$\mathcal{CS}-\mathcal{F}$ *low*: The Engineering of Pervasive Workflows

Hussein Zedan

Opatija, September 2012

STRL

Why?

- A workflow is a set of activities, each performs a piece of functionality within a given context and may be constrained by some security requirements. These activities are coordinated to collectively achieve a required business objective.
- The specification of such coordination is presented as a set of "execution constraints" which include *parallelisation, serialisation, restriction, alternation, compensation* and so on.

Why?

- Activities within workflows could be carried out by humans, various software-based application programs, or processing entities according to some organisational rules, such as meeting deadlines or performance improvement.
- Workflow execution can involve a large number of different participants, services and devices which may cross the boundaries of various organisations and accessing variety of data.

Therefore....

- Modern workflows are CRITICAL systems.
- They are
  - Highly distributed
  - Context-critical
  - Security-critical
  - Time-critical
  - Business-critical
- We need a unified model within which modern workflows can be modelled, analysed and, being critical, be provably correct

Computational Model

- our model has three distinct components:
  - Context
  - Activity and
  - Guard

Context

- Contexts can take a variety of forms : different platforms and operating systems, hand-held devices, web-services, etc.
- A context is characterised by, what we call *context frame* , which is a set of variables (or attributes) of interests.

  For

  - PDAs attributes of interests could be *processor speed, memory size, battery life time* .
  - a human context, *age, qualification, work experience* may be of interest.
  - a patient context, *body temperature, blood pressure, kidney functions* are more appropriate attributes.

Context

- The changes in the attributes are only **observed** and then acted upon.
- Context attributes are predicated upon to form a **context guard** so as a decision may be taken to execute an activity or choose different but more suitable context, etc.
- Context guards are also important as mechanisms to express **security policies** and for the design of variety of enforcement mechanisms of these policies that, for example, controls access to sensitive data/information.

Context

- An activity in our model does not exist in isolation. Indeed it requires a context to house it.
- Activities within a workflow **move into** a context to be executed but may choose to **move out** to another context in order to complete its functionality.
- In this way, context can be nested in a larger context in a compositional fashion.

Activity

- An *activity* is a computational unit that describes a piece of work that contributes toward the accomplishment of a given goal.
- An activity has
  - *a goal*,
  - *an input*,
  - *an output*,
  - *performed in a particular order*,
  - *associated with a particular context* ,
  - *uses resources/information*,
  - *may affect more than one organisation unit*,
  - *creates some value for users.* and
  - *properly terminates* – in the same or in a different context.

STRL

Activity

- An activity starts in one context but may terminate in a different context. This means that an activity has the ability to be **mobile** and moves from one context to another.
- But as an activity in our model is tightly associated with a context, mobility occurred at a context level , i.e. an activity moves with its context.

Activity

- Activities may be composed concurrently to produce a new activity which terminates if and only if all of its components terminate, i.e. we adopt the *distributed termination* convention.
- we assume a single clock for an instant of a workflow.
- Activities are also composed in alteration and in a non-deterministic fashions.
- An activity can also be conditionally executed after the passability of its condition or guard.

Guards

- Each activity/context is governed by a set of *context* and/or *security* policies/constraints which are continually changing due to either the occurrence of an event and/or the passage of time.
- *access control* policies: *subjects* – such as human, activities, platforms; *object* – This is a resource which is there to be used. It has a state where a subject can alter once it is granted to do so and *action* – is an activity where once the access is granted, it can be executed.
- *ECA* is another formulation of policy.

$\mathcal{CS}-\mathcal{F}$ *low*: Graphical Representation

$$\boxed{Context\_id : \langle Frame \rangle}$$

$$\boxed{\begin{array}{l} Policies\_Constraints \\ \parallel Context\_Constraints \\ \parallel Behavioural\_Description \end{array}}$$

where *Frame* is given as:

$$\boxed{Frame :: \langle Context\_Attributes \rangle}$$

$\mathcal{CS}-\mathcal{F}$ *low*: Graphical Representation

$PDA_1 : \langle s, w, p \rangle$
$(PDA_1(s, w, p) \parallel P_{PDA_1}(Ch, x, y, z))$

STRL

$\mathcal{CS}-\mathcal{F}\,low$: Graphical Representation

$Ali : \langle loc, t \rangle$

$PDA_1 : \langle s, w, p \rangle$

$(PDA_1(s, w, p) \parallel P_{PDA_1}(Ch, x, y, z))$

$\parallel P_{Ali}$

$\mathcal{CS}-\mathcal{F}\,low$: Graphical Representation

$PDA_1 : \langle s, w, p \rangle$
$(PDA_1(s, w, p) \parallel P_{PDA_1}(Ch, x, y, z))$

$\parallel$

$PDA_{10} : \langle s_1, w_1, p_1 \rangle$
$PDA_{10}(s_1, w_1, p_1) \parallel$ $P_{PDA_{10}}(Ch, a, b, c)$

STRL

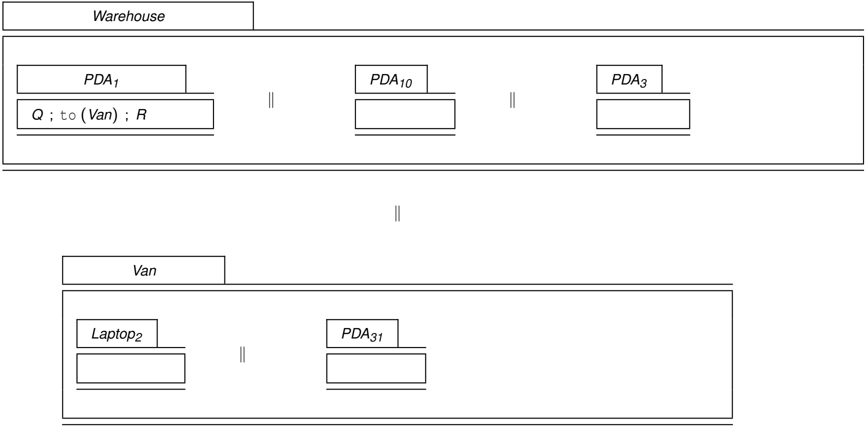
$\mathcal{CS}-\mathcal{F}$ *low*: Graphical Representation

- Activities can communicate by exchanging messages over channels.
- The communication is synchronous and is modelled using handshake message passing communication primitives: $\mathtt{C} \,!\, \mathtt{v}$ (output) and $\mathtt{C} \,?\, \mathtt{x}$ (input).
- $P_{PDA_1} \ \widehat{=}\ \ldots; \mathtt{Ch} \,!\, \mathtt{Temp}_{\mathtt{value}} \,; \ldots$

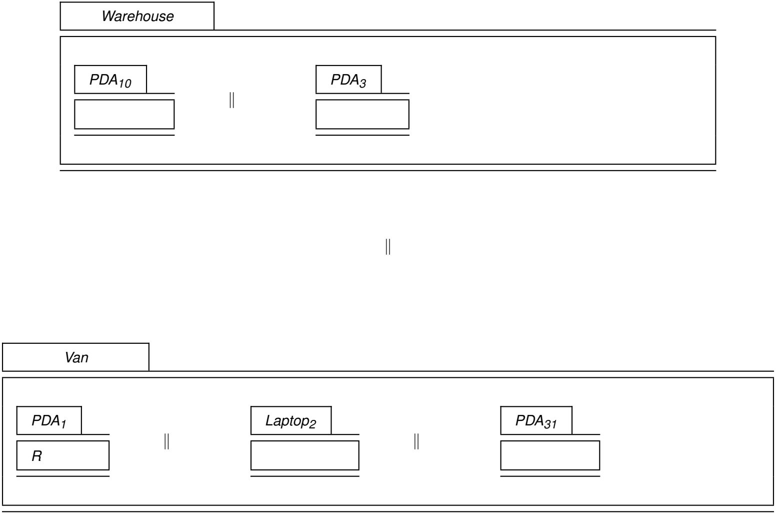
  and

  $P_{PDA_{10}} \ \widehat{=}\ \cdots ; \mathtt{Ch} \,?\, x \,; \cdots.$

$\mathcal{CS}-\mathcal{F}$ *low*: Mobility



$\mathcal{CS}-\mathcal{F}$ *low*: Mobility



$\mathcal{CS}-\mathcal{F}\,low$: textual Representation

$$
\begin{array}{lcl}
P, Q & ::= & \mathtt{skip} \mid \mathtt{abort} \mid x := v \mid \mathtt{delay}(t) \mid [t_1 \ldots t_n]\, P \mid c\,!\,v \mid c\,?\,x \\
 & & \mid \alpha\, \langle \tilde{x} \rangle : \{P\} \mid \mathtt{to}(\alpha) \mid \mathtt{var}\ \tilde{x}\ \mathtt{in}\ P\, \{Q\} \mid \mathtt{chan}\ \tilde{c}\ \mathtt{in}\ P\, \{Q\} \\
 & & \mid \mathtt{in}\ \alpha \cdot P(\tilde{x}) \mid P\,;Q \mid P \parallel Q \mid P \triangleright_t^G Q \mid \mathtt{while}\ G \cdot\ \mathtt{do}\ P\ \mathtt{od} \\
 & & \mid [p_1] : G_1 \rightarrow P \,\square\, [p_2] : G_2 \rightarrow Q \\
G & ::= & \mathtt{true} \mid b \mid \mathtt{not}\ G \mid G_1\ \mathtt{and}\ G_2 \mid \mathtt{somewhere}(\alpha) \cdot G
\end{array}
$$

STRL

Context

$\alpha \, \langle \tilde{x} \rangle$:

$\{$

P

$\}$

Context: *Ikea*

$$Ikea : \langle \quad \rangle$$

$$\{$$

$$P_{Ikea} \parallel PDA_{23} : \{ Q \}$$

$$\}$$

Context: *Ikea*

$$Ikea : \langle Damp_{Level}, Smoke_{Alarm} \rangle$$

$$\{$$

$$P_{Ikea} \parallel PDA_{23} : \{Q\}$$

$$\}$$

Context: *Ikea*

$$
\begin{array}{l}
Ikea : \langle Damp_{Level}, Smoke_{Alarm} \rangle \\
\quad \{ \\
\qquad P_{Ikea} \parallel PDA_{23} : \\
\qquad\qquad\qquad \{ \\
\qquad\qquad\qquad\quad TakeStock \; ; \\
\qquad\qquad\qquad\quad \mathtt{not}\ (Damp_{Level} \geq 25 \vee Smoke_{Alarm}) \rightarrow \\
\qquad\qquad\qquad\qquad\qquad \mathtt{to}(Van) \; ; \; Place\ Order \\
\qquad\qquad\qquad \} \\
\quad \}
\end{array}
$$

STRL

Context, Location and Holes

- Central to our model is that activities do not operate in the ether. They need contexts which identify their *locations* and within which they execute, terminate and may move out of them to another contexts.
- Unlike other formalisms, the notion of **holes** exists in which processes can move to. This makes the models rather clumsy and static with a fixed number of holes.

Context, Location and Holes

- The term "context" is used here instead of "location" for the later can indicate/require notions such as
  - Proximity,
  - Coordinates,
  - Neighborhoods, etc.

  which in our view adds extra complication which is not needed.

Context, Location and Holes

- Two special contexts, which we call *SKIP* and *STOP*:
  - *SKIP* is an empty context and nothing is happening in it and there are no observables.
  - *STOP* is the most un-inhabited context and will remain so forever! Further, if it moves into another context, it makes the host context un-inhabitable too. It is a context that needs to be avoided at all cost.

Context, Location and Holes

- Context, like activities, can communicate synchronously via channels. Whenever a context moves, its channels move with it. This is a powerful mobility notion as all what we needed is a single label to identify a context. The connectivity's between contexts (or their exact coordinates, neighborhoods, etc.) becomes irrelevant.

Examples: Adaptable activities

$$ShopFloor :$$

$$\{$$

$$win \parallel linx$$

$$\}$$

STRL

Examples: Adaptable activities

```
win :
{
  var f in edit
  {
    notepad(f)
  }
}
```

Examples: Adaptable activities

```
linx :
{
  var f in edit
  {
    emacs(f)
  }
}
```

Examples: Adaptable activities

$$\textit{Employee}: \quad \{ \quad \texttt{somewhere}\,(\textit{ShopFloor}) \cdot \textit{edit}(\textit{file}) \quad \}$$

STRL

Examples: Adaptable activities

$$win:$$

$$\{$$

$$\texttt{var}\ f\ \texttt{in}\ edit$$

$$\{$$

$$notepad(f)$$

$$\}$$

$$\|$$

$$Employee:$$

$$\{$$

$$\texttt{somewhere}(ShopFloor)\cdot\ edit(file)\quad\}\quad\}$$

Examples: Adaptable activities

$linx :$

$\{$

$\texttt{var}\ f\ \texttt{in}\ edit$

$\{$

$emacs(f)$

$\}$

$\|$

$Employee :$

$\{$

$\texttt{somewhere}(ShopFloor) \cdot\ edit(file)\ \ \}\ \ \}$

Examples: Policies – *ECA*

$$
\begin{array}{l}
\texttt{while true} \\
\texttt{do} \\
\quad \{ \\
\qquad G_{event_1}\ \texttt{and}\ G_{condition_1} \ \rightarrow\ P \\
\qquad \Box \\
\qquad G_{event_2}\ \texttt{and}\ G_{condition_2} \ \rightarrow\ Q \\
\qquad \ldots \\
\qquad \Box \\
\quad \}\ \ \texttt{od}
\end{array}
$$

Examples: Policies – *ECA*

$$System \ \widehat{=}$$

$$Flows \parallel EventAnalyser \parallel ECA$$

LAYERS: Assumptions

Without lose of generality, we assume that

- There is only one parallel operator, $\|$, in our system. Nesting concurrency can be dealt with by applying the transformation to the most inner $\|$ and continue to move to the outer constructs.
- The length of all activities in the system are the same. This can be easily achieved using the semantics of `skip`. I.e.

$$\texttt{skip} \; ; \; S \equiv S \; ; \; \texttt{skip} \equiv S$$

$$
\begin{array}{lcl}
C_1\ \langle \tilde{a} \rangle : & & C_2\ \langle \tilde{b} \rangle : \\
\quad \{ & & \quad \{ \\
\quad \mathtt{var}\ \tilde{x},\ \tilde{y}\ \mathtt{in} & & \quad \mathtt{var}\ \tilde{x}_1,\ \tilde{y}_1\ \mathtt{in} \\
\qquad \{ & & \qquad \{ \\
\qquad\quad P_1; & & \qquad\quad Q_1; \\
\qquad\quad P_2; & & \qquad\quad Q_2; \\
\qquad\quad P_3; & \| & \qquad\quad Q_3; \\
\qquad\quad P_4; & & \\
\qquad\quad P_5; & & \\
\qquad \} & & \qquad \} \\
\quad \} & & \quad \}
\end{array}
$$

$C_1 \langle \tilde{a} \rangle$:		$C_2 \langle \tilde{b} \rangle$:
$\{$		$\{$
`var` $\tilde{x}, \tilde{y}$ `in`		`var` $\tilde{x}_1, \tilde{y}_1$ `in`
$\{$		$\{$
P_1;		`skip`;
P_2;		Q_1;
P_3;	$\parallel$	`skip`;
P_4;		Q_2;
P_5;		Q_3;
$\}$		$\}$
$\}$		$\}$

Let us consider an example:

$$\mathcal{R} \mathrel{\widehat{=}} C_1 \ \langle \tilde{a} \rangle : \{ \ \mathcal{P} \ \} \parallel C_1 \ \langle \tilde{b} \rangle : \{ \ \mathcal{Q} \ \}$$

$C_1 \ \langle \tilde{a} \rangle$:

$\{$

$\texttt{var}\ x,\ y,\ z,\ chan_1\ \texttt{in}$

$\{$

$y := y + x;$

$z := y \times z;$

$chan_1 \ ! \ z;$

$\}$

$\}$

$\parallel$

$C_2 \ \langle \tilde{b} \rangle$:

$\{$

$\texttt{var}\ x_1,\ chan_1\ \texttt{in}$

$\{$

$chan_1 \ ? \ x_1;$

$x_1 := x_1 \times x_1;$

$\}$

$\}$

Definition

A layer, L of a workflow, $\mathcal{S}$, is a logical horizontal partition that cut across all concurrent threads of $\mathcal{S}$ $\square$.

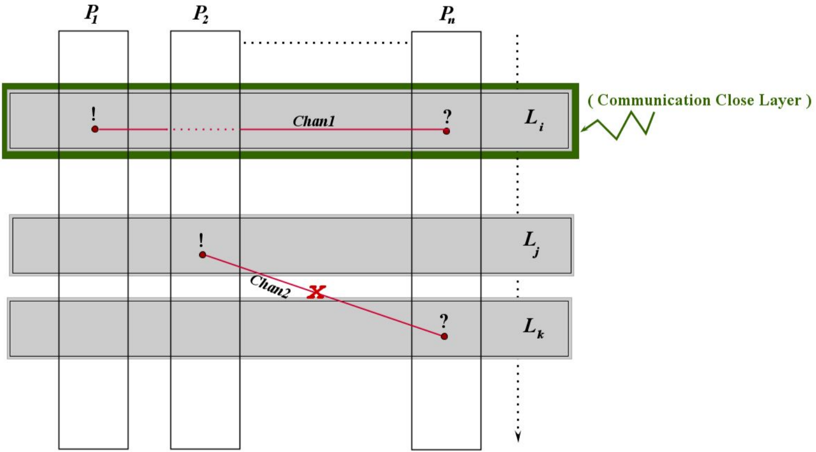
Definition

A Layer L is called communicating layer if it contains at least one communication primitive. It is called communication-closed if a communication starts and terminates in the same layer.

A non-communicating layer is that which contains no communication primitives.

Definition

A super-structure over a workflow, $\mathcal{S}$, is a quasi-sequential composition of layers from $\mathcal{S}$ $\square$.


P1
P2
Pn
(Communication Close Layer)
!
Chan1
?
Li
!
Lj
Chan2
?
Lk

$L_1 \mathrel{\widehat{=}}$	$y := y + x$	$\parallel$	$chan_1 \; ? \; x_1$
$L_2 \mathrel{\widehat{=}}$	$z := y \times z$	$\parallel$	$x_1 := x_1 \times x_1$
$L_3 \mathrel{\widehat{=}}$	$chan_1 \; ! \; z$	$\parallel$	`skip`
$L_4 \mathrel{\widehat{=}}$			
	$y := y + x;$		`skip`;
		$\parallel$	
	$z := y \times z$		`skip`
$L_5 \mathrel{\widehat{=}}$			
	$z := y \times z \; ;$		`skip` ;
	$chan_1 \; ! \; z \; ;$	$\parallel$	$chan_1 \; ? \; x_1 \; ;$
	`skip`		$x_1 := x_1 \times x_1$

The following are some super-structures:

1. $\mathcal{S}_{\mathcal{R}_1} \mathrel{\widehat{=}} \mathcal{R}$
2. $\mathcal{S}_{\mathcal{R}_2} \mathrel{\widehat{=}} L_1 \; ; \; L_2 \; ; \; L_3$
3. $\mathcal{S}_{\mathcal{R}_3} \mathrel{\widehat{=}} L_4 \; ; \; L_5$

Under what condition(s) will a super-structure workflow be equivalent to the original one? .

It is clear that, in the example above, L_2 and L_4 are non-communicating layers while L_1, L_3 and L_5 are communication-closed. $\mathcal{S}_{\mathcal{R}_3}$ and $\mathcal{S}_{\mathcal{R}_1}$ are a quasi-sequential workflow whilst $\mathcal{S}_{\mathcal{R}_2}$ is not.

Theorem

For any $\mathcal{CS}-\mathcal{F}low$ workflow system $\mathcal{S}$ there exist a semantically equivalent quasi-sequential system, $\mathcal{S}_{\mathcal{L}}$.

Proof: Choices

$$G_1 \rightarrow P$$

$$\square$$

$$G_2 \rightarrow Q$$

STRL

Proof: Choices

The following workflow, $\mathcal{S}'$, is a such safe decomposition:

$\mathcal{S}' \ \widehat{=}$

$$\left\{ \begin{array}{lcr} \left[\begin{array}{l} G_1 \ \rightarrow \ P \ ; \ GFlag \ := \ \mathtt{false} \\ \Box \\ G_2 \ \rightarrow \ GFlag \ := \ \mathtt{true} \end{array} \right] & ; & (S_{11}) \\ \\ \left[\begin{array}{l} GFlag \ \rightarrow \ Q \\ \Box \\ \mathtt{not} \ GFlag \ \rightarrow \ \mathtt{skip} \end{array} \right] & & (S_{12}) \end{array} \right\}$$

Now, if we have another workflow $\mathcal{D}$ of the same structure as $\mathcal{S}$ then

$$\mathcal{S} \ \| \ \mathcal{D}$$

can be "safely" decomposed into the structure:

$((\mathcal{S}_{11} \parallel \mathcal{D}_{11}\ )\ ;\ (\mathcal{S}_{12} \parallel \mathcal{D}_{12}\ ))\Box\ ((\mathcal{S}_{21} \parallel \mathcal{D}_{21}\ )\ ;\ (\mathcal{S}_{22} \parallel \mathcal{D}_{22}))$

where each $\mathcal{S}_{ij}$, for all $i, j\ = 1, 2$, is either a non-communicating layer or a communication-closed layer.

Proof: Choices

This can be rewritten as

$((\mathcal{S}_{11} \parallel \mathcal{D}_{11})\Box(\mathcal{S}_{21} \parallel \mathcal{D}_{21}));$

$((\mathcal{S}_{11} \parallel \mathcal{D}_{11})\Box(\mathcal{S}_{22} \parallel \mathcal{D}_{22}));$

$((\mathcal{S}_{12} \parallel \mathcal{D}_{12})\Box(\mathcal{S}_{21} \parallel \mathcal{D}_{21}));$

$((\mathcal{S}_{12} \parallel \mathcal{D}_{12})\Box(\mathcal{S}_{22} \parallel \mathcal{D}_{22}))$

These structures demonstrate that layers can be composed, respectively, as a series of alternative or sequentially. In fact, structures such as iteration, conditional, interrupt, etc. can also be used.

Proof: Iterations

We assume that

1. Loops are *finite*
2. Communication symmetry is assured (i.e., communication-deadlock free)

It should be noted that due to (1) above, a finite loop can be replaced as a set of sequentially composed statements and because of (2), we can always ensure (using `skip`) that each layer is communication-closed layer.

Proof: Iterations

Using

$\texttt{while}\ G\ do\{P\} \equiv G \rightarrow P\ ;\ (\texttt{while}\ G\ do\{P\})$

then, if we have

$$\texttt{while}\ G\ do\{P\} \parallel Q$$

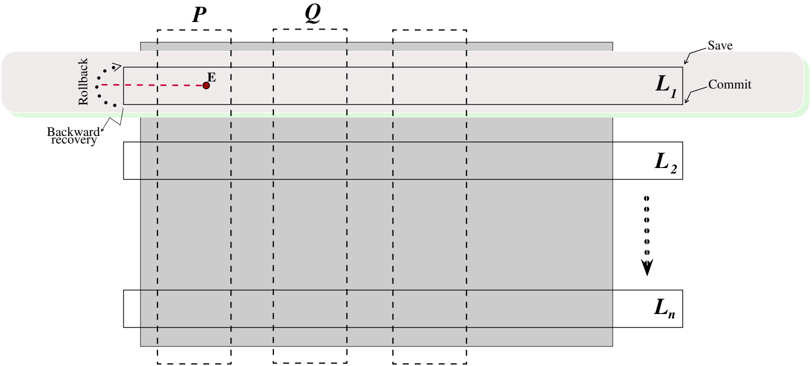
Then we can transform this to the semantically equivalent $\mathcal{CS}-\mathcal{F}\,low$ system

$$((G \rightarrow P) \parallel Q)\ ;\ (\texttt{while}\ G\ do\{P\})$$

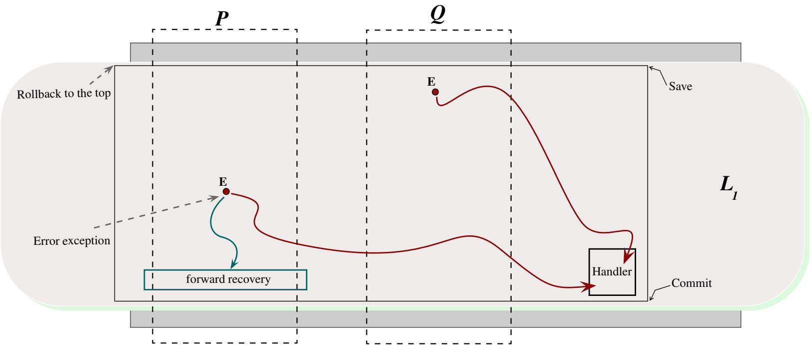
Then, we layer $((G \rightarrow P) \parallel Q)$ into communication-closed layers (depending on the structure of P and Q, and repeat the process on the $\texttt{while}\ G\ do\{P\}$, and so on.

STRL

Layers: Fault-Tolerance



Layers: Fault-Tolerance



Layers Usages

- Analysis
  - The idea here is to transform the existing $\mathcal{CS}-\mathcal{F}$ *low* design into a semantically equivalent communication-closed layer design in which the analyses are easier than the original one. The rational is that the resulting layer-design is quasi-sequential and hence all existing formalisms for sequential systems can be deployed.
- Design

Layers Design Methodology

1. **Requirements Decomposition.**
   - decompose the given workflow requirements into a number of sub-requirements which can be as fine or coarse grain as we wish. This process *iterative* in nature.
   - Experience has shown that, identifying, what we call *Actors* helps in specifying layer interfaces.
2. **Layer Design.**
   - Design layers which conform/satisfy its requirement.
   - The layers however have to be *communication-closed* layers.
3. **Integration.**
   - Compose/integrate all layers into a complete $\mathcal{CS}-\mathcal{F}$ *low* workflow.

Example: One-place Buffer – Layer Decomposition

We can easily identify two major layers:

1. *Initialisation.* Involves the *Buffer* and the *Authorised_User*, and using channel *push*.
2. *Operations.* This involves the *Buffer* and any other user.

Example: One-place Buffer – Layer Design

$$Init \;\widehat{=}$$

$$push\,!\,v \quad \| \quad (push\,?\,x\;;\;empty := \mathtt{false}) \;\|\; \mathtt{skip}$$

Example: One-place Buffer – Layer Design

$$Operation \mathrel{\widehat{=}} ($$

$$\mathtt{skip} \parallel \left[\begin{array}{l} \mathtt{while\ true\ do} \\ \quad \{ \\ \qquad empty \rightarrow push\,?\,x \\ \quad \square \\ \qquad \mathtt{not}\ empty \rightarrow pull\,!\,v \\ \quad \} \\ \qquad \mathtt{od} \end{array}\right] \parallel \left[\begin{array}{l} pull\,?\,x\,; \\ push\,!\,v \end{array}\right]$$

Example: One-place Buffer – Layer Design

It is clear that each of the above layers are communication-closed and the resulting quasi-sequential system is

$$Sys_L \ \widehat{=}\ Init \ ; \ Operation$$

Example: One-place Buffer – Integration

In this phase, the layers are integrated to obtain the final system:

$$Sys \mathrel{\widehat{=}} Authorised\_User \parallel Buffer \parallel User$$

where

$$Buffer \mathrel{\widehat{=}} ($$

$push\,?\,x\,;$

$empty := \mathtt{false}\,;$

$\mathtt{while\ true}$

$\mathtt{do}$

$\{$

$empty \rightarrow push\,?\,x$

$\square$

$\mathtt{not}\ empty \rightarrow pull\,!\,v\,;$

$empty := \mathtt{true}$

$\}$

Example: One-place Buffer – Integration

The users are modelled as

$$Users :: \; ($$

$$\begin{aligned} Authorised\_User \;&\widehat{=}\; push \,!\, v \\ User \;&\widehat{=}\; pull \,?\, x \;;\; push \,!\, v \end{aligned}$$

We note that, as the layers were designed communication-closed, then $Sys\_L \equiv Sys$.

Context-Aware Ward: $\mathcal{CAW}$

$$\mathcal{CAW} \;\widehat{=}\; ($$

$$
\begin{array}{l}
\mathit{nurse}\ \langle \tilde{x_n} \rangle\ :\ \{P_n\} \\
\| \mathit{bed} \langle \tilde{w} \rangle\ :\ \{P_b\} \\
\| \mathit{patient}\ \langle \tilde{w_1} \rangle\ :\ \{P_p\} \\
\| \mathit{nurse} - \mathit{office}\ \langle \tilde{w_2} \rangle\ :\ \{P_{n.o}\} \\
\| \mathit{medicine} - \mathit{room}\ \langle \tilde{w_3} \rangle\ :\ \{P_{m.r}\} \\
\| \\
\mathit{tray}\ \langle \tilde{x_t} \rangle\ :\ \{P_t \parallel \mathit{Cont_1}\ \langle \mathit{pat}_1,\ \tilde{a}_1 \rangle\ :\ \{P_1\} \\
\qquad \left(\begin{array}{ll}
\| & \mathit{Cont_2}\ \langle \mathit{pat}_2,\ \tilde{a_2} \rangle\ :\ \{P_2\} \\
\| & \cdots\cdots\cdots \\
\| & \mathit{Cont_k}\ \langle \mathit{pat}_k,\ \tilde{a_k} \rangle\ :\ \{P_k\} \\
\ \} &
\end{array} \right)
\end{array}
\tag{1}
$$

$$P_n \mathrel{\widehat{=}} \texttt{chan } chan_{n.t} \texttt{ in} \{$$

```
while true
```

$chan_{n.t}$! any ;

to(bed) ;

$[epr(P_i) \parallel HandOutDrug(P_i)]$

$chan_{n.t}$! any ;

to$(nurse - office)$

$$HandOutDrug \mathrel{\widehat{=}} \texttt{var}\ T,\ D,\ N,\ i\ \texttt{in}\{$$

```
while i ≤ N
do
    ([T]([D]GiveDrug) || delay(T) ) ;
    i = i + 1
od
```

STRL

Animation and Validation

In addition to equational theory and operational semantics for

$\mathcal{CS}-\mathcal{F}$ *low* we have

- Denotational semantics: A CCA-specification semantics
- Reduction semantics

Reduction Rules

$$P \rightarrow P' \quad \Rightarrow \quad \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P\} \rightarrow \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P'\} \qquad \text{(Reduction Var)}$$

$$P \rightarrow P' \quad \Rightarrow \quad \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P\} \rightarrow \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P'\} \qquad \text{(Reduction Chan)}$$

$$P \rightarrow P' \quad \Rightarrow \quad \alpha < \tilde{x} >: \{P\} \rightarrow \alpha < \tilde{x} >: \{P'\} \qquad \text{(Reduction Contxt)}$$

$$P \rightarrow P' \quad \Rightarrow \quad \mathcal{C}(P) \rightarrow \mathcal{C}(P') \qquad \text{(Reduction Contxt)}$$

$$P \rightarrow P' \quad \Rightarrow \quad P \parallel Q \rightarrow P' \parallel Q \qquad \text{(Reduction Par)}$$

$$P \equiv Q,\ Q \rightarrow Q',\ Q' \equiv P' \quad \Rightarrow \quad P \rightarrow P' \qquad \text{(Reduction } \equiv \text{)}$$

STRL

Reduction Rules

$$(Chan\,?\,\tilde{y})\;;\;P \parallel (Chan\,!\,\tilde{z})\;;\;Q$$
$$\rightarrow \quad P\{\tilde{y} \leftarrow \tilde{z}\} \parallel Q \qquad \text{(Reduction Com-1)}$$

$$\alpha : \{((Chan\,?\,\tilde{y})\;;\;P) \parallel Q\} \parallel \beta : \{((Chan\,!\,\tilde{z})\;;\;R) \parallel S\}$$
$$\rightarrow \quad \alpha : \{P(\tilde{y} \leftarrow \tilde{z}) \parallel Q\} \parallel \beta : \{R \parallel S\} \qquad \text{(Reduction Com-2)}$$

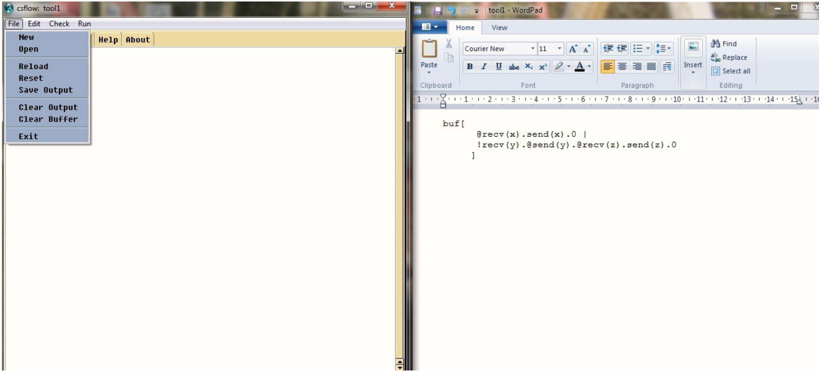
$$\alpha : ((Chan\,?\,\tilde{y})\;;\;P) \parallel Q) \parallel \beta : (\alpha : (Chan\,!\,\tilde{z})\;;\;R) \parallel S)$$
$$\rightarrow \quad \alpha : (P(\tilde{y} \leftarrow \tilde{z})) \parallel \beta : (R \parallel S) \qquad \text{(Reduction Com-3)}$$

$$\alpha : (\beta : (Chan\,?\,\tilde{y}\;;\;P) \parallel Q) \parallel \beta : (Chan\,!\,\tilde{z}\;;\;R) \parallel S$$
$$\rightarrow \quad \alpha : (P(\tilde{y} \leftarrow \tilde{z}) \parallel Q) \parallel \beta(R \parallel S) \qquad \text{(Reduction Com-4)}$$

$$\alpha : (\beta : (Chan\,?\,\tilde{y}\;;\;P) \parallel Q) \parallel \beta : (\alpha : ((Chan\,!\,\tilde{z}\;;\;R \parallel S))$$
$$\rightarrow \quad \alpha : (P(\tilde{y} \leftarrow \tilde{z}) \parallel Q) \parallel \beta(R \parallel S) \qquad \text{(Reduction Com-5)}$$

$$\beta : \{\texttt{to}(\alpha)\,.\,P \parallel Q\} \parallel (\alpha : \{R\})$$
$$\rightarrow \quad \alpha : \{\beta : \{P \parallel Q\}\;R\} \qquad \text{(Reduction Mob)}$$

STRL


csflow: tool1
File
Edit
Check
Run
New
Open
Reload
Reset
Save Output
Clear Output
Clear Buffer
Exit
Help
About
tool1 - WordPad
Home
View
Paste
Courier New
11
Find
Replace
Select all
Insert
Clipboard
Font
Paragraph
Editing
buf[
@recv(x).send(x).0 |
!recv(y).@send(y).@recv(z).send(z).0
]