

Deriving Component Designs from Global Requirements

Gregor v. Bochmann

**School of Information Technology and Engineering (SITE)
University of Ottawa
Canada**



uOttawa

L'Université canadienne
Canada's university

<http://www.site.uottawa.ca/~bochmann/talks/Deriving.ppt>

Presentation at the
1st Intern. Workshop on
Model Based Architecting and Construction
of Embedded Systems
Toulouse, Sept. 2008

Abstract

- This paper is concerned with the early development phases of distributed applications, service compositions and workflow systems. It deals with the transformation of a global requirements model, which makes abstraction from the physical distribution of the different system functions, into a system design that identifies a certain number of distributed components. At this design level, some of the global activities may be seen as collaborations among several components. The temporal constraints of the global requirements on the execution of the different activities imply certain coordination messages between the different system components. The paper presents a transformation algorithm that derives, from a given global behavior, the local behaviors for each of the system components including the exchange of coordination messages for the global synchronization of the activities. The global behavior is defined in terms of standard sequencing operators that correspond to the concepts found in UML Activity Diagrams and similar formalisms; for sequential execution, weak and strong sequencing is distinguished. The derived component behaviors ensure that their joint execution satisfies the ordering constraints of the global requirements model, they avoid any possible race conditions, and introduce a relatively small number of coordination messages. In many cases, these messages are required anyway for carrying the dataflow which underlies the global requirements model.

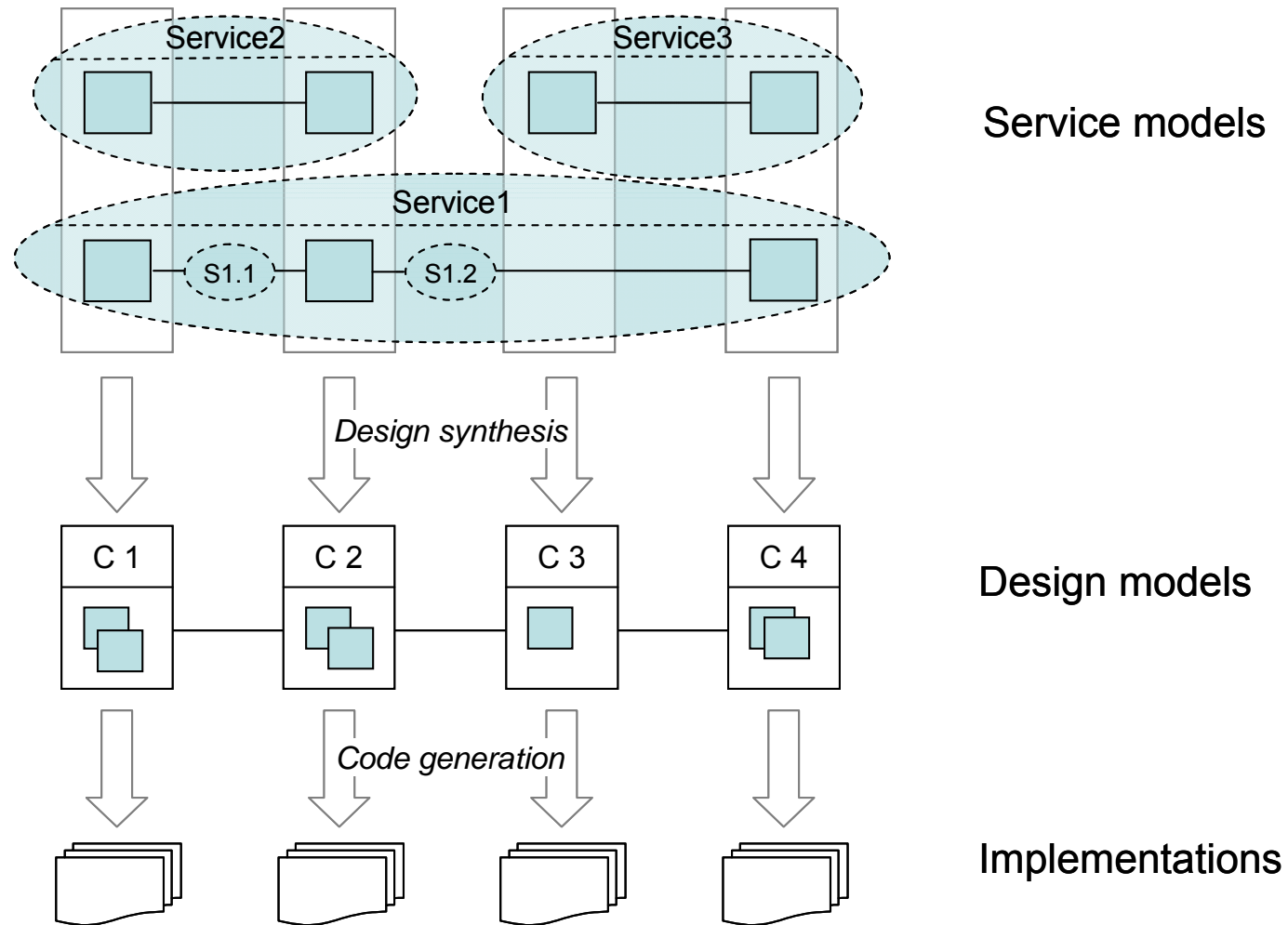


The problem (early phase of development process)

- Define
 - Global functional requirements
 - Non-functional requirements
- Make high-level architectural choices
 - Identify system components
 - Define underlying communication service
- Derive dynamic behavior of system components
 - Required messages to be exchanged and order of exchanges
 - Coding of message types and parameters



The problem – a figure



The problem

(functional and non-functional requirements)

- Define
 - Global functional requirements
 - Non-functional requirements
- Make high-level architectural choices
 - Identify system components
 - Define underlying communication service
- Derive dynamic behavior of system components
 - Required messages to be exchanged and order of exchanges
 - Coding of message types and parameters



Notations for global requirements

- UML Sequence diagrams
- UML Activity Diagrams
- XPDL (workflow modeling)
- Use Case Maps
- BPEL (Web Services) – Note: defines centralized behavior
- WS-CDL (“choreography”)
- Collaborations (for communication services, work at the university of Trondheim, Norway)

Question:

How do they fit with the above



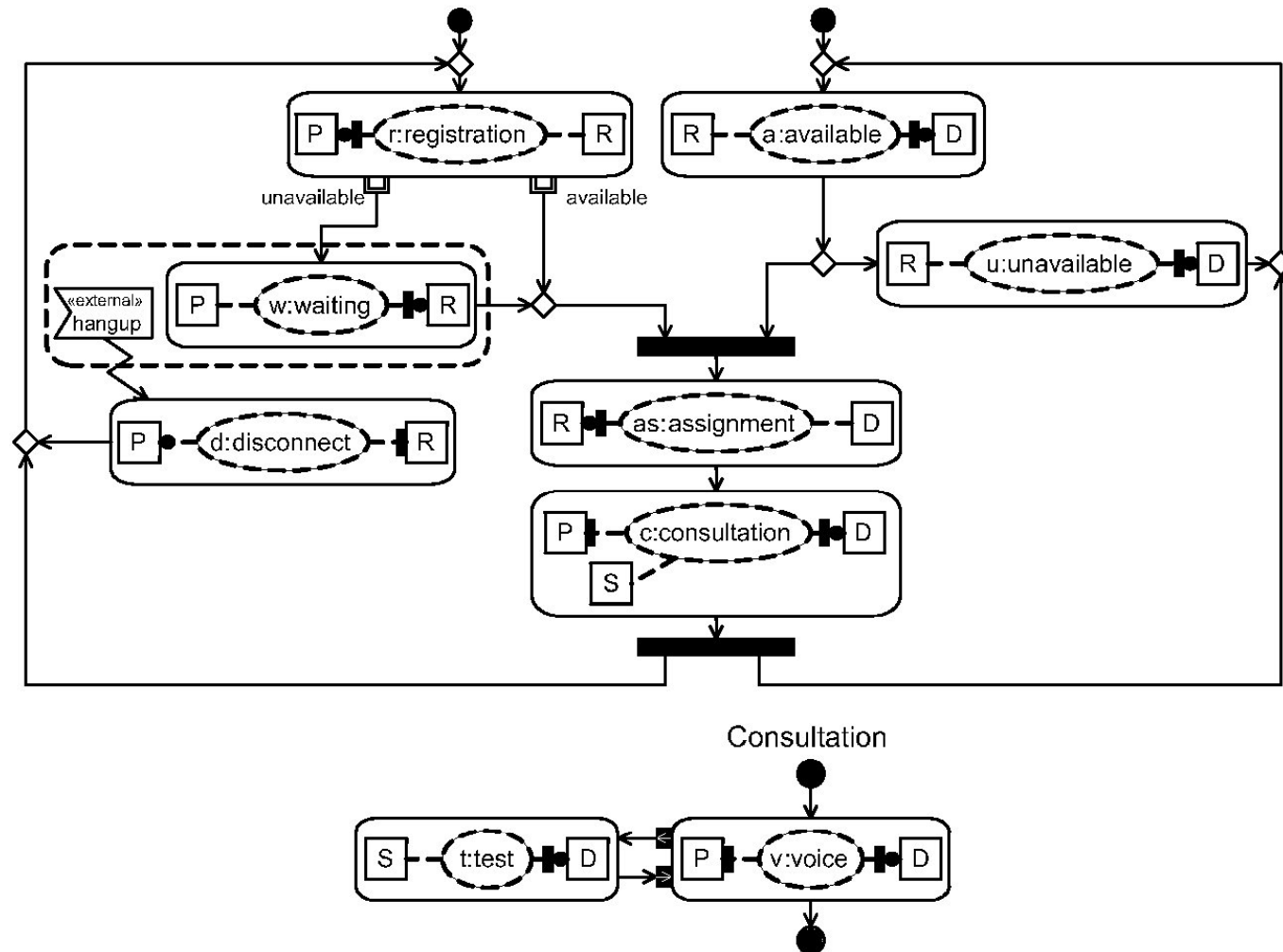
Overview

- Specification of requirements
 - Scenarios – activity diagrams – workflows
 - Collaboration activities
- Deriving component designs – previous work
- Weak sequencing
- Novelty of our approach
 - Receive buffers and message consumption
 - Choice indication messages
- Derivation algorithm
- Conclusions



Specification of requirements

Example: Telemedicine



Specification of requirements

Operators for
describing
temporal behavior

Construct	Notation
primitive activity	$\langle \text{action} \rangle^{(r)}$
invocation of a sub-collaboration	$\langle \text{subcol} \rangle^{(R)}$
strong sequence	$C_1 ;_s C_2$
weak sequence	$C_1 ;_w C_2$
choice	$C_1 \square C_2$
strong while loop	$C_1 *_s C_2$
weak while loop	$C_1 *_w C_2$
concurrency	$C_1 \parallel C_2$
interruption	$C_1 > C_2 \text{ else } C_3$



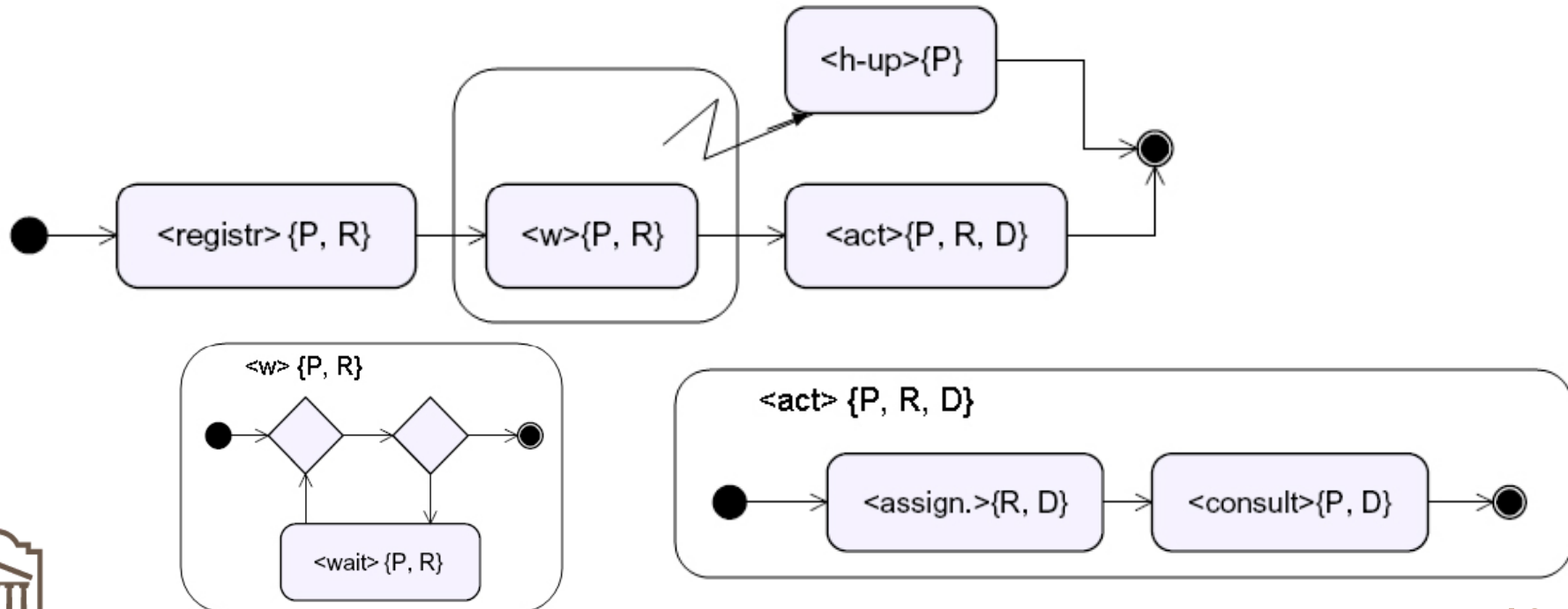
Specification of requirements

Telemedicine example

$\langle \text{telemed} \rangle = \langle \text{registr} \rangle_{\{sP_t, R\}} ;_w$
 $(\langle w \rangle_{\{P, R\}} \mid \langle \text{h-up} \rangle_{\{sP_t\}} \text{ else } \langle \text{act} \rangle_{\{P, R, D\}})$

$\langle w \rangle_{\{P, R\}} = \langle \text{wait} \rangle_{\{P_t, sR\}}^* \epsilon$

$\langle \text{act} \rangle_{\{P, R, D\}} = \langle \text{assign} \rangle_{\{sR_t, D\}} ;_w \langle \text{consult} \rangle_{\{P_t, sD_t\}}$



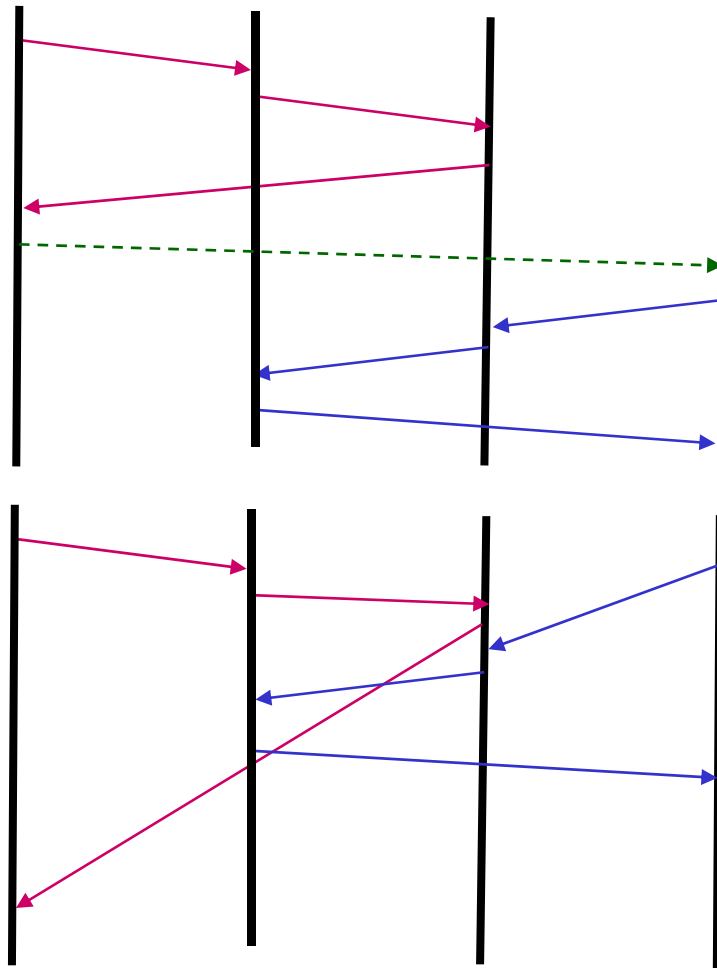
Deriving component designs – previous work

- Initially, only strong sequencing, choice and concurrency operators (Bochmann and Gotzhein, 1986)
- Later: notations based on FSMs, LOTOS, or Petri nets
- **Main conclusions:**
 - Strong sequencing requires coordination messages
 - e.g. for “C1 ; C2 ” from terminating roles of C1 to starting roles of C2
 - Algorithm for calculating starting and terminating roles
 - **Choice propagation:** need for unique message parameters



(Introduced for High-Level MSCs)

Example



strongly sequenced
(blue after red)

→ Coordination message

weakly sequenced
(blue after red)

(there are often race conditions)



Novelty of our approach

Include weak sequencing, loops and interruption

- How to avoid race conditions between the reception of different messages:
 - **Receive buffer pool**: distinguish between message **reception** and **consumption** (readiness for consumption is determined by receiver)
- **When is a component ready for the next sub-collaboration ?**
 - Difficulty in case of Choice: $C1 \ [] \ C2$. Consider a given component:
 - If involved in $C1$ and $C1$ is chosen it will receive a message to start its activities for $C1$ and it knows when it has completed its activities
 - If involved in $C1$, but not in $C2$, and $C2$ is chosen: it will not be informed and will never know whether $C2$ was chosen or whether it should wait for the message starting $C1$.
 - Choice indication message
 - Sent by a component involved in $C2$ to each component not involved in $C1$ when $C2$ is executed



Derivation algorithm

- Preliminary Step - **Architectural choices**: allocate collaboration roles to different system components
- Step 1a: Calculate **starting, terminating and participating roles** for each sub-collaboration (see Table 2)
 - Step 1b: Use **architectural choices** to determine starting, terminating and participating components for each sub-collaboration (**mapping roles to components**)
- Step 2: For each component, use recursively defined **transformation function** to derive the behavior of the component from the global requirements (see Table 3)



Step 2 : Coordination messages

- **Flow message** (as used earlier for strong sequencing)
- **Choice indication message**
- **For interrupts**
(transaction semantics: commit or abort)
 - **Interrupt-enable message**
 - **Interrupt message**
 - **Interrupt Flow message:** has additional parameter indicating the occurrence of an interrupt (like commit/abort message for distributed transactions)



Extract from Table 3 (i)

$C = \langle \text{action} \rangle^{(r)}$	$T_c(C) = \text{if } \text{Alloc}(r) = c \text{ then } \langle \text{action} \rangle \text{ else } \epsilon$
$C = \text{invoke } \langle \text{subcol} \rangle^{(R)}$	$T_c(C) = \text{if } c \text{ in } \text{Alloc}(R) \text{ then } \langle \text{subcol} \rangle \text{ else } \epsilon$
$C = C_1 ;_s C_2$	$T_c(C) = T_c(C_1) \text{ “;” } \text{SFM}(C_1, C_2) \text{ “;” } \text{RFM}(C_1, C_2) \text{ “;” } T_c(C_2)$ $\text{SFM}(C_1, C_2) = \text{if } c \text{ in } \text{Alloc}(\text{TR}(C_1)) \text{ then}$ “send fm(x) to all c' in $(\text{Alloc}(\text{SR}(C_2)) - \{c\})$ ” $\text{RFM}(C_1, C_2) = \text{if } c \text{ in } \text{Alloc}(\text{SR}(C_2)) \text{ then}$ “receive fm(x) from all c' in $(\text{Alloc}(\text{TR}(C_1)) - \{c\})$ ” ”
$C = C_1 ;_w C_2$	$T_c(C) = T_c(C_1) \text{ “;” } T_c(C_2)$



Extract from Table 3 (ii)

$$C = C_1 \mid > C_2 \text{ else } C_3$$

We assume that C_2 has the form “ $\langle \text{action} \rangle^{(r)} ;_s C_2'$ “.

$T_c(C) = \text{NormalBeh} \parallel^* \text{InterruptBeh}$ (variables: Interr, Interrupted, I-Enabled)

NormalBeh =

if c *in* $\text{Alloc}(\text{PR}(C_1))$ *then* “ $(T_c(C_1) \mid > (\text{wait}(\text{Interr}); \text{Interrupted} := \text{true};) \text{ else } \varepsilon);$ ”

if c *in* $\text{Alloc}(\text{TR}(C_1))$ *then* “send $\text{fim}(x, \text{Interrupted})$ to all c' in

“ $(\text{Alloc}(\text{SR}(C'_2)) \cup \text{Alloc}(\text{SR}(C_3))) - \{c\}$;”

if c *in* “ $(\text{Alloc}(\text{SR}(C'_2)) \cup \text{Alloc}(\text{SR}(C_3)))$ *then* “

“ (for all c' in $(\text{Alloc}(\text{TR}(C_1)) - \{c\})$ do (receive $\text{fim}(x, i)$; *if* i *then* $\text{Interrupted} := \text{true}$);

if not Interrupted *then* $\text{DOcim}_c(C_3, C'_2)$;)

$\parallel^* (\text{wait}(\text{Interrupted}); \text{DOcim}_c(C'_2, C_3))$) “

else “ $(\text{DOcim}_c(C'_2, C_3) \parallel \text{DOcim}_c(C_3, C'_2))$; “

InterruptBeh = *if* $c = r$ *then* “ *if* c *in* $(\text{Alloc}(\text{SR}(C_1)))$ *then* “ $\text{I-Enabled} := \text{true}$; “ *else*

“for all c' in $(\text{Alloc}(\text{SR}(C_1)) - \{c\})$ do (receive $\text{iem}(z)$ from c' ; $\text{I-Enabled} := \text{true}$)

$\parallel (\text{wait}(\text{I-Enabled}); \langle \text{action} \rangle$ (* this may never happen *);

$\text{Interr} := \text{true}$; send $\text{im}(z)$ to all c' in $(\text{Alloc}(\text{PR}(C_1)) - r)$; “

else (* c not equal r *) (*if* c *in* $(\text{Alloc}(\text{SR}(C_1)))$ *then* “send $\text{iem}(z)$ to r ; “

if c *in* $(\text{Alloc}(\text{PR}(C_1)))$ *then*

“(receive $\text{im}(z)$ from r (*may not happen *); $\text{Interr} := \text{true}$;)”



Example: Telemedicine

(high-level view: assuming $\langle \text{registr} \rangle$, $\langle w \rangle$ and $\langle \text{act} \rangle$ are given)

$$\begin{aligned} T_c(\langle \text{telemed} \rangle) &= T_c(\langle \text{registr} \rangle); T_c(\langle w \rangle \mid \langle \text{h-up} \rangle; \varepsilon \text{ else } \langle \text{act} \rangle) \\ &= T_c(\langle \text{registr} \rangle); (\text{NormalBeh } c \mid \mid^* \text{InterruptBeh } c) \end{aligned}$$

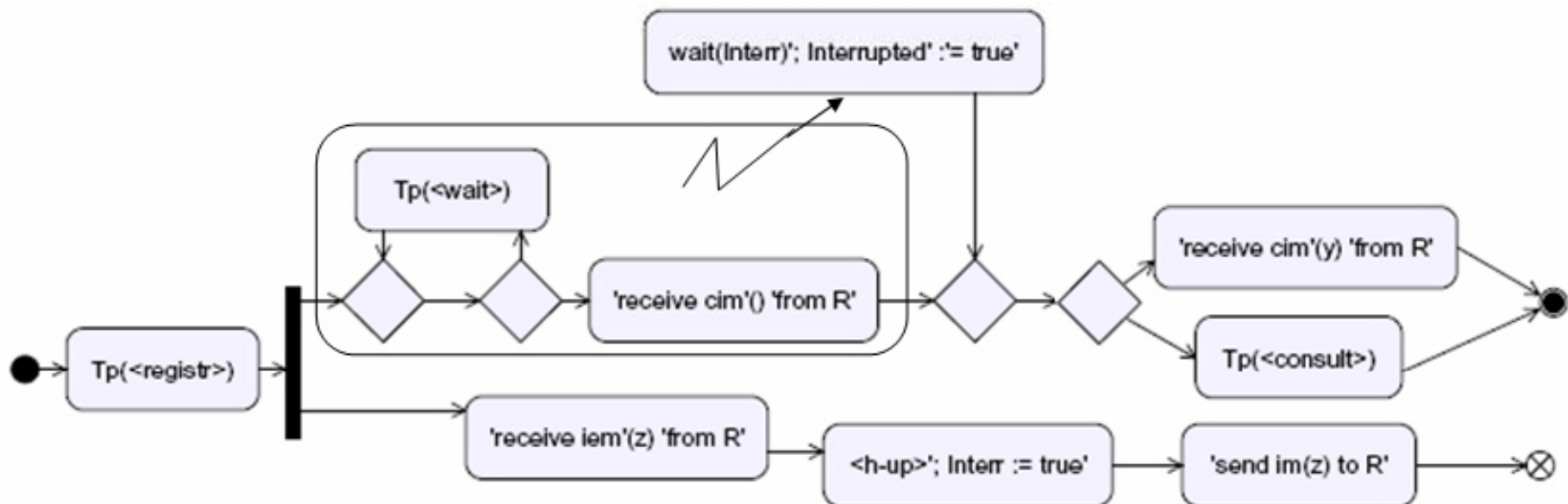
for $c = P, R$ and D and where

$$T_D(\langle \text{registr} \rangle) = \varepsilon$$
$$\begin{aligned} \text{NormalBeh } P &= (T_P(\langle w \rangle) \mid \langle \text{wait}(\text{Interr}); \text{Interrupted} := \text{true}; \rangle \text{ else } \varepsilon); \\ &(\text{receive cim}(y) \text{ from } R \mid \mid T_P(\langle \text{act} \rangle)) \end{aligned}$$
$$\text{InterruptBeh } P = \text{receive iem}(z) \text{ from } R; \langle \text{h-up} \rangle; \text{Interr} := \text{true}; \text{send im}(z) \text{ to } R$$
$$\begin{aligned} \text{NormalBeh } R &= (T_R(\langle w \rangle) \mid \langle \text{wait}(\text{Interr}); \text{Interrupted} := \text{true}; \rangle \text{ else } \varepsilon); \\ &(\text{receive fim}(x, i) \text{ from } P; \text{if } i \text{ then } \text{Interrupted} := \text{true}; \text{if not } \text{Interrupted} \\ &\text{then } T_R(\langle \text{act} \rangle) \mid \mid \langle \text{wait}(\text{Interrupted}); \text{send cim}(y) \text{ to } D \text{ and } P \rangle) \end{aligned}$$
$$\text{InterruptBeh } R = \text{send iem}(z) \text{ to } P; \text{receive im}(z) \text{ from } P; \text{Interr} := \text{true}$$
$$\text{NormalBeh } D = T_D(\langle \text{act} \rangle) \mid \mid \text{receive cim}(y) \text{ from } R$$
$$\text{InterruptBeh } D = \varepsilon$$


Example: Telemedicine

Behavior of P component

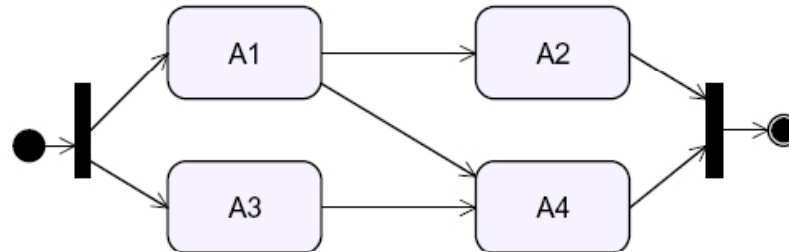
Note: <registr>, <wait> and <consult> are primitive collaborations whose behavior is not necessarily known, only their starting, terminating and participating roles must be known.



Aspects not covered in this paper

- Proof of correctness of derivation algorithm
- More complex temporal order relationships

- Example:



- Data flow
- Concurrent sessions and dynamic selection of collaboration partners



Conclusions

- **Distributed system design in several steps:**
 1. **Requirements model:** global behavior in terms of certain activities (collaborations) and their temporal ordering.
 2. **Architectural choices:** Based on architectural and non-functional requirements, allocate collaboration roles to system components
 3. **Deriving component behavior**
- **Step 3 can be automated.**
- **Proposed modeling language for requirements:**
 - **Activity diagrams** where an activity may be a collaboration between several roles
 - **Identify roles** for each activity (participating, starting, terminating)
 - **Hierarchical description** of requirements in terms of sub-activities (collaborations)
- **Many fields of application:** workflow, service composition for communication services, e-commerce applications, Web Services.
- **Implementation of the derivation algorithm**

