Behavioural Contracts for Components

Cyril Carrez

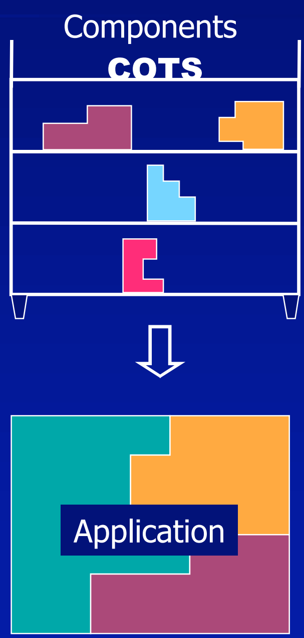
01/03/2004

NTNU

Design by Assembly

- ADL (90's)
  - components
  - connectors
  - configuration
- UML 2.0 (2003)

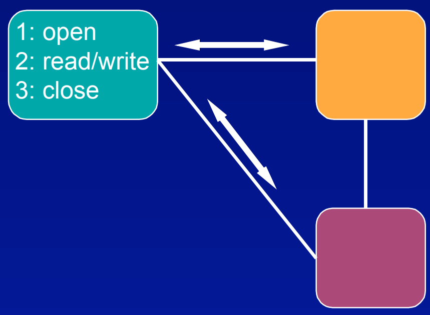
Classification
[Medvidovic & Taylor]



- Behavioural typing with explicit types
  - Regular types [Nierstrasz]
  - «non understood message» [Najm et al.]
- Contracts
  - Design by Contract [Meyer]
  - Classification [Beugnard et al.]
    - Syntactic / **behaviour** (pre/post) / **synchronisation** / QoS

Framework of the study

- Components
  - specification + code
- *Non uniform services*
- Dynamic links



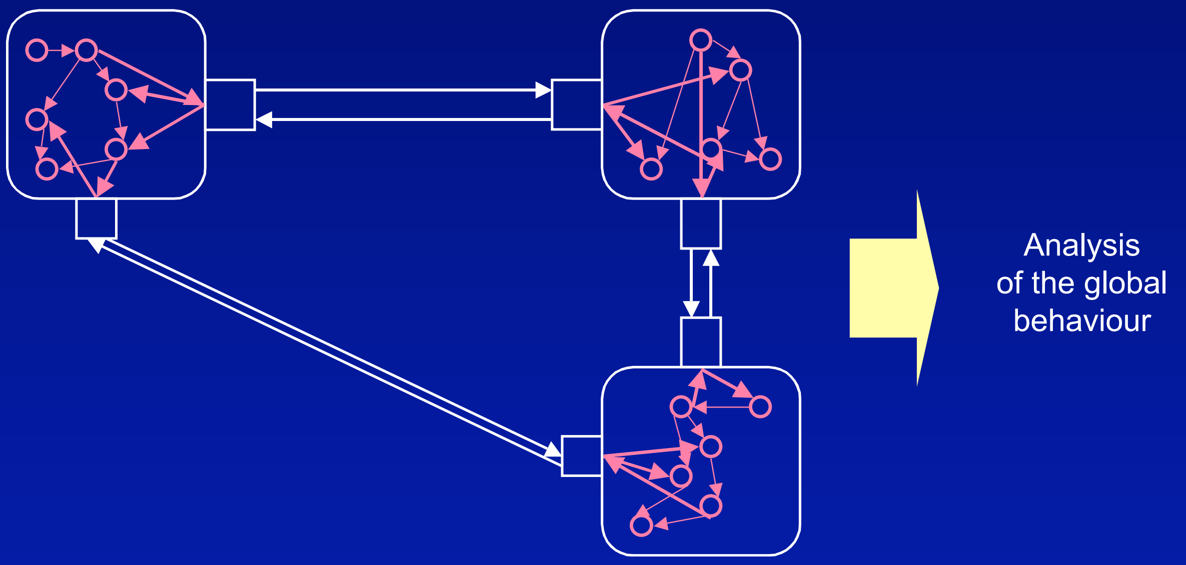
Objectives

- Safety properties: no external deadlock
- Liveness properties: messages will be consumed

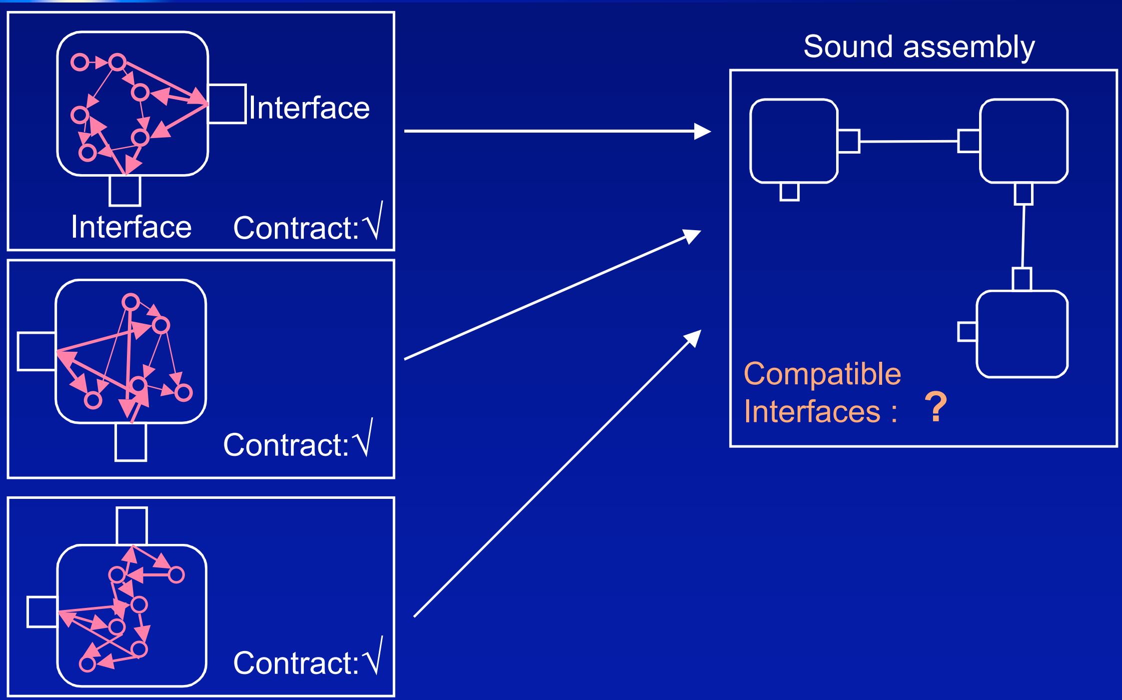
Roadmap

- The approach
- Interface language
- Component semantics
- Contract respect
- Sound assembly
- Conclusion & Perspectives

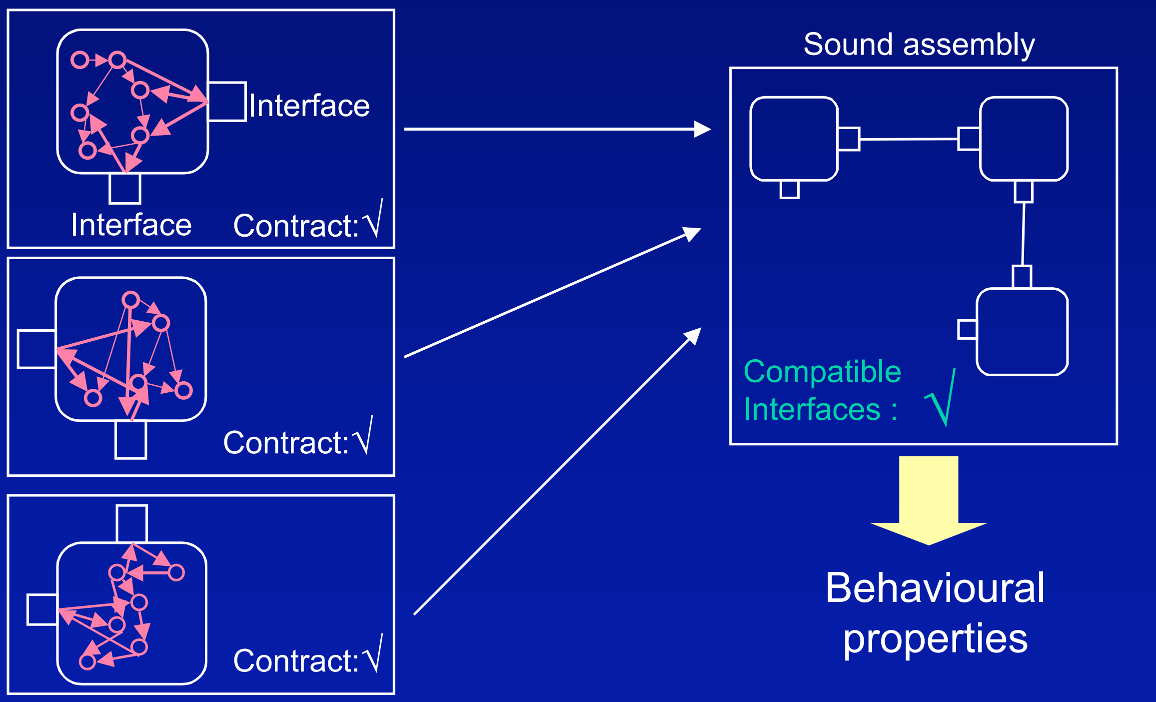
Approaches: Darwin, Wright,...



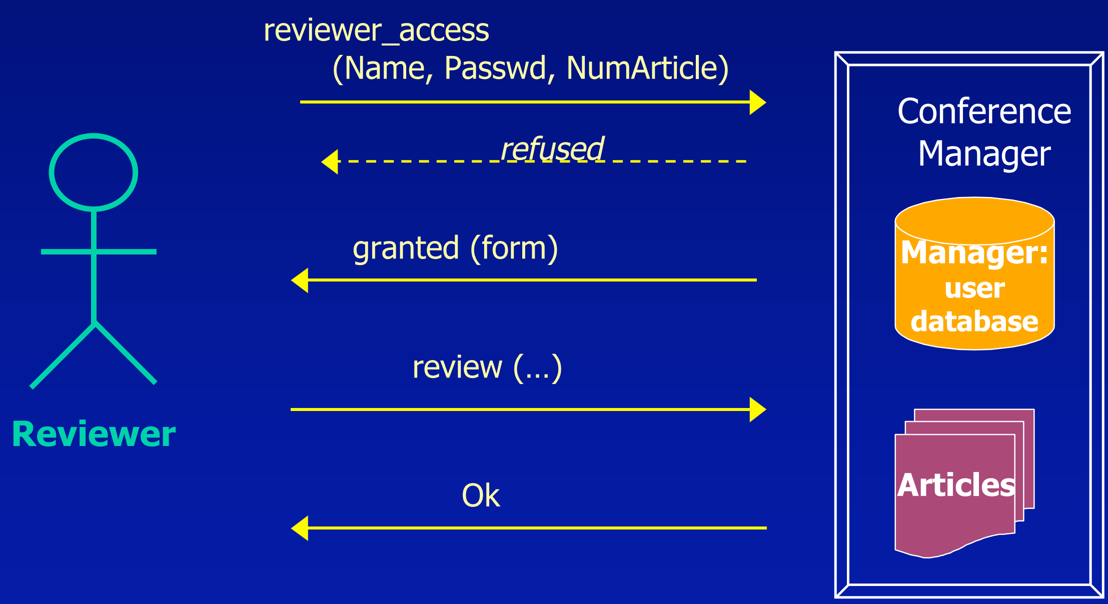
Our approach



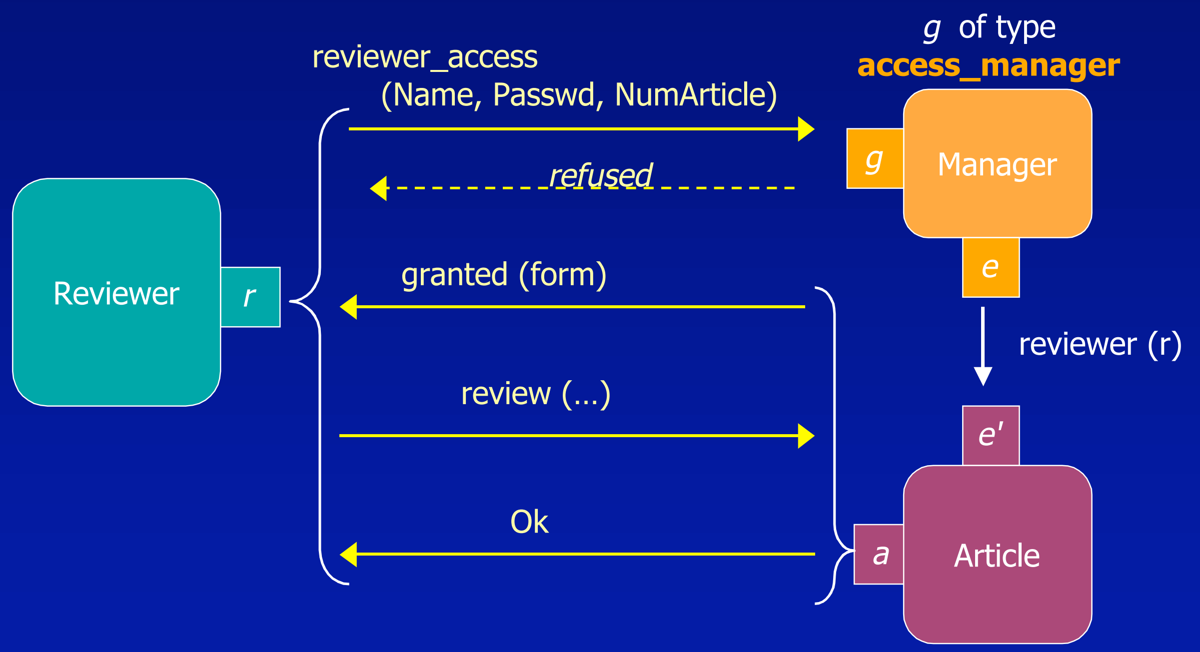
Our approach



Interface types: example



Interface types: example



Example: Type access_manager

- **access_manager** =
  may ? [reviewer_access (string,string,integer)**;**
  must ! [refused**; 0**
  + granted (strings)**; reviewer_manager]]**

- **reviewer_manager** =
  must ? [review (strings)**; must ! [** Ok**; reviewer_manager_chg**
  + error**; reviewer_manager]]**
- **reviewer_manager_chg** =
  may ? [review (strings)**; must ! [** Ok**; reviewer_manager_chg**
  + error**; reviewer_manager_chg]]**

Example: Type access_manager

allowed: you can send, I guarantee the reception

- **access_manager** =
  may ? [reviewer_access (string,string,integer);
  must ! [refused; **0**
  \+ granted (strings); **reviewer_manager**]]

obligation: I must send

You must send

- **reviewer_manager** =
  must ? [review (strings); **must !** [Ok; **reviewer_manager_chg**
  \+ error; **reviewer_manager**]]
- **reviewer_manager_chg** =
  may ? [review (strings); **must !** [Ok; **reviewer_manager_chg**
  \+ error; **reviewer_manager_chg**]]

Compatibility: *Comp (I, J)*

J \ I	must ?	may ?	must !	may !	0
must ?			√		
may ?		√	√	√	√
must !	√	√			
may !		√			
0		√			√

$$Comp(\ mod_I\ !\ [\ \Sigma_k\ M_k\ ;\ I_k\ ],\ mod_J\ ?\ [\ \Sigma_l\ \ M_l\ ;\ J_l\ ]\ ) =_{\text{def}}$$

$$Comp_{\text{mod}}\ (\ mod_I\ !,\ mod_J\ ?\ )$$

$$\wedge\ (\ \forall k,\ \exists l : Comp_{\text{msg}}(\ M_k,\ M_l\ ) \wedge Comp(\ I_k,\ J_l\ )\ )$$

$$Comp_{\text{msg}}(\ M_!\ (I_i\ ),\ M_?(J_i\ )\ )\ =_{\text{def}}\ M_! = M_?\ \wedge\ \ \forall i,\ I_i \leq J_i$$

Compatibility: *Comp (I, J)*

	must ?	may ?	must !	may !	0
must ?			√		
may ?		√	√	√	√
must !	√	√			
may !		√			
0		√			√

- reviewer_manager =
  must ? [review (strings); **must ! [** Ok; reviewer_manager_chg
  + error; reviewer_manager **]]**
  reviewer_manager_chg = **may ? […]**
- enter_review =
  must ! [review (strings); **must ? [** Ok; **0**
  + error; enter_review **]]**

Compatibility: *Comp (I, J)*

	must ?	may ?	must !	may !	0
must ?			√		
may ?		√	√	√	√
must !	√	√			
may !		√			
0		√			√

- reviewer_manager =
  must ? [review (strings); **must ! [** Ok; reviewer_manager_chg
  + error; reviewer_manager **]]**
  reviewer_manager_chg = **may ? […]**
- enter_review =
  must ! [review (strings); **must ? [** Ok; **0**
  + error; enter_review **]]**

Compatibility: *Comp (I, J)*

	must ?	may ?	must !	may !	0
must ?			√		
may ?		√	√	√	√
must !	√	√			
may !		√			
0		√			√

- reviewer_manager =
  must ? [review (strings)**; must ! [** Ok; reviewer_manager_chg
  + error; reviewer_manager **]]**

  reviewer_manager_chg = **may ? […]**
- enter_review =
  must ! [review (strings)**; must ? [** Ok; **0**
  + error; enter_review **]]**

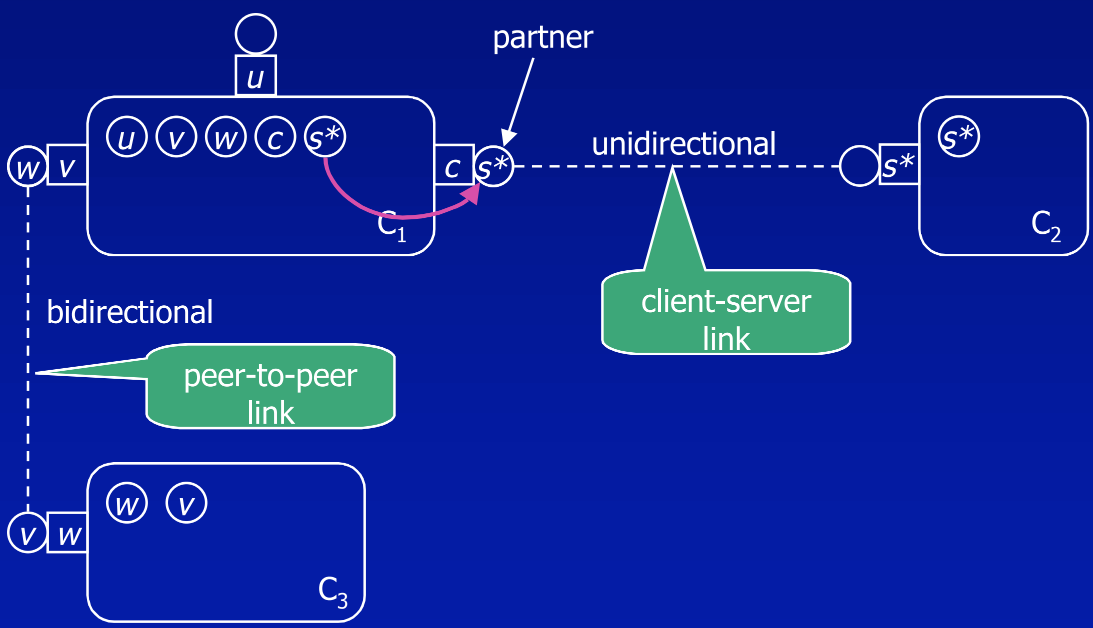
Subtyping: *T ≤ S*

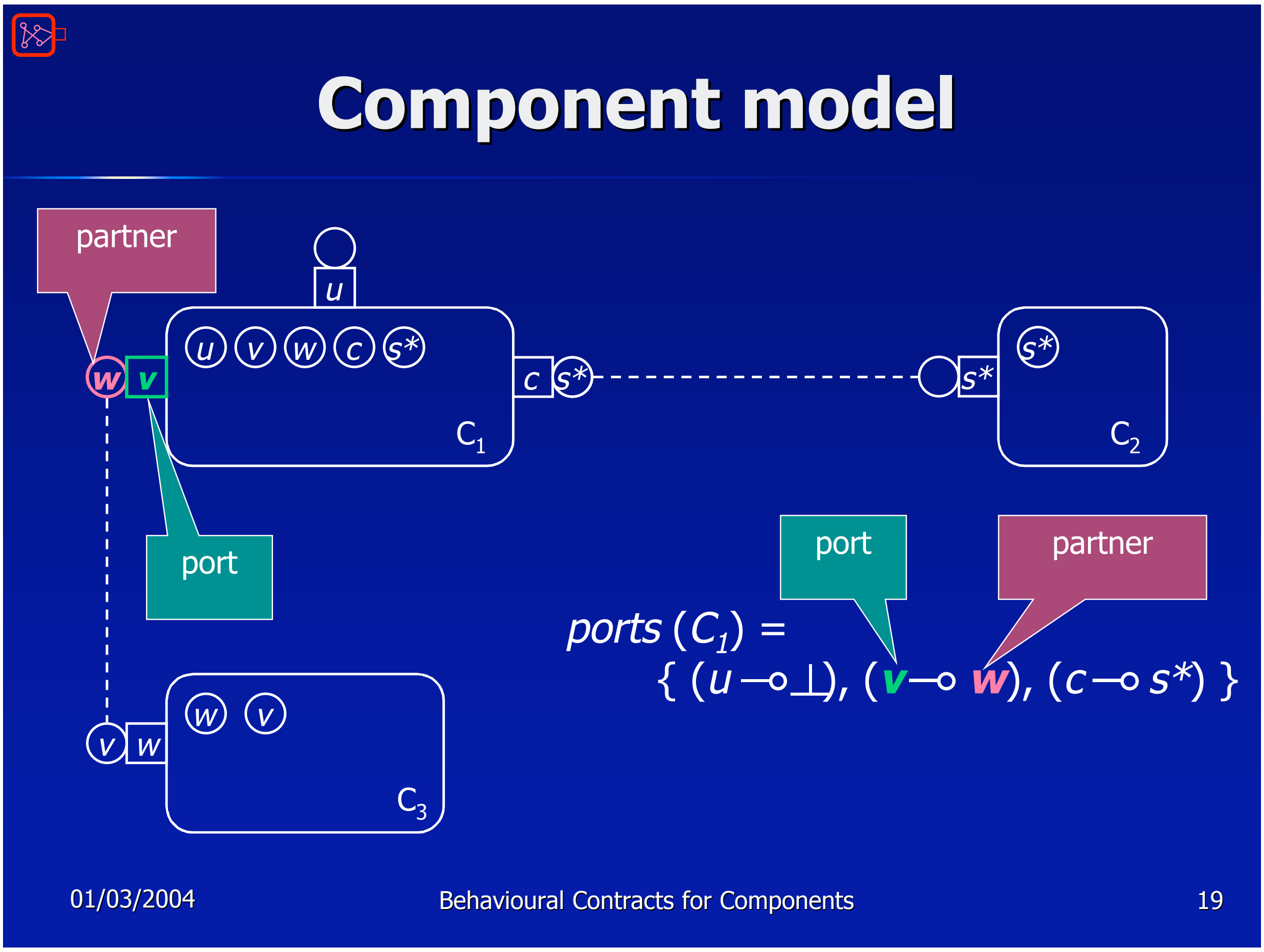
- Compatibility: sent message ≤ received message
- receivings:
  - *mod* **?** $M_1+M_2+M_3 \leq$ *mod* **?** M_1+M_2
  - contra-variant: $M(I) \leq M(J) \Leftrightarrow J \leq I$
- sendings:
  - *mod* **!** $M_1 \leq$ *mod* **!** M_1+M_2
  - co-variant: $M(I) \leq M(J) \Leftrightarrow I \leq J$
- modalities:
  - **may ? ≤ must ?**
  - **may ? ≤ 0**
  - **may ? ≤ may !**
  - **must ! ≤ may !**
  - **0 ≤ may !**

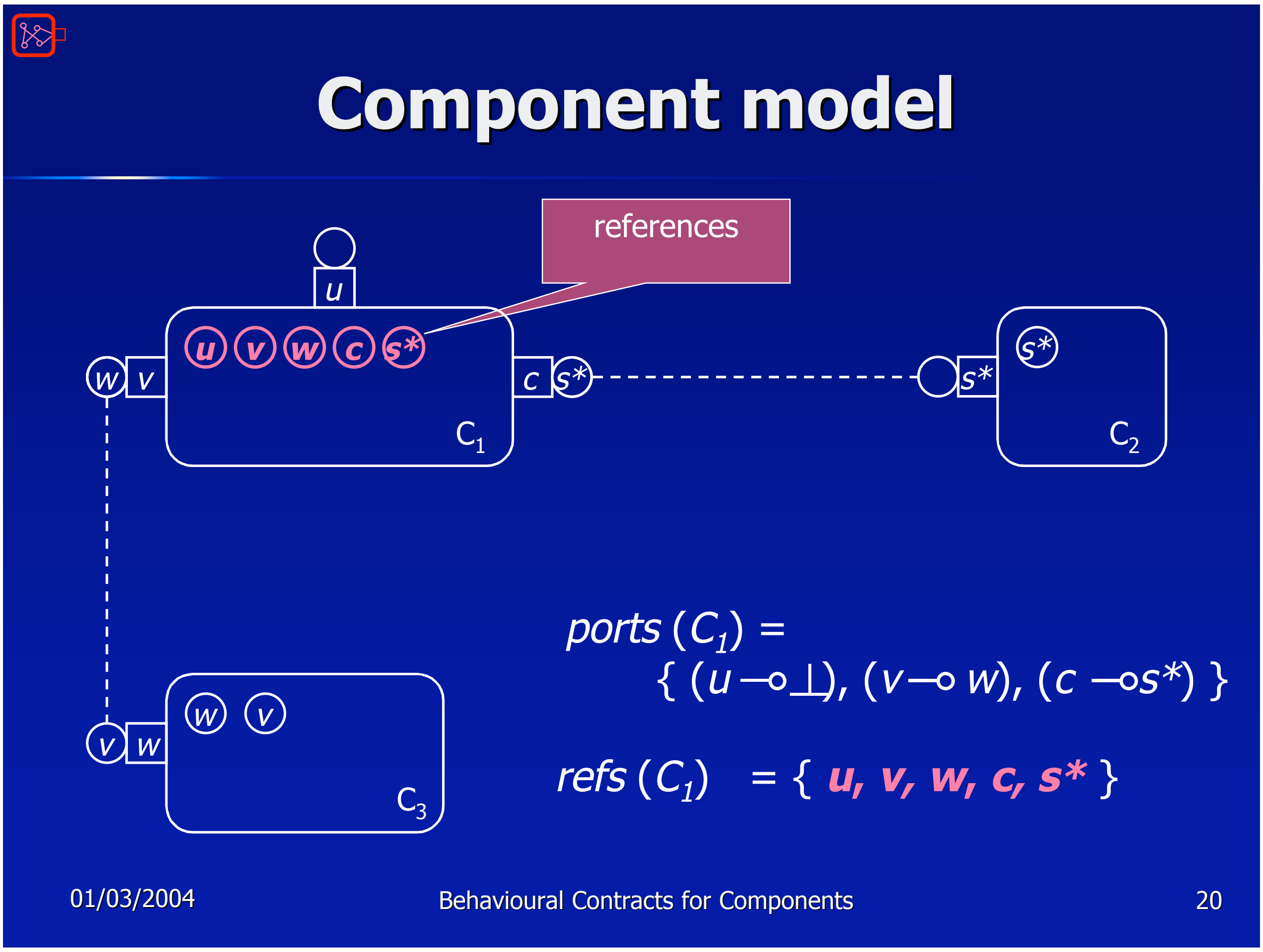
Properties of the subtypes

- $\leq$ is transitive
- Subtype can replace super-type
  - $Comp\,(I, S)\ \&\ (T \leq S) \Rightarrow Comp\,(I; T)$
- Greater compatible super-type:
  - dual: $J^D =_{def} J$ with reversed sendings and receivings
  - $Comp\,(I, J) \Leftrightarrow I \leq J^D$
- Demonstrations
  - by induction on the structure of the types

Component model




Component model
partner
port
u
u v w c s*
w v
c s*
s*
s*
C_1
C_2
port
partner
ports (C_1) =
{ (u ⊸ ⊥), (v ⊸ w), (c ⊸ s*) }
w v
v w
C_3
01/03/2004
Behavioural Contracts for Components
19


Component model
references
u
u v w c s*
w v
c s*
s*
s*
C_1
C_2
ports (C_1) =
{ (u ⊸ ⊥), (v ⊸ w), (c ⊸ s*) }
refs (C_1) = { u, v, w, c, s* }
w v
v w
C_3
01/03/2004
Behavioural Contracts for Components
20

Component model: ports

- Model based on observation of ports
- State of a port : $u\rho^{\sigma}$

  – ρ = action = $\begin{cases} \textbf{!} & u \text{ is in a sending state} \\ \textbf{?} & u \text{ is in a receiving state} \\ \textbf{0} & u \text{ has no action} \end{cases}$

  – σ = activity = $\begin{cases} \textbf{a} & u \text{ is active} \\ \textbf{s} & u \text{ is suspended} \\ \textbf{i} & u \text{ is idle} \end{cases}$

- Example:
  – $u\,?^{a}$ = active in receiving $u\,!^{s}$ = suspended in sending

Component model: threads

- Multi-threaded components
- Dependencies between ports: $x \rightarrowtail y$
  – activity of *x* is suspended until *y* terminates or becomes idle
- A thread is a chain *(head, queue)*
  – *head:* current active port,
  – *queue:* ordered sequence of suspended ports
  – can dynamically grow/diminish

$$\underbrace{u_1!^{s} \rightarrowtail u_2!^{s} \rightarrowtail \ldots\ldots u_{n-1}!^{s}}_{\text{queue}} \rightarrowtail \underbrace{u_n?^{a}}_{\text{head}}$$

Component model: threads

- Multi-threaded components
- Dependencies between ports: $x \rightarrowtail y$
  - activity of *x* is suspended until *y* terminates or becomes idle
- A thread is a chain *(head, queue)*
  - *head:* current active port,
  - *queue:* ordered sequence of suspended ports
  - can dynamically grow/diminish

$$u_1!^{s} \rightarrowtail u_2!^{s} \rightarrowtail \dots\dots\ u_{n-1}!^{a}$$

queue head

Component semantic

- Component: B (P, R, T)

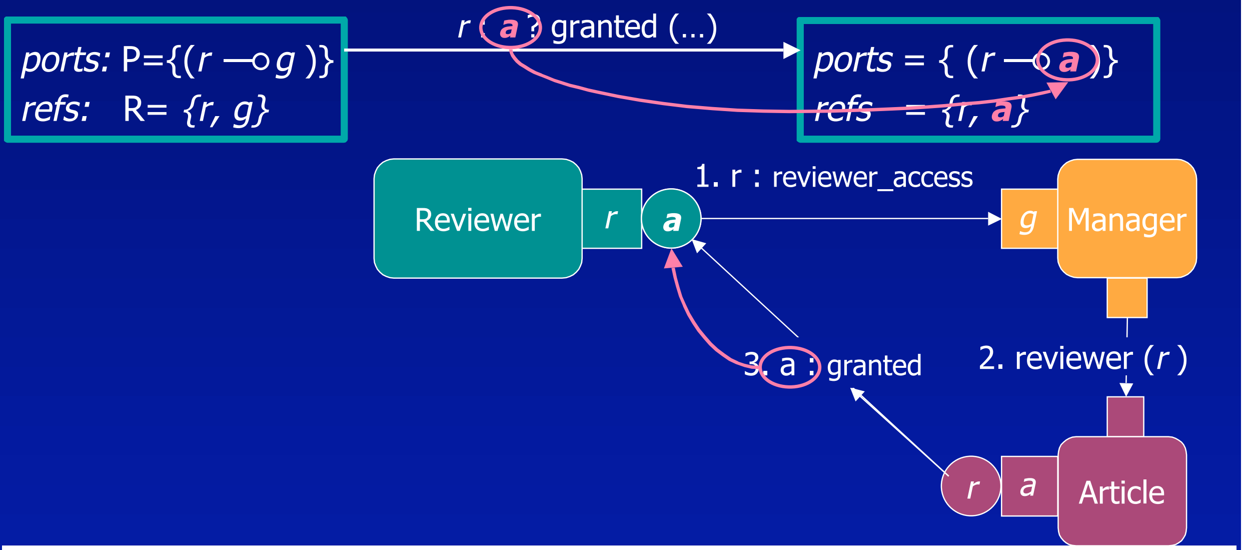
state ports, references, threads

- Operational semantic
  - *B* (P, R, T), *Com* $\longrightarrow$ *B'* (P', R', T'), *Com'*

async. com.: Fifo queues

- 11 Rules:
  - creation / removal of ports
  - binding
  - (de)activation of ports (idle, active, suspended)
  - sending/receiving messages

Example: RECV for Reviewer component



$$\frac{T' = T[u\rho/u?] \qquad R' = R \cup \{\text{refs}(\tilde{v}), u''\} - \{u' \mid (u \multimap u') \wedge \text{peer}(u')\} \qquad Com' = Com[u\triangleright] \qquad P' = P[u \multimap u''] \text{ si } \text{peer}(u)}{B(P, R, T), Com \xrightarrow{u:u''?M(\tilde{v})} B'(P', R', T'), Com'} \triangledown$$

Some other rules

$$\text{C-BIND} \quad \frac{P' = P[u \multimap v]}{B(P, R, T), Com \xrightarrow{\text{bind}(u \multimap v)} B'(P', R, T), Com} \square$$

$$\square \triangleq (u \multimap \bot) \wedge T(u) = !^{a,i} \wedge v \in R \wedge (\text{peer}(v) \Rightarrow v \notin CoDom(P))$$

-only sending ports, not suspended
-peer reference is attached to 1 port

$$\text{C-ACTV} \quad \frac{T' = T[u \rightarrowtail v]}{B(P, R, T), Com \xrightarrow{\text{actv}(u \rightarrowtail v)} B'(P, R, T'), Com} \quad T(u) = !^{a} \wedge T(v) = !^{i}$$

-a port cannot suspend on a receiving port

$$\text{C-SEND} \quad \frac{R' = R - \text{peer}(\tilde{v} \cup \{u\}) \qquad T' = T[u\rho/u!] \qquad Com' = Com[u' \triangleleft u : M(\tilde{v})]}{B(P, R, T), Com \xrightarrow{u:u'!M(\tilde{v})} B'(P, R', T'), Com'} \triangle$$

-peer reference is private: known only to the partner

Component and contracts

- Contractual component: B(…),$\tilde{C}$
  - correct behaviour

$$\frac{\tilde{C} \xrightarrow{\alpha} \tilde{C}' \qquad B(\ldots) \xrightarrow{a} B'(\ldots) \qquad a:\alpha}{B(\ldots),\ \tilde{C} \xrightarrow{a:\alpha} B(\ldots),\ \tilde{C}'}$$

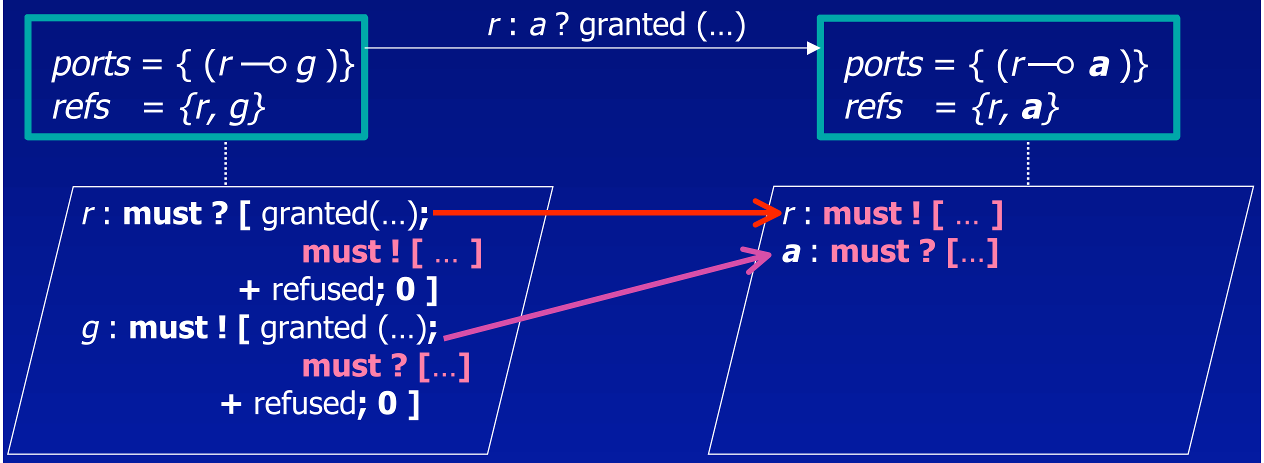
  - unauthorized transition

$$\frac{\tilde{C} \not\xrightarrow{\alpha} \tilde{C}' \qquad B(\ldots) \xrightarrow{a} B'(\ldots) \qquad a:\alpha}{B(\ldots),\ \tilde{C} \xrightarrow{a:\alpha} Error}$$

  - missing required transition

$$\frac{\tilde{C} \xrightarrow{\alpha} \tilde{C}' \qquad B(\ldots) \not\xrightarrow{a} B'(\ldots) \qquad a:\alpha}{B(\ldots),\ \tilde{C} \xrightarrow{a:\alpha} Error} \quad \mathrm{mod}(\alpha) = \textbf{must}$$

Example: RECV for Reviewer component



$$\frac{u:T \equiv mod\ \textbf{?}\ M_\Sigma \qquad u':T' \equiv mod'\ \textbf{!}\ M'_\Sigma \qquad B(\mathsf{P},\mathsf{R},\mathsf{T}) \xrightarrow{u:u''\textbf{?}m_k} B'(\mathsf{P}',\mathsf{R}',\mathsf{T}')}{(B(\mathsf{P},\mathsf{R},\mathsf{T}),\tilde{C}) \xrightarrow{u:u''\textbf{?}m_k} (B'(\mathsf{P}',\mathsf{R}',\mathsf{T}'),\ \tilde{C}[u:T_k/T,\ u'':T'_k/u':T'] \Leftarrow \tilde{v}':\tilde{U}'_k)} \ \blacktriangle$$

Some other rules

$$\text{BIND}\ \frac{u:T \qquad v:S \qquad B(\mathsf{P},\mathsf{R},\mathsf{T}) \xrightarrow{\text{bind}(u\multimap v)} B'(\mathsf{P}',\mathsf{R},\mathsf{T})}{(B(\mathsf{P},\mathsf{R},\mathsf{T}),\tilde{C}) \xrightarrow{\text{bind}(u\multimap v)} (B'(\mathsf{P}',\mathsf{R},\mathsf{T}),\tilde{C})}\ Comp(T,S)$$

$$\text{BIND-ERR}\ \frac{u:T \qquad v:S \qquad B(\mathsf{P},\mathsf{R},\mathsf{T}) \xrightarrow{\text{bind}(u\multimap v)} B'(\mathsf{P}',\mathsf{R},\mathsf{T})}{(B(\mathsf{P},\mathsf{R},\mathsf{T}),\tilde{C}) \rightarrow Error}\ \neg Comp(T,S)$$

$$\text{RECV-ERR}\ \frac{u:T \equiv mod\ ?[*]M_\Sigma \qquad \forall k, B(\mathsf{P},\mathsf{R},\mathsf{T}) \stackrel{u:u'?m_k}{\not\longrightarrow} B'(\mathsf{P}',\mathsf{R}',\mathsf{T}')}{(B(\mathsf{P},\mathsf{R},\mathsf{T}),\tilde{C}) \rightarrow Error}$$

$$\text{RECV-UN}\ \frac{u:T \equiv mod\ ?\ M_\Sigma \qquad B(\mathsf{P},\mathsf{R},\mathsf{T}) \xrightarrow{u:u'?m_k} B'(\mathsf{P}',\mathsf{R}',\mathsf{T}')}{(B(\mathsf{P},\mathsf{R},\mathsf{T}),\tilde{C}) \xrightarrow{u:u'?m_k} (B'(\mathsf{P}',\mathsf{R}',\mathsf{T}'),\tilde{C}[u:T_k/T] \Leftarrow u':T_k{}^{\mathcal{D}}, \tilde{v}:\tilde{U}_k)}\ \blacktriangle \wedge (u \multimap \bot)$$

-RECV from unknown partner: take the greater type

Sound assembly of components

- Component honouring a contract
  - *B* is well-typed: *B(P,R,T)*,$\tilde{C}$ never leads to *Error*
- Assembly of components:

$$\mathcal{A} = \{\ (B_1(\mathsf{P}_1,\mathsf{R}_1,\mathsf{T}_1),\tilde{C}_1), \ldots, (B_n(\mathsf{P}_n,\mathsf{R}_n,\mathsf{T}_n),\tilde{C}_n), Com\ \}$$

  - reference closed
  - only client/server and peer-to-peer bindings
  - all ports are active and independent
- Sound assembly:
  - all components respect their contract
  - ports bound to each other are compatible

Properties

Soundness is maintained through evolution

– a sound configuration of components never leads to *Error*

$$\forall\, C : \mathcal{A} \longrightarrow^{*} C, \quad C \not\longrightarrow Error$$

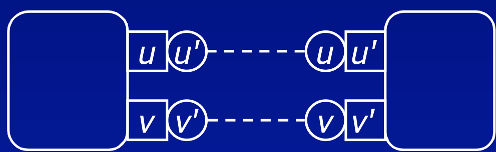
All the messages are eventually consumed

$$\forall\, u,v,i,M : (u \multimap v) \in \mathrm{P}_i ,\ C \xrightarrow{u:v\,!\,M} C'$$

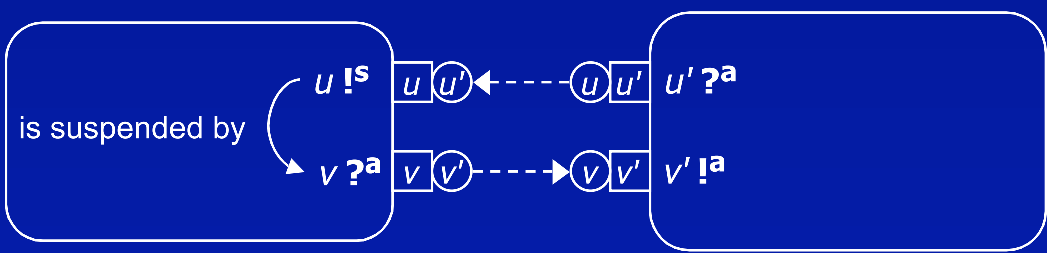
$$\Rightarrow \exists\, C'',C''' \text{ such that } C' \longrightarrow^{*} C'' \xrightarrow{v:u\,?\,M} C'''$$

External deadlock

- During assembly : no verification of the global behaviour
  - *u* and *u'* types are compatible
  - *v* and *v'* types are compatible

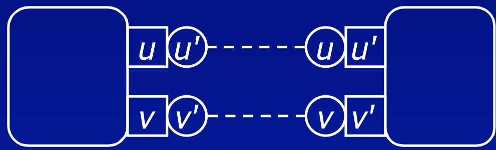


- During execution :

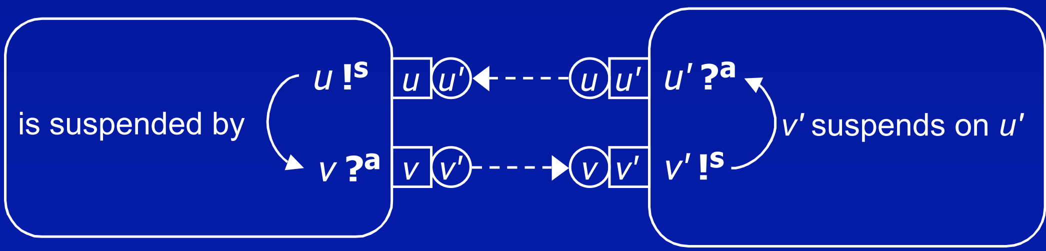


External deadlock

- During assembly : no verification of the global behaviour
  - *u* and *u′* types are compatible
  - *v* and *v′* types are compatible



- During execution :



Property: external deadlock freeness

- A port cannot suspend on a receiving port

  – external deadlock:

  – $u\ S\ v =_{def} u \rightarrowtail v \ \vee\ u \dashrightarrow v$ ($\dashrightarrow$ external dependency)

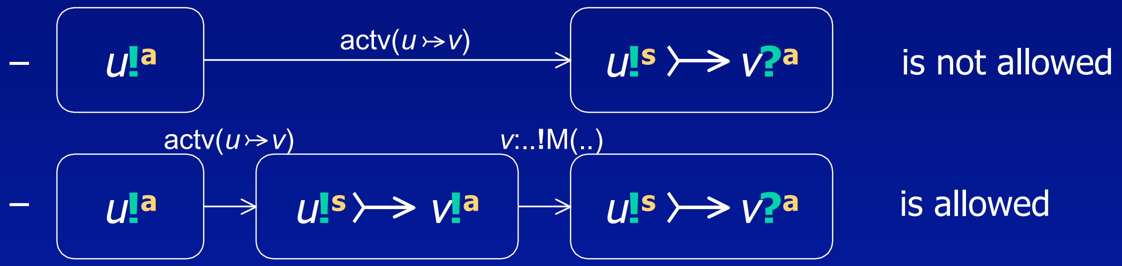
  – Ext_deadlock (C) $=_{def}$

  $\exists\ (u_i)_{1..n} \in C$ such that $\forall\ k < n :\ \ u_i\ S\ u_{i+1}\ \wedge\ u_n\ S\ u_1$

- Demonstration (deadlock freeness):
  - by induction & Reductio ad absurdum

Constraints on the component

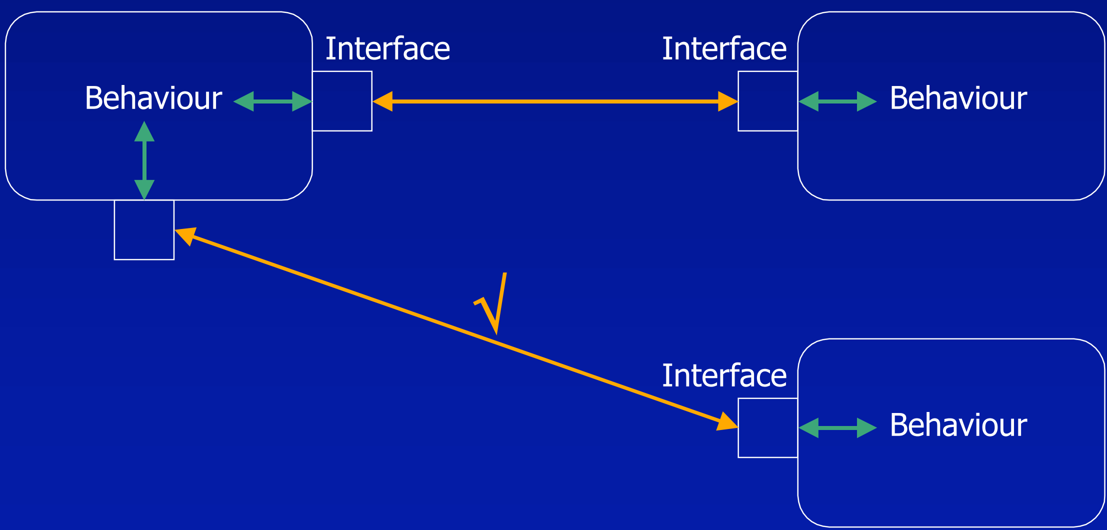
- a port cannot suspend on a receiving port:



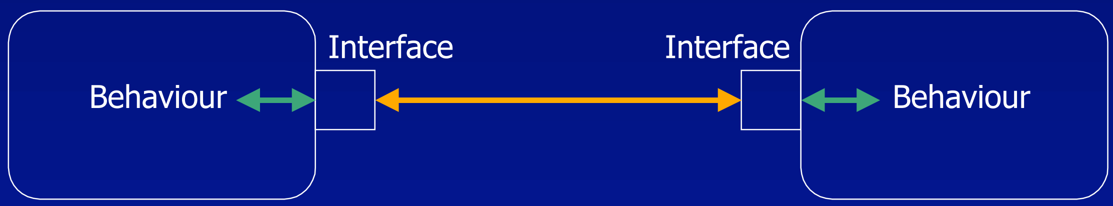
- a receiving port cannot be suspended: $u?^{s}$ forbidden
- bindings: only sending & (active or idle) ports: $u!^{a,i}$
- a '**must !**' is not suspended by a '**may ?**'
- unbind is not allowed
- [nonrentrant servers]

Application

- Sound extension of running application



Conclusion

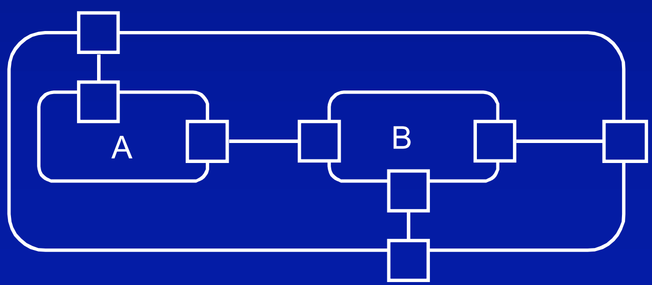


- Contract conformance: ⟷ verification during compilation
- Compatible interfaces: ⟷ verification during deployment

- Properties of a sound assembly
  - safety: a configuration never leads to *Error*
  - safety: external deadlock freeness
  - liveness: all sent message are eventually consumed

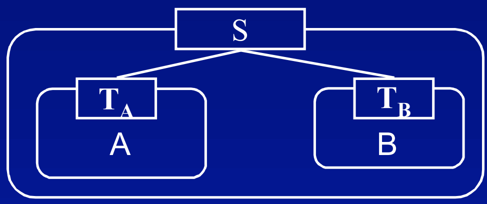
Perspectives

- Interfaces: infinite state machines
- Integration to existing component platforms
- UML Profile
- Composite components & delegation:



Future Work

- Application to UML2.0: multiple delegation



- Application to a language
- From interface contracts to component contracts
- Extension to timed interfaces

- Application to PATS!!

Get the slides!
www.cyril-carrez.net
www.item.ntnu.no/~carrez