Challenges for the Model-based Development of Distributed Real Time Systems

(Position Paper)

Michael Giddings, Jan Jürjens*, Pat Allen

Computing Department
The Open University, GB

[*from 1 Oct 2008 also:
Microsoft Research (Cambridge)]

J.Jurjens@open.ac.uk

Large Distributed Real Time Systems

Problems regarding development:

- Performance failures are discovered at the end of development when they are expensive to resolve.
- Poor traceability between design stages.
- Difficult to analyze / predict performance from design models.
- Common problems with design models:
  - Allow different views of the system to become disjointed.
  - Are frequently at the wrong level of abstraction.
  - May not correctly represent the virtual flow of data changes across the system in iterative systems.
- Design Specialists require a functional overview without which requirements will be inadequately specified.

Using MDA for real time systems

- In MDA Platform Independent Models are automatically translated into Platform Specific Models – enables consistency and traceability between design stages. Promotes re-use and abstraction.
- PIM can include non functional requirements.

Challenges:

- Given a PIM, how to do performance analysis that truthfully reflects actual performance of the running system ?
- Make PIM sufficiently easy to understand for user domain experts.

Performance Analysis on PIMs

- Non functional requirements are associated with end events or displays.
- PIM needs to link Functional Elements to relevant end events.
- Use Functional Element diagrams (from CoRE) to do this.
- Having identified which Functional Elements are associated with an end event, can annotate with non functional requirement tags.
- Use these to link the non functional requirements to representations in activity diagrams which show Functional Elements.
- Performance Analysis can be undertaken from these activity diagrams in association with information about delays associated with iteration and scheduling.

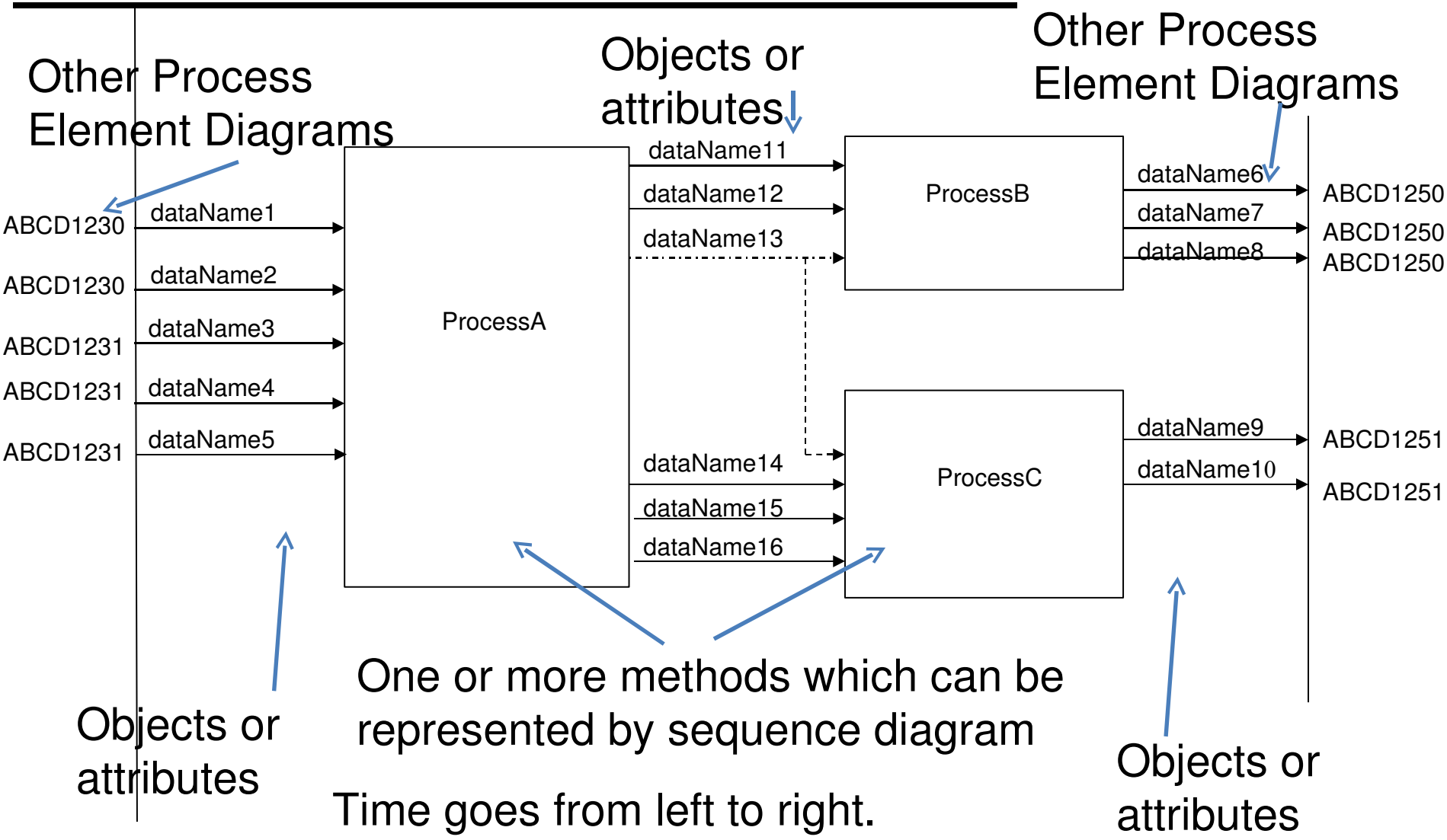
UML profile for Schedulability, Performance and Time: Refers to the domain viewpoint but does not specifically identify Functional and Process Elements. Give formal definition of Functional and Process Elements within the context of this profile.

Research Goals

- Can performance be assessed at the requirements phase using Process Elements and Schedulers?
- Can the PIM be extended to include a process oriented view of the system in order help the domain experts trap the requirements of a complex distributed systems?
- Can Process Elements and schedulers be used to model abstractions of hardware elements such as COTS equipment and busses?
- Can the implementation be split using a relationship between loosely coupled process elements and schedulers?

Use UML for the OO view. Can describe scheduler in UML although standard iterators and schedulers independent from the rest of the design will be used. Scheduler logic may be a table or an activity diagram showing Functional/Process Elements.

Functional Element Diagrams (CoRE)



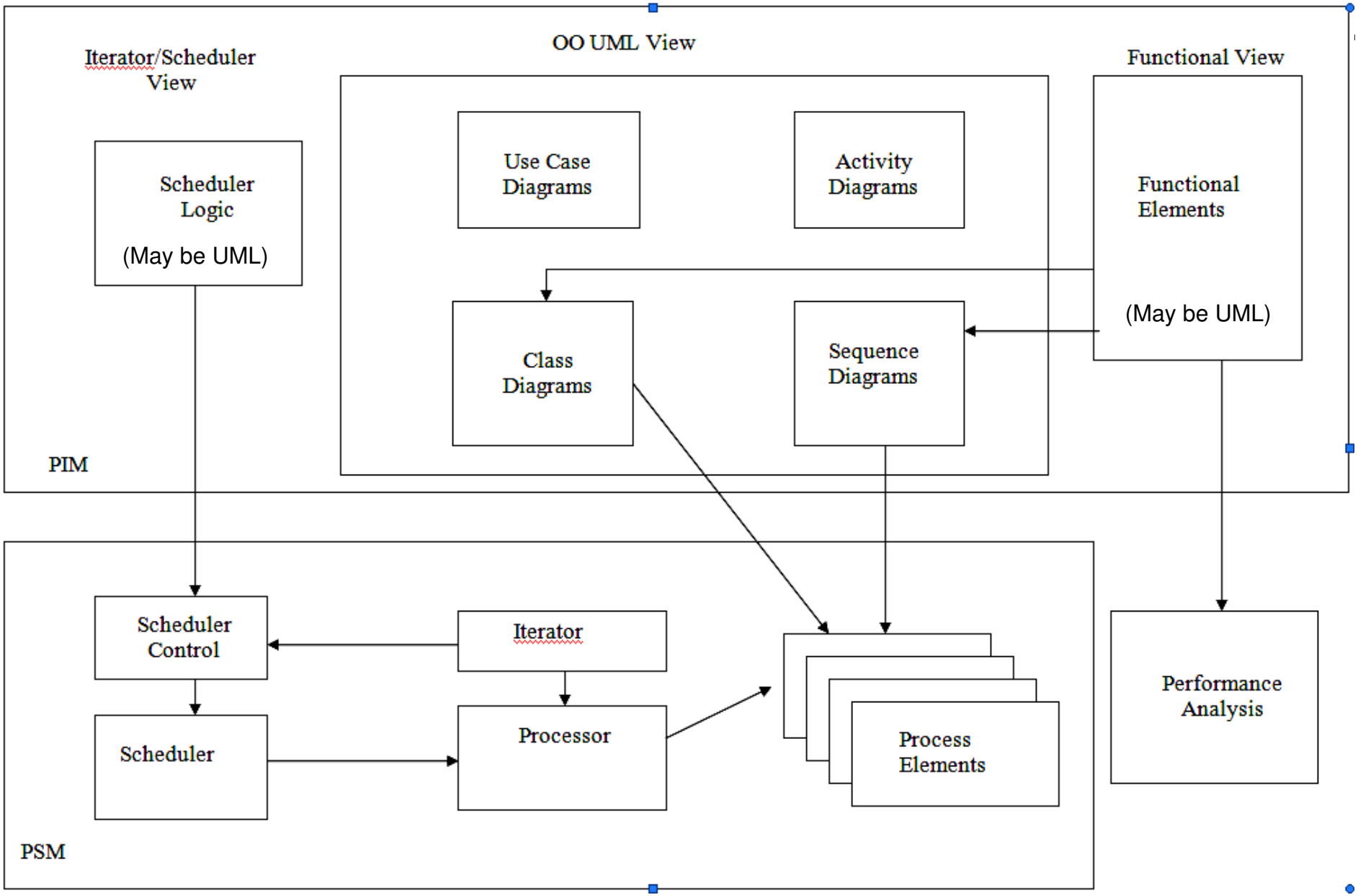
Functional Elements

- Processes Elements are units of functionality which can be represented by a sequence diagram.
- Represents single functional element that can be scheduled as a component part of an iteration.
- Data represent objects or attributes. Dashed lines represent triggers.
- Functional elements can be designed/implemented independently.
- Can compose diagrams to identify all functions that functional element depends on using the diagram numbers on the left and right edges.
- Can thus identify the functional elements that contribute to end event.
- Single or groups of Functional elements together with scheduler/ iterator models can be animated in a way that reflects real performance.
- For abstraction, a higher level Functional Diagram can be decomposed into simpler diagrams until the base line Functional Elements are shown.
- Can reuse Functional Elements where necessary. Can obtain complex functions from combinations of the Functional Elements.

Linking Functional Element Diagrams with UML

- Diagram annotated for links from its parts to the OO UML view.
- Functional elements can be used to identify classes and methods.
- Annotated with traceability trackers and other non functional requirements like safety and mission criticality.
- Performance diagrams (tagged activity diagrams) can be generated containing the processing time for individual functional elements and the delays caused by the scheduler and iteration delays.
- The PIM can thus be animated in a way that reflects real performance.

PIM to PSM Translation



PIM to PSM Translation

- Functional Elements can be translated into Process Elements automatically.
- Process Elements can be substituted for Functional Elements in PIM animation.
- Standard schedulers can be used
- Scheduler logic can be changed independently of Functional Elements and Process Elements.
- Performance analysis can be undertaken by animation or by extracting data flow from the Functional Model.

Conclusions

- The use of MDA based on a extension of UML with CoRE's Functional Element diagrams and Process Elements is expected to offer the possibility of overcoming some of the practical difficulties involved in the development of complex distributed iterating systems, in particular wrt:
  - traceability between design stages
  - performance analysis and prediction on the basis of the PIM.

Large Distributed Real Time Systems

Large Distributed Iterating Real Time systems are:

- Time consuming to develop.
- Frequently cost more than estimated
- System development frequently overruns.

Real Time Architecture

Large Real Time Systems comprise of a mixture of:-

- Iterating Systems
- State Driven Systems
- Free Running Systems

This position paper/presentation concentrates on

- Large Distributed Real Time Systems containing software that iterates.

Fault Semantics Modeling

For redundancy model *R*, stereotype *s* {ℜℜcrash/performance⊤⊤,ℜℜvalue⊤⊤}, have subset Faults$_R$(*s*) of {delay(*t*), loss(*p*), corrupt(*q*)}, with interpretation:

- *t*: expected maximum time delay,
- *p*: probability that value not delivered within *t*,
- *q*: probability that value delivered in time corrupted

(in each case incorporating redundancy).
Or use ℜℜrisk⊤⊤ stereotype with {fault} tag.

Example

Suppose redundancy model *R* uses controller with redundancy *3* and fastest result. Then can define:

- delay(*t*): *t* delay of fastest controller,
- loss(*p*): *p* probability that fastest result not delivered within *t*,
- corrupt(*q*): *q* probability that fastest result is corrupted

(each wrt. given fault semantics).

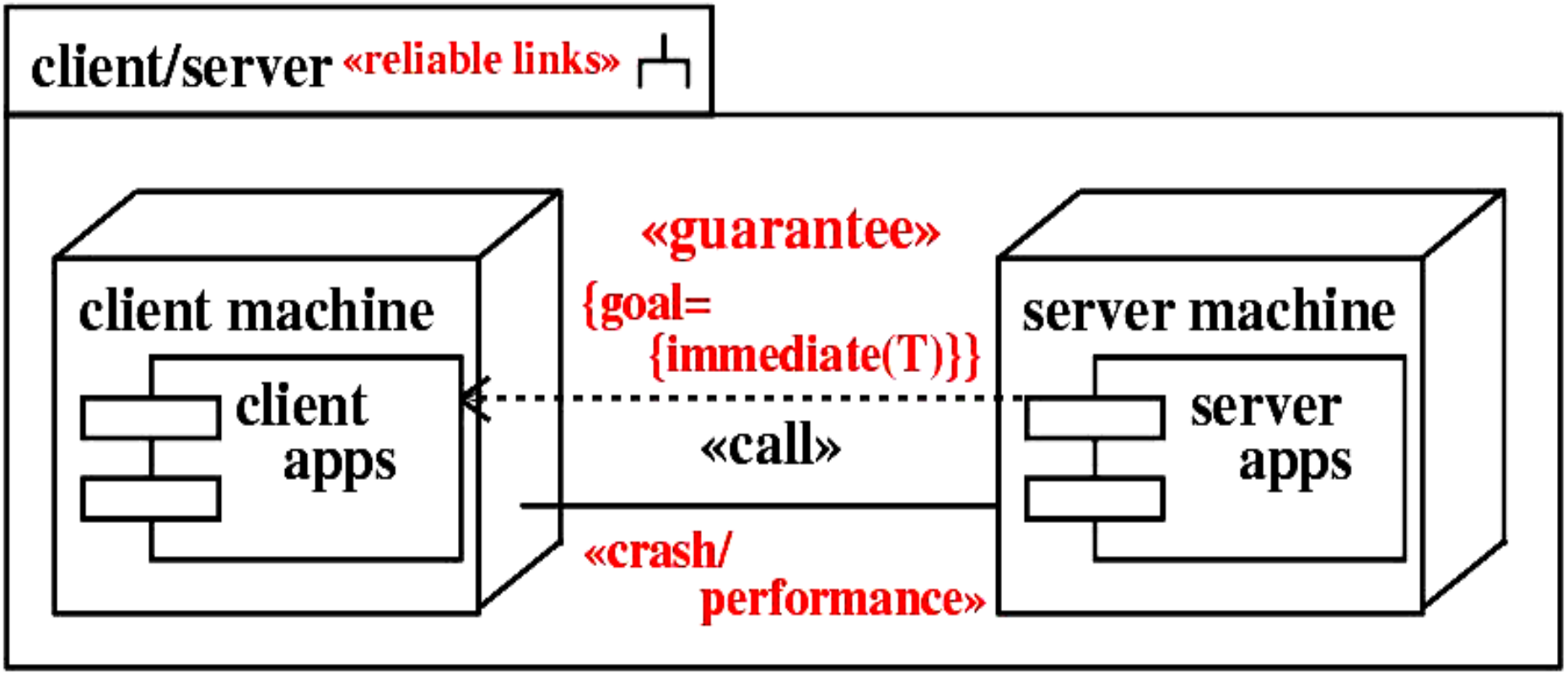
ℜℜguarantee⊤⊤

Describe guarantees required from communication dependencies resp. system components.

Tags: {goal} with value subset of {immediate(*t*), eventual(*p*), correct(*q*)}, where

- *t*: expected maximum time delay,
- *p*: probability that value is delivered within *t*,
- *q*: probability that value delivered in time not corrupted.

Reliable Architecture



Is this a reliable architecture ?

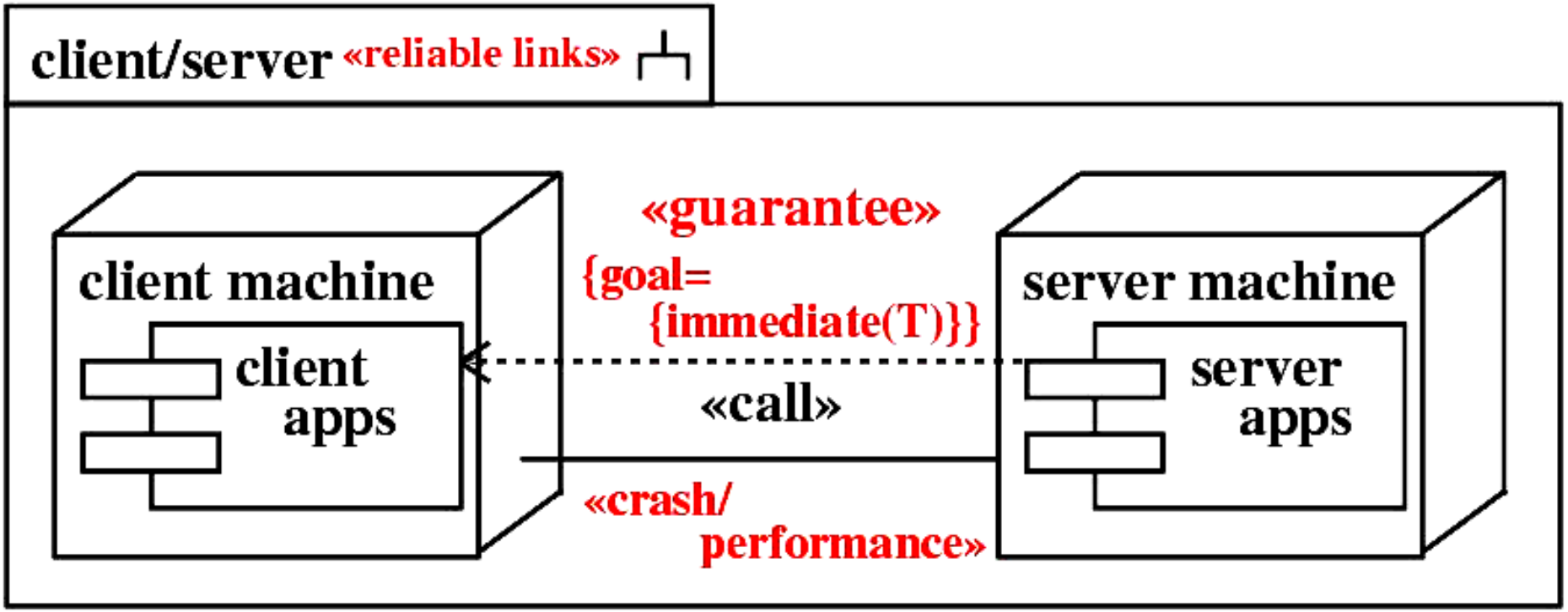
ℜℜreliable links⊤⊤

Physical layer should meet reliability requirements on communication given redundancy model *R*.

Constraint: For dependency *d* stereotyped ℜℜguarantee⊤⊤ and each corresponding communication link *l* with stereotype *s*:

- if {goal} has immediate(*t*) as value then delay(*t'*) ⟩\Faults$_R$(*s*) implies *t'*≤*t*,
- if {goal} has eventual(*p*) as value then loss(*p'*) ⟩\Faults$_R$(*s*) implies *p'*≤1-*p*, and
- if {goal} has correct(*q*) as value then corruption(*q'*) ∈Faults$_R$(*s*) implies *q'*≤1-*q*.

Example ℜℜreliable links⊤⊤



Given redundancy model none, ℜℜreliable links⊤⊤ fulfilled iff $T \geq t$ where delay(t) ⟩\ $Faults_{none}$(ℜℜcrash/performance⊤⊤).

Tool Support: Fault Models

lq^l_n: messages on link l delayed further n time units.

p^h_n: probability of fault at n^{th} iteration in history h.

For link l stereotyped s where $loss(p) \in Faults_R(s)$,

- history may give $lq^l_0 := \varnothing$; then append p to $(p^h_n)_{n \in \mathcal{N}}$,
- or no change, then append $1\text{-}p$.

For link l stereotyped s where $corruption(q) \in Faults_R(s)$,

- history may give $lq^l_0 := \{\blacksquare\}$; then append q,
- or no change; append $1\text{-}q$.

For link l stereotyped s with $delay(t) \in Faults_R(s)$, and $lq^l_0 \neq \varnothing$, history may give $lq^l_n := lq^l_0$ for $n \leq t$; append $1/t$.

Then for each n, $lq^l_n := lq^l_{n+1}$.