we get the formula (with $k' < k$)

  $$\forall\ E: weight(E) \leq k' \Rightarrow weight(S \otimes E) \leq k$$

  which is equivalent to the proposition

  $$weight(S) \leq k-k'$$

Remark on Non-functional Requirements

- It is not so obvious what it means that a property is "non-functional"
- If weight is a monitored or a controlled variable then it is part of the functional system properties
- We may distinguish between
  - qualitative properties
  - quantitative properties

Final Remarks

Rich specifications

- Probability
- Continuous time, continuous signals
- Quality by quantitative properties
  - Performance
    - Quantitative response times
    - Resource consumption
  - Reliability
  - Functional safety
  - Security
  - ...
- The goal is to treat all these properties in a modular way