Integrated Formal Modeling Techniques and UML

Dr Jin Song Dong

Computer Science Department

National University of Singapore

May 2001

`http://www.comp.nus.edu.sg/~dongjs`

The 23rd IEEE International Conference on Software Engineering (ICSE 2001, Toronto, Canada) Tutorial Notes (Thanks Drs. Roger Duke and Brendan Mahony for the joint research work and some teaching slides)

Overview

- State-based Formalism: Z/Object-Z
- Event-based Formalism: CSP/Timed-CSP
- Timed Communicating Object Z – TCOZ
- Active Objects and Network Topology
- Case Study: Lift System
- Sensor, Actuator and Control Systems
- Unified Modeling Language (UML)
- Linking TCOZ with UML
- Z family on the Web with their UML pictures

Why Formal Specification?

A formal specification should

- add clarity and understanding by giving a description of the system which is
  - complete
  - unambiguous
  - easily analysed;
- lead to better code that is
  - reliable
  - accurate
  - maintainable
  - reusable
  - verified.

Formal Specification and Software Engineering

(Towards an integrated methodology for software engineering.)

Formal specification

- is not a replacement, but rather an enhancement of existing methodologies;
- can only be effective if integrated within an overall methodology for soft ware engineering.

Implications of using formal specification

- training in the use of notation
- integration with informal methodologies
- translation for client consumption
- emphasis upon abstraction

Some Specification Languages

Process Algebra

Systems modelled as processes partaking in communication:

CSP, CCS, LOTOS

State Oriented

Systems modelled by an underlying state which can undergo change:

VDM, Z, Object-Z

Algebraic

Systems modelled by equations related by axioms (re-writing rules):

ACT 1, CLEAR, OBJ, Larch

The Z Specification Language

- developed originally at Programming Research Group, Oxford University
- based on set theory and predicate logic
- system described by introducing fixed sets and variables and specifying the relationships between them using predicates
- declarative, not procedural
- system state determined by values taken by variables subject to restrictions imposed by state invariant
- operations expressed by relationship between values of variables before, and values after, the operation
- variable declarations and related predicates encapsulated into schemas
- schema calculus facilitates the composition of complex specifications
- J. Woodcock and J. Davies , Using Z: Specification, Refinement, and Proof. Prentice-Hall, 1996

Types

Z is strongly typed: every expression is given a type.

Any set can be used as a type.

The following are equivalent within set comprehension

$$(x, y) : A \times B$$
$$x : A;\ y : B$$
$$x, y : A \quad \text{(when } B = A\text{)}$$

Notice that

$$\forall S : \mathbb{P} A \bullet \ldots \quad \text{not} \quad \forall S \subseteq A \bullet \ldots$$

In order to support the use of standard units of measurement, Hayes and Mahony has extended the Z typing system with standard units of measurement, e.g. time quantities are represented by the type

$$\mathbb{T} == \mathbb{R}\,\mathsf{s},$$

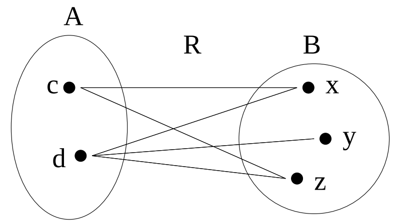
which represents real-valued time measured in seconds.

Relations

A relation R from A to B, denoted by

$$R : A \leftrightarrow B,$$

is a subset of $A \times B$.



R is the set $\{(c,x),(c,z),(d,x),(d,y),(d,z)\}$

Notation: the predicates

$$(c,z) \in R \quad \text{and} \quad c \mapsto z \in R \quad \text{and} \quad c \; \underline{R} \; z$$

are equivalent.

$$\begin{array}{lll} \mathrm{dom}\, R & \text{is the set} & \{a : A \mid \exists\, b : B \bullet a \; \underline{R} \; b\} \\ \mathrm{ran}\, R & \text{is the set} & \{b : B \mid \exists\, a : A \bullet a \; \underline{R} \; b\} \end{array}$$

Examples

$$\begin{array}{|l} \_ \leqslant \_ : \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline \forall\, x, y : \mathbb{N} \bullet \\ \qquad x \leqslant y \;\Leftrightarrow\; \exists\, k : \mathbb{N} \bullet x + k = y \end{array}$$

i.e. the relation $\leqslant$ is the infinite subset

$$\{(0,0),(0,1),(1,1),(0,2),(1,2),(2,2),\ldots\}$$

of ordered pairs in $\mathbb{N} \times \mathbb{N}$.

$$\begin{array}{|l} \mathit{divides} : \mathbb{N}_1 \leftrightarrow \mathbb{N} \\ \hline \forall\, x : \mathbb{N}_1;\; y : \mathbb{N} \bullet \\ \qquad x \; \underline{\mathit{divides}} \; y \;\Leftrightarrow\; \exists\, k : \mathbb{N} \bullet x\, k = y \end{array}$$

$3 \; \underline{\mathit{divides}} \; 6 \quad \text{but} \quad \neg\,(3 \; \underline{\mathit{divides}} \; 7)$

Domain and Range Restriction/Subtraction

Suppose $R : A \leftrightarrow B$ and $S \subseteq A$ and $T \subseteq B$; then

$$\begin{array}{lll} S \lhd R & \text{is the set} & \{(a,b) : R \mid a \in S\} \\ R \rhd T & \text{is the set} & \{(a,b) : R \mid b \in T\} \\ \\ S \ntriangleleft R & \text{is the set} & \{(a,b) : R \mid a \notin S\} \\ R \ntriangleright T & \text{is the set} & \{(a,b) : R \mid b \notin T\} \end{array}$$

e.g. if

$has\_sibling : People \leftrightarrow People$ then

$$\begin{array}{lll} female \lhd has\_sibling & \text{is the relation} & is\_sister\_of \\ has\_sibling \rhd female & \text{is the relation} & has\_sister \\ \\ female \ntriangleleft has\_sibling & \text{is the relation} & is\_brother\_of \\ has\_sibling \ntriangleright female & \text{is the relation} & has\_brother \end{array}$$

Functions

A (partial) function f from a set A to a set B, denoted by

$$f : A \rightarrow\!\!\!\!\!+ B,$$

is a subset f of $A \times B$ with the property that for each $a \in A$ there is at most one $b \in B$ with $(a,b) \in f$. The function f is a *total* function, denoted

$$f : A \rightarrow B,$$

if and only if $\mathrm{dom}\, f$ is the set A.

The predicates

$$(a,b) \in f \quad \text{and} \quad f(a) = b$$

are equivalent.

Examples:

$$
\begin{array}{|l}
root : \mathbb{N} \nrightarrow \mathbb{N} \\
\hline
\mathrm{dom}\, root = \{n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet m^2 = n\} \\
\forall\, n : \mathrm{dom}\, root \bullet (root(n))^2 = n
\end{array}
$$

$$
\begin{array}{|l}
+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\
\hline
\forall (n, m) : \mathbb{N} \times \mathbb{N} \bullet +(n, m) = n + m
\end{array}
$$

Generic Definitions

$$
\begin{array}{|l|}
\hline
[X, Y] \\
first : X \times Y \rightarrow X \\
\hline
\forall\, x : X;\ y : Y \bullet first(x, y) = x \\
\hline
\end{array}
$$

Exercise: Generic Purge Function

A generic purge function takes a time period (a timeout of type $\mathbb{T}$) and a set of time stamped elements of generic type X, and returns a set of updated time stamped elements. For example

$$ps(2\,\mathsf{s}, \{(1\,\mathsf{s}, a), (3\,\mathsf{s}, b), (7\,\mathsf{s}, c)\}) = \{(1\,\mathsf{s}, b), (5\,\mathsf{s}, c)\}$$

.

Solution

$$
\begin{array}{|l}
\hline\hline
[X] \\
ps : (\mathbb{T} \times \mathbb{F}(\mathbb{T} \times X)) \to \mathbb{F}(\mathbb{T} \times X) \\
\hline
\forall\, t : \mathbb{T};\ s : \mathbb{F}(\mathbb{T} \times X) \bullet \\
\qquad ps(t, s) = \{(t_1, e) : \mathbb{T} \times X \mid \exists (t_o, e) : s \bullet t_o - t = t_1\} \\
\hline
\end{array}
$$

or

$$
\begin{array}{|l}
\hline\hline
[X] \\
\ldots \\
\qquad ps(t, s) = \{(t_o, e) : s \mid t_o > t \bullet (t_o - t, e)\} \\
\hline
\end{array}
$$

Any difference between the two?

Sequences

A sequence s of elements from a set A, denoted

$s : \mathrm{seq}\, A$,

is a function $s : \mathbb{N} \rightarrowtail\!\!\!\!\!\!+ A$ where $\mathrm{dom}\, s = 1 \mathrel{..} n$ for some natural number n. For example,

$\langle b, a, c, b \rangle$ denotes the sequence (function)$\{1 \mapsto b, 2 \mapsto a, 3 \mapsto c, 4 \mapsto b\}$

The empty sequence is denoted by $\langle\ \rangle$.

The set of all sequences of elements from A is denoted $\mathrm{seq}\, A$ and is defined to be

$\mathrm{seq}\, A == \{s : \mathbb{N} \rightarrowtail\!\!\!\!\!\!+ A \mid \exists\, n : \mathbb{N} \bullet \mathrm{dom}\, s = 1 \mathrel{..} n\}$

We define $\mathrm{seq}_1\, A$ to be the set of all non-empty sequences, i.e.

$\mathrm{seq}_1\, A == \mathrm{seq}\, A - \{\langle\ \rangle\}$

Notice that: $\langle a, b, a \rangle \neq \langle a, a, b \rangle \neq \langle a, b \rangle$

Special Functions for Sequences

Concatenation

$\langle a, b\rangle \frown \langle b, a, c\rangle = \langle a, b, b, a, c\rangle$

Head, Last

$$\begin{array}{|l}
head, last : \mathrm{seq}_1\ A \rightarrow A \\
\hline
\forall\, s : \mathrm{seq}_1\ A \bullet head(s) = s(1) \wedge last(s) = s(\#s)
\end{array}$$

$head\langle c, b, b\rangle = c \qquad last\langle c, b, b\rangle = b$

Tail

$$\begin{array}{|l}
tail : \mathrm{seq}_1\ A \rightarrow \mathrm{seq}\ A \\
\hline
\forall\, s : \mathrm{seq}_1\ A \bullet \langle head(s)\rangle \frown tail(s) = s
\end{array}$$

$tail\langle c, b, b\rangle = \langle b, b\rangle$

Combining Formal Specification with Object-Oriented Design

goes against the conventional view of separating the concerns of functionality and design, but

- adds clarity and leads to simplification of large systems;
- helps with system abstraction and suggests a refinement into object-oriented code.

Object-Z is an extension to Z developed at the University of Queensland. It supports the design of object-oriented design.

- R. Duke and G. Rose, Formal Object Oriented Specification Using Object-Z. Cornerstones of Computing Series (editors: R. Bird, C.A.R. Hoare), Macmillan Press, March 2000.
- G. Smith, The Object-Z Specification Language, Kluwer Academic Publishers, 2000.

Object-Z Basics: Buffer Example

Buffer$[X]$

$max : \mathbb{N}$
$items : \mathrm{seq}\, X$
Δ
$size : \mathbb{N}$
$\#items = size \wedge size \leq max$

INIT

$items = \langle\, \rangle$

Join

$\Delta(items)$
$i? : X$

$items' = \langle i? \rangle \frown items$

Leave

$\Delta(items)$
$i! : X$

$size \neq 0 \wedge items = items' \frown \langle i! \rangle$

Exercise: For this *Buffer* class specify:

(a) an operation *count* which, given a message, outputs the number of times that message occurs in the buffer;

(b) an operation *duplicate* which appends to the buffer the message currently at the head of the buffer, provided the buffer is not empty or already full;

(c) an operation *titanic* whereby a sequence of messages is appended to the buffer except those messages for which there is no room are discarded (the buffer is like a life-boat on the Titanic: people queue to get on, but once the boat is full all the remaining people are left behind);

(d) an operation *penguin* whereby, like the operation *titanic*, a sequence of messages is input to the buffer, but this time the messages on the end of the sequence are accepted while those at the front are discarded if there is no room (the messages are acting like penguins, pushing out the messages already in the buffer once the buffer is full).

Solution

count

$m? : X$
$count! : \mathbb{N}$

$count! = \#(items \rhd \{m?\})$

duplicate

$\Delta(items)$

$\#items \in 1 \mathrel{..} (max - 1)$
$items' = items \frown \langle \mathrm{head}\ items \rangle$

titanic

$\Delta(items)$
$s? : \mathrm{seq}\ X$

$items' = (1 \mathrel{..} max) \lhd (items \frown s?)$

penguin

$\Delta(items)$
$s? : \mathrm{seq}\ X$

$\exists\, s : \mathrm{seq}\ X \bullet$
$\quad s \frown items' = items \frown s?$
$\quad s \neq \langle\ \rangle \Rightarrow \#items' = max$

Two Linked Buffers (single thread)

TwoBuffers[X]

$b_1, b_2 : Buffer[X]$

$b_1 \neq b_2$

INIT

$b_1.\mathrm{INIT} \wedge b_2.\mathrm{INIT}$

$Join \mathrel{\widehat=} b_1.Join$
$Leave \mathrel{\widehat=} b_2.Leave$
$Transfer \mathrel{\widehat=} b_1.Leave \parallel b_2.Join$

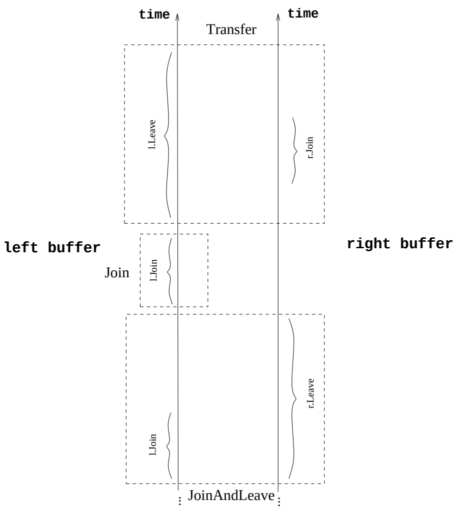


Figure 1: passive events

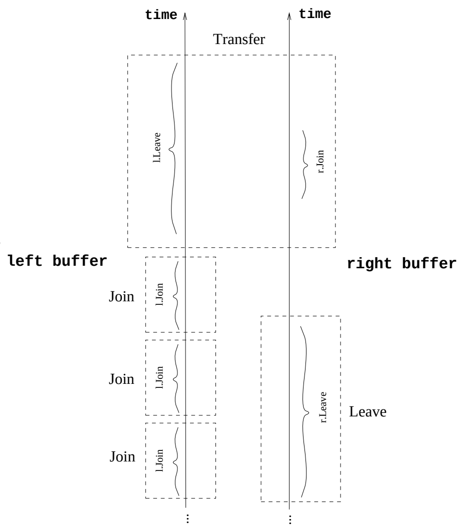


Figure 2: active events

CSP/Timed CSP

- Hoare's CSP (Communicating Sequential Processes) an *event* based notation primarily aimed at describing the sequencing of behaviour within a process and the synchronisation of behaviour (or *communication*) between processes.
- Timed CSP extends CSP by introducing a capability to quantify temporal aspects of sequencing and synchronisation.

- S. Schneider. Concurrent and Real-time Systems: The CSP Approach, Wiley, 1999.
- A.W. Roscoe. The Theory and Practice of Concurrency. Prentice-Hall, 1997.
- J. Davies, Specification and Proof in Real-Time CSP, Cambridge University Press, 1993.
- C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall International, 1985.

Events

A process engages in *events*; each event is an atomic action. e.g. the events for a vending machine are

coin—insert a coin

choc—extract a chocolate

The set of events that a process can possibly engage in is the *alphabet* of the process e.g. the alphabet of the vending machine is

$$\{coin, choc\}$$

Traces

A *trace* is a finite sequence of events.
A (deterministic) process is specified by the set of traces denoting its possible behaviour. e.g. the traces of the vending machine:

$$\langle\ \rangle$$
$$\langle coin \rangle$$
$$\langle coin, choc \rangle$$
$$\langle coin, choc, coin \rangle$$
$$\ldots$$

Any execution of the process will be one of these sequences. If $s \frown t$ is a trace of a process,
then so also is s;
i.e. the set of traces is prefix closed.

Examples

(1) $STOP_A$ is the process with alphabet A that can do nothing.

$$traces(STOP_A) = \{\langle\ \rangle\}$$

(2) $CLOCK$ is the process with $\alpha CLOCK = \{tick\}$ which can 'tick' at any time.

$$traces(CLOCK) = tick^*$$

(3) VM is the process with $\alpha VM = \{coin, choc\}$ which repeatedly supplies a chocolate after a coin is inserted.

$$traces(VM) = \{s : \mathrm{seq}\{coin, choc\} \mid \exists\, n : \mathbb{N} \bullet s \leqslant \langle coin, choc\rangle^n\}$$

(4) $WALK$ is a one-dimensional random walk process with $\alpha WALK = \{left, right\}$.

$$traces(WALK) = (left \cup right)^*$$

Prefix

A process which may participate in event a then act according to process description P is written

$$a@t \to P(t).$$

The event a is initially enabled by the process and occurs as soon as it is requested by its environment, all other events are refused initially. The event a is sometimes referred to as the *guard* of the process. The (optional) timing parameter t records the time, relative to the start of the process, at which the event a occurs and allows the subsequent behaviour P to depend on its value. For examples:

$$VMU = coin \to STOP$$

$$SHORTLIFE = (beat \to (beat \to STOP)) = beat \to beat \to STOP$$

$$VMS = coin \to choc \to STOP$$

Understanding Timed Prefix

Let P be a process which has two free time variables t_1 and t_2. A possible execution of the prefix:

$a\,@t_1 \rightarrow b\,@t_2 \rightarrow P$

$\downarrow$ 3 (time passed)

$a\,@t_1 \rightarrow b\,@t_2 \rightarrow P[(t_1+3)/t_1]$

$\downarrow a$ (event occur)

$b\,@t_2 \rightarrow P[3/t_1]$

$\downarrow$ 4 (time passed)

$b\,@t_2 \rightarrow P[3/t_1][(t_2+4)/t_2]$

$\downarrow b$ (event occur)

$P[3/t_1][4/t_2]$

Other CSP/Timed-CSP primitives:

- $P;\ Q$ (sequential composition)
- $P \,|[\, X \,]|\, Q$ (synchronous), $P \,|||\, Q$ (asynchronous)
- $a \rightarrow P \,\Box\, b \rightarrow Q$ (external choice), $a \rightarrow P \sqcap b \rightarrow Q$ (internal choice)
- $P_1 \triangledown e \rightarrow P_2$ (interrupt process)
- $\textsc{Wait}\ t;\ P$ (delay), $a \rightarrow P \triangleright \{t\}\ Q$ (time-out)

Sequential Composition

- The second form of sequencing is process sequencing. A distinguished event $\checkmark$ is used to represent and detect process termination.
- The sequential composition of P and Q, written $P;\ Q$, acts as P until P terminates by communicating $\checkmark$ and then proceeds to act as Q.
- The termination signal is hidden from the process environment and therefore occurs as soon as enabled by P. The process which may only terminate is written SKIP.

Parallel composition

The parallel composition of processes P and Q, synchronised on event set X, is written

$$P \,|[\, X \,]|\, Q.$$

No event from X may occur in $P \,|[\, X \,]|\, Q$ unless enabled jointly by both P and Q. When events from X do occur, they occur in both P and Q simultaneously and are referred to as *synchronisations*. Events not from X may occur in either P or Q separately but not jointly. For example, in the process described by

$$(a \to P) \,|[\, a \,]|\, (c \to a \to Q)$$

all a events must be synchronisations between the two processes.

In an asynchronous parallel combination

$$P \,|||\, Q$$

both components P and Q execute concurrently without any synchronisations.

Choice

Diversity of behaviour is introduced through two choice operators.

The external choice operator allows a process a choice of behaviour according to what events are requested by its environment. The process

$$(a \to P) \Box (b \to Q)$$

begins with both a and b enabled. The environment chooses which event actually occurs by requested one or the other first. Subsequent behaviour is determined by the event which actually occurred, P after a and Q after b respectively.

Internal choice represents variation in behaviour determined by the internal state of the process. The process

$$a \to P \sqcap b \to Q$$

may initially enable either a, or b, or both, as it wishes, but must act subsequently according to which event actually occurred. The environment cannot affect internal choice.

Channel

A channel is a collection of events of the form $c.n$: the prefix c is called the *channel name* and the collection of suffixes is called the *values* of the channel.

When an event $c.n$ occurs it is said that *the value n is communicated on channel c*. When the value of a communication on a channel is determined by the environment (external choice) it is called an *input* and when it is determined by the internal state of the process (internal choice) it is called an *output*.

It is convenient to write $c?n : N \to P(n)$ to describe behaviour over a range of allowed inputs instead of the longer $\Box\, n : N \bullet c.n \to P(n)$. Similarly the notation $c!n : N \to P(n)$ is used instead of $\bigsqcap n : N \bullet c.n \to P(n)$ to represent a range of outputs.

e.g.

$$COPYBIT = in.0 \to out.0 \to COPYBIT \;\; \Box \;\; in.1 \to out.1 \to COPYBIT$$

$$\alpha COPYBIT = \{in.0, out.0, in.1, out.1\}$$

Interrupt

The interrupt process $P_1 \triangledown e \rightarrow P_2$ behaves as P_1 until the first occurrence of interrupt event e, then the control passes to P_2.

Recursion

Recursion is used to given finite representations of non-terminating processes. The process expression

$$\mu P \bullet a?n : \mathbb{N} \rightarrow b!f(n) \rightarrow P$$

describes a process which repeatedly inputs a natural on channel a, calculates some function f of the input, and then outputs the result on channel b.

Timeout

The timeout construct passes control to an exception handler if no event has occurred in the primary process by some deadline.

The process

$$(a \rightarrow P) \triangleright\{t\}\ Q$$

will try to perform $a \rightarrow P$, but will pass control to Q if the a event has not occurred by time t, as measured from the invocation of the process. For example,

$$MayPrint1 = (receive \rightarrow print \rightarrow STOP) \triangleright\{60\}\ shutdown \rightarrow STOP$$

$$MP1(t) = (receive \rightarrow print \rightarrow STOP) \triangleright\{60 - t\}\ shutdown \rightarrow STOP$$

Exercise: Transmitter

A transmitter which repeatedly send a given message x until it receives and acknowledgement. Assume that the transmitter is in an environment which is always ready to accept a *send* message, then it will send the message every 5 time units until an *ack* message is received. (hint using recursion together with timeout).

Solution

$$Transmit(x) = send!x \rightarrow ((ack \rightarrow STOP) \triangleright\{5\}\ Transmit(x))$$

Delay

A process which allows no communications for period t then terminates is written $\textsc{Wait}\, t$. The process

$$\textsc{Wait}\, t;\ P \ = \ STOP \rhd\{t\}\ P$$
$$a \xrightarrow{t} P \ = \ a \to \textsc{Wait}\, t;\ P \ = \ a \to (STOP \rhd\{t\}\ P)$$

is used to represent P delayed by time t.

State parameters

In general, the behaviour of a process at any point in time may be dependent on its internal state and this may conceivably take an infinite range of values.

It is often not possible to provide a finite representation of a process without introducing some notation for representing this internal process state.

The approach adopted by CSP is to allow a process definition to be parameterised by state variables. Thus a definition of the form

$$P_{n:N} \mathrel{\widehat{=}} Q(n)$$

represents a (possibly infinite) family of definitions, one for each possible value of n.

There is no inherent notion of process state in CSP, but rather these annotations are a convenient way to provide a finite representation of an infinite family of process descriptions.

Exercise: A generic timed-collection

The generic timed-collection denotes a collection of elements of type X with a time stamp. Operations are allowed to add elements to and delete elements from the collection. When deleting an element from the collection, the oldest element should be removed and output to the environment. The collection has the following timing properties. Firstly, that it updates the internal state during a *add* or *delete* operation. Secondly, each element of the collection becomes *stale* if it is not passed on within t_o time units of being added to the collection. Stale elements should never be passed on, but are instead purged from the collection upon becoming stale.

Solution

$$TimedCollection \mathrel{\widehat=} TC_{\varnothing}.$$

$$TC_{\varnothing} \mathrel{\widehat=} left?e : X \to TC_{\{(t_o, e)\}}$$

$$\begin{array}{l} TC_{\{(t,a)\} \cup s} \mathrel{\widehat=} \\ \quad (left?e : X \mathbin{@} t_i \to TC_{ps(t_i, \{(t,a)\} \cup s) \cup \{(t_o, e)\}} \;\Box \\ \quad right!a \mathbin{@} t_i \to TC_{ps(t_i, s)}) \rhd \{t\}\; TC_{ps(t,s)} \end{array}$$

where $(t, a) = find\_oldest(\{(t, a)\} \cup s)$.

$$\begin{array}{|l} [X] \\ \hline find\_oldest : \mathbb{P}_1(\mathbb{T} \times X) \to (\mathbb{T} \times X) \\ \hline \forall s : \mathbb{P}_1(\mathbb{T} \times X) \bullet \\ \quad \exists (t, e) : s \bullet t = min(\mathrm{dom}\, s) \\ \quad find\_oldest(s) = (t, e) \end{array}$$

Object-Z and Timed CSP

- Object-Z
  - ✓ an excellent tool for modeling data states
  - × but difficult for modelling real-time concurrent systems
- Timed CSP
  - ✓ Good for specifying the timed process and communication
  - × Like CSP, cumbersome to capture the data state of a complex system
- Timed Communicating Object Z: a blending of Object-Z and Timed CSP

Related Work

* Z/OZ with CSP: Fischer, Smith, Derick, Suhl, Bolton, Davies, Woodcock ...
* Z with CCS: Galloway, Stoddart, Taguchi, Araki ...

Timed Communicating Object Z (TCOZ)

$TimedBuffer[X]$

$items : \mathrm{seq}\ X$
$left, right : \mathbf{chan}$

$\textsc{Init}$
$items = \langle\ \rangle$

Add
$\Delta(items)$
$i? : X$

$items' = \langle i? \rangle ^\frown items$

$Remove$
$\Delta(items)$

$items' = items \rhd \{last(items)\}$

$Join \mathrel{\widehat=} [i : X] \bullet left?i \to Add;\ \textsc{Deadline}\ tj$

$Leave \mathrel{\widehat=} [items \neq \langle\ \rangle] \bullet right!last(items) \to Remove;\ \textsc{Deadline}\ tl$

$\textsc{Main} \mathrel{\widehat=} \mu\, Q \bullet (Join \,\Box\, Leave);\ Q$

Abstract syntax

$[ZS, ZE]$ [Z schemas and expressions]

$TZE ::=$ [TCOZ constructor expressions]

$ref \langle\langle NAME \rangle\rangle \mid \textsc{Stop} \mid \textsc{Skip} \mid \textsc{Wait}\langle\langle ZE \rangle\rangle \mid (\_ \bullet \_)\langle\langle ZS \times TZE \rangle\rangle \mid$

$(\_ \rightarrow \_)\langle\langle \Sigma \times TZE \rangle\rangle \mid (\_ . \_ \rightarrow \_)\langle\langle \Sigma \times ZE \times TZE \rangle\rangle \mid$

$(\_ \Box \_)\langle\langle TZE \times TZE \rangle\rangle \mid \ldots \mid (\mu \_ \bullet \_)\langle\langle NAME \times TZE \rangle\rangle$

$$TZB == \{\mathcal{C} : NAME \twoheadrightarrow TZE \mid \forall tze : \operatorname{ran} \mathcal{C};\ N : NAME \bullet N \in \operatorname{sig} tze \Rightarrow N \in \operatorname{dom} \mathcal{C}\}$$ [TCOZ class body]

$TZC_p == [init : ZS;\ \mathcal{C} : TZB]$ [passive classes]

$TZC_a ==$ [active classes]
$[init : ZS;\ \mathcal{C} : TZB;\ main : TZE \mid \operatorname{sig} main \subseteq \operatorname{dom} TZB]$

TCOZ Semantics

The support of timing primitives in TCOZ is made possible through the adoption of Reed's timed-failures semantics for Timed CSP. The timed-failures semantics models CSP processes in terms of timed event-traces and timed event-failures. This semantic model allows CSP to be extended with time related primitives such as delays, timeouts, and clock-interrupts. In order to support objects with encapsulated state this model is extended to include an initial state and state update events. Object-Z operations are modelled as terminating sequences of timed state-update events.

- B. Mahony and J.S. Dong. Overview of the Semantics of TCOZ. *Integrated Formal Methods (IFM'99)*, pages 66-85, Springer-Verlag, York, UK, June 1999.

Exercise: A generic timed-collection in TCOZ

The generic timed-collection denotes a collection of elements of type X with a time stamp. Operations are allowed to add elements to and delete elements from the collection. When deleting an element from the collection, the oldest element should be removed and output to the environment. The collection has the following timing properties. ...

Solution

$$
\begin{array}{l}
\textbf{Class } \mathit{TimedCollection}[X] \\
\quad \mathit{tc} : \mathbb{P}(\mathbb{T} \times X) \\
\quad \mathit{left}, \mathit{right} : \mathbf{chan} \\
\quad \Delta \\
\quad t : \mathbb{T};\ \mathit{oldest} : X \\
\quad \hline
\quad \mathit{tc} \neq \varnothing \Rightarrow (t, \mathit{oldest}) \in \mathit{tc} \land t = \mathit{min}\, \mathrm{dom}\, \mathit{tc}
\end{array}
$$

$$
\begin{array}{l}
\textsc{Init} \\
\hline
\mathit{tc} = \varnothing
\end{array}
$$

$$
\begin{array}{l}
\mathit{Add}_0 \\
\hline
\Delta(\mathit{tc}) \\
e? : X;\ t_i : \mathbb{T} \\
\hline
\mathit{tc}' = \mathit{ps}(t_i, \mathit{tc}) \cup \{(t_o, e?)\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Delete}_0 \\
\hline
\Delta(\mathit{tc}) \\
t_i : \mathbb{T} \\
\hline
\mathit{tc}' = \mathit{ps}(t_i, \mathit{tc} \setminus \{(t, \mathit{oldest})\})
\end{array}
$$

$$
\begin{array}{l}
\mathit{Add} \mathrel{\widehat=} [e : X;\ t_i : \mathbb{T}] \bullet \mathit{left}?e\,@t_i \rightarrow \mathit{Add}_0 \\
\mathit{Delete} \mathrel{\widehat=} [t_i : \mathbb{T}] \bullet \mathit{right}!\mathit{oldest}\,@t_i \rightarrow \mathit{Delete}_0 \\
\textsc{Main} \mathrel{\widehat=} \mu\, T \bullet [\mathit{tc} = \varnothing] \bullet \mathit{Add};\ T\ \Box \\
\qquad\qquad [\mathit{tc} \neq \varnothing] \bullet ((\mathit{Add}\ \Box\ \mathit{Delete}) \rhd \{t\}\ \mathit{tc} := \mathit{ps}(t, \mathit{tc}));\ T
\end{array}
$$

The Notion of Active Object

- Active objects have their own thread of control.
- Passive objects are controlled by other objects in a system.
- A class for defining active objects is called an *active class*
- A class for defining passive objects is called a *passive class*.
- In TCOZ, MAIN, a non-terminating process definition, distinguishes the active and the passive classes.

Inheritance between active/passive classes

- When a new active class is derived from an existing active class, the MAIN process must always be redefined explicitly.
- A new active class can be derived from an existing passive class, in this case, a MAIN process definition needs to be added.
- A new passive class can also be derived from an existing active class, in this case, the MAIN process of the existing class is implicitly removed.
- A new passive class can be derived from an existing passive class following the same rules as the standard Object-Z.

Composition and interaction of active objects

A

$v : T;\ \ldots$
$c : \textbf{chan};\ \ldots$

$OpA_1 \mathrel{\widehat=} \ldots$
$\ldots$
$\textsc{Main} \mathrel{\widehat=} \ldots$

B

$a : A$
$\ldots$

$\ldots$
$OpB_1 \mathrel{\widehat=} a.OpA_1$
$\ldots$

A

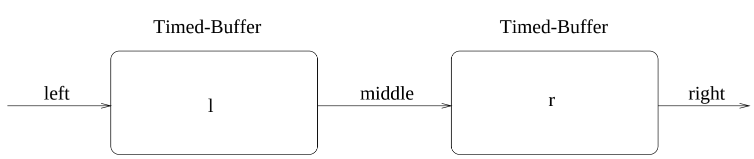
$\upharpoonright(v, \ldots, OpA_1, \ldots, OpA_n)$

$\ldots$

$OpA_1 \mathrel{\widehat=} \ldots$
$\ldots$
$\textsc{Main} \mathrel{\widehat=} \ldots$

Identifying the object name with its $\textsc{Main}$ process, e.g. if ob_1 and ob_2 are active object components, then $ob_1 \mathbin{|||} ob_2$ means $ob_1.\textsc{Main} \mathbin{|||} ob_2.\textsc{Main}$.

Two Communicating Timed Buffers

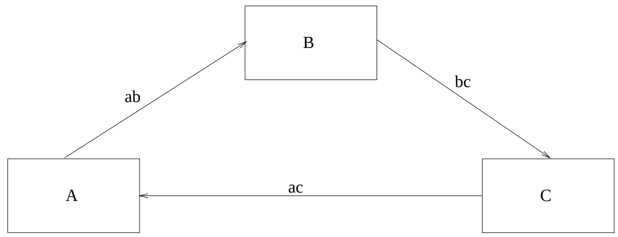


$TwoBuffers[X]$

$l : TimedBuffer[X][middle/right]$
$r : TimedBuffer[X][middle/left]$

$\textsc{Main} \mathrel{\widehat=} (l \mathbin{|[\, middle \,]|} r \setminus middle)$

Complex network topologies



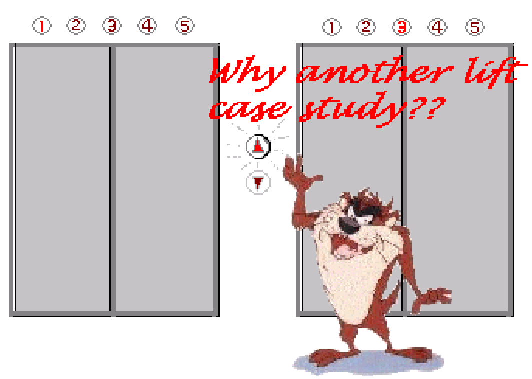
$$(A[bc'/bc] \,|[\, ab, ac \,]|\, (B[ac'/ac] \,|[\, bc \,]|\, C[ab'/ab]) \setminus ab, ac, bc)[ab, ac, bc/ab', ac', bc']$$

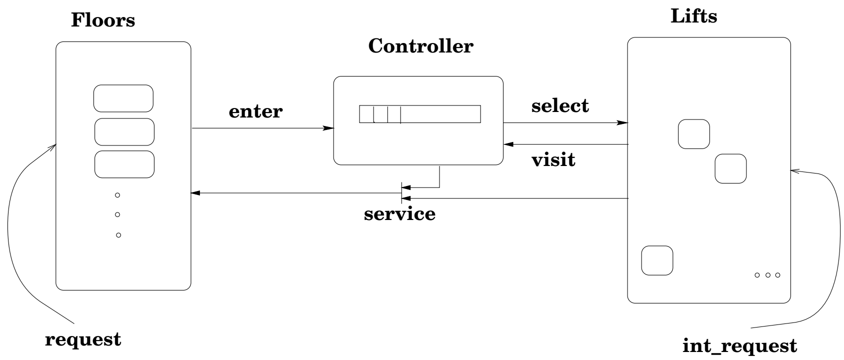
$$\boxed{(\| \, v_1, v_2, v_3 \ldots \bullet v_1 \overset{ch_{12}}{\longleftrightarrow} v_2, v_2 \overset{ch_{23}}{\longleftrightarrow} v_3, v_3 \overset{ch_{13}}{\longleftrightarrow} v_1, \ldots)}\,(A, B, C)$$

$$\boxed{\|(A \overset{ab}{\longleftrightarrow} B, B \overset{bc}{\longleftrightarrow} C, C \overset{ca}{\longleftrightarrow} A)} \quad or \quad \boxed{\|(A \overset{ab}{\leftrightarrow} B \overset{bc}{\leftrightarrow} C \overset{ca}{\leftrightarrow} A)}$$

The Lift Case Study

- Multi-floors with multi-elevators
- Non-trivial
- Commonly used example
- Both CSP and Object-Z have been applied (but no real-time issues)





Detailed model can be found at:

B. Mahony and J.S. Dong. Timed Communicating Object Z. IEEE Transactions on Software Engineering, 26(2):150-177, Feb 2000.

http://www.comp.nus.edu.sg/~dongjs/papers/tse00.ps

Button

$state : On \mid Off$

INIT: $state = Off$

TurnOn: $\Delta(state)$ — $state' = On$

TurnOff: $\Delta(state)$ — $state' = Off$

TopFloor

$downbutton : Button_{©}$
$request, enter, service :$ **chan**

INIT: $downbutton.\mathrm{INIT}$

$PressDown \mathrel{\widehat=} [downbutton.state = Off] \bullet$
$\quad (request?Down \to downbutton.TurnOn);\ (enter!Down \to Skip)$
$DownOff \mathrel{\widehat=} service?Down \to downbutton.TurnOff$
$\mathrm{MAIN} \mathrel{\widehat=} \mu\, T \bullet (PressDown \,\square\, DownOff);\ T$

BottomFloor

$upbutton : Button_{©}$

INIT: $upbutton.\textsc{Init}$

…

MiddleFloor

$TopFloor, BottomFloor$

$\textsc{Main} \mathrel{\widehat=} \mu\, M \bullet (PressDown \mathbin{\Box} DownOff \mathbin{\Box} PressUp \mathbin{\Box} UpOff);\ M$

$Floor \mathrel{\widehat=} TopFloor \cup BottomFloor \cup MiddleFloor$

Floors

$floors : \mathrm{seq}\ Floor_{©}$

…

$\textsc{Main} \mathrel{\widehat=} \ |||\ f : \mathrm{ran}\ floor$

Lift door control

Door

$open, conf, close, servo, sensor : \mathbf{chan}$

$OpenDoor, CloseDoor \mathrel{\widehat=} \ldots$

$CycleDoor \mathrel{\widehat=} OpenDoor;\ conf \rightarrow$

$(\mu\, CD \bullet \textsc{Wait}\ t_o;\ CloseDoor \triangledown \{sensor?(self, Interrupt)\}\ OpenDoor;\ conf \rightarrow CD)$

$\textsc{Main} \mathrel{\widehat=} \mu\, D \bullet open \rightarrow CycleDoor;\ close \rightarrow D$

Moving the lift

Shaft

$move, arrive : \mathbf{chan}$

$\textsc{Main} \mathrel{\widehat=} \mu\, S \bullet move?n \rightarrow \textsc{Wait}\ |n| * \bar{t} + delay;\ arrive \rightarrow S$

Internal_Q

$panel : \mathrm{seq}\ Button_{©}$
$int\_request, int\_sched, int\_serv : \mathbf{chan}$

$NextUp, NextDown, \textsc{Main} \mathrel{\widehat=} \ldots$

LiftControl

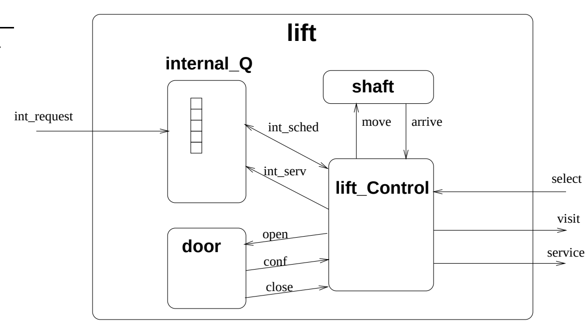
$fl : \mathbb{N}$
$md : MoveDirection$
$move, arrive : \mathbf{chan}$ [shaft]
$\ldots$

$\ldots$
$\textsc{Main} \mathrel{\widehat=} \mu\, LC \bullet$
$\quad \ldots \rightarrow Internal;\ LC\ \Box$
$\quad \ldots \rightarrow External;\ LC$

Lift

$iq : Internal\_Q_{©}$
$lc : LiftControl_{©}$
$s : Shaft_{©}$
$d : Door_{©}$

$\textsc{Main} \mathrel{\widehat=} \|($
$\quad lc \xleftrightarrow{move, arrive} s;$
$\quad lc \xleftrightarrow{open, close, conf} d;$
$\quad lc \xleftrightarrow{int\_sched, int\_serv} iq)$



Lifts

$lifts : \mathbb{P}\ Lift_{©}$

$\textsc{Main} \mathrel{\widehat=} \quad ||| \ l : lifts$

Controller

$requests : \mathrm{seq}(\mathbb{N} \times MoveDirection)$
$enter, select, visit, service : \mathbf{chan}$

Join
...

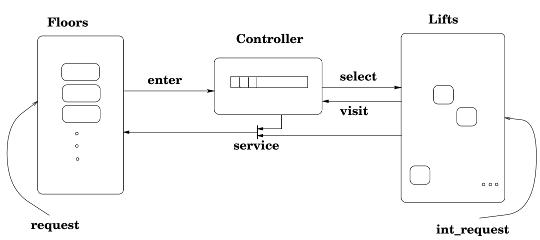
Remove
...

$Dispatch \mathrel{\widehat=} [...] \bullet select!item \rightarrow Remove$

$CheckServ \mathrel{\widehat=}$
$\quad [item : \mathbb{N} \times MoveDirection] \bullet visit?item \rightarrow ...$

$\textsc{Main} \mathrel{\widehat=} \mu\, C \bullet (Join \;\Box\; Dispatch \;\Box\; CheckServ);\;\; C$

The Lift System

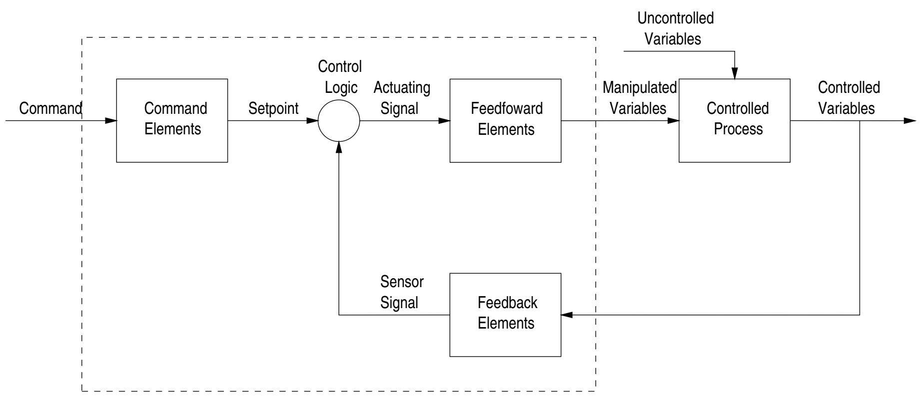


LiftSystem

$fs : Floors_{©}$
$ls : Lifts_{©}$
$contr : Controller_{©}$

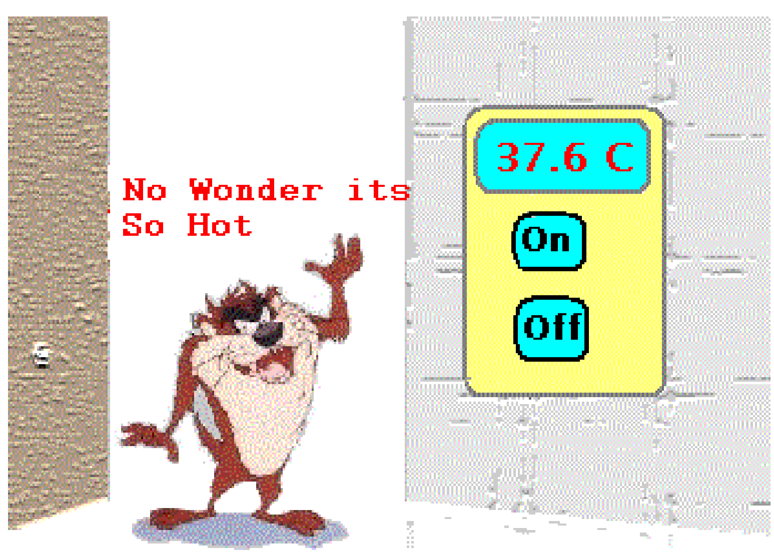
$\textsc{Main} \mathrel{\widehat=} \Vert (fs \xleftrightarrow{enter} contr \xleftrightarrow{select,\, check} ls \xleftrightarrow{service} fs)$

Sensors and Actuators — Control Systems



- CSP channel mechanism is discrete
- CSP channel mechanism is synchronous

Example: Digital Temperature Display



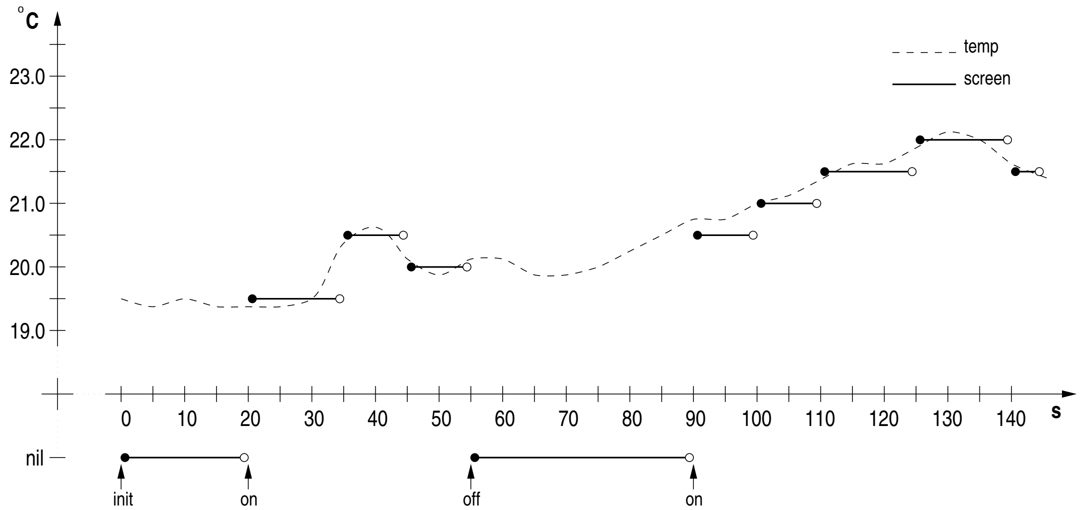


Figure 3: The office communication scenario.

$$temp : \mathbb{R}°\mathsf{C}\,\textbf{sensor}, \quad == \quad temp : \mathbb{R}\,\mathsf{s} \to \mathbb{R}°\mathsf{C}\,.$$

Internally, *temp* takes the syntactic role of a CSP channel. Whenever a value v is communicated on the internal channel at a time t, $temp(t) = v$.

$$screen : Display\,\textbf{actuator},$$

where

$$Display \ ::= \ Temp\langle\!\langle \mathbb{N} * 0.5°\mathsf{C} \rangle\!\rangle \mid nil.$$

The internal role is that of the local state variable.

DTD

temp : $\mathbb{R}$ **sensor**
screen : *Display* **actuator**
on, *off* : **chan**

INIT
screen = *nil*

SetScreen
$\Delta(screen)$
$t? : \mathbb{R}°\mathsf{C}$

$\exists\, dt : \mathbb{N} * 0.5°\mathsf{C} \bullet$
$\quad dt = t \pm 0.5°\mathsf{C} \wedge$
$\quad screen' = Temp(dt)$

$Show \mathrel{\widehat{=}} ([t : \mathbb{R}°\mathsf{C}] \bullet temp?t \rightarrow SetScreen;\ \textsc{Deadline}\, 5\,\mathsf{s};\ \textsc{WaitUntil}\, 5\,\mathsf{s};\ Show)$
$\quad \nabla\ off \rightarrow NoShow$

$NoShow \mathrel{\widehat{=}} screen := nil;\ on \rightarrow Show$

$\textsc{Main} \mathrel{\widehat{=}} on \rightarrow Show$

Asynchronous active object

Synchronous active objects

- have discrete interfaces, synchronous channels;
- are highly dependent.

Asynchronous active objects

- have analog interfaces, asynchronous sensor/actuators;
- are highly independent;
- can be further classified into *periodic* and non-periodic objects.

Exercise: a calendar clock

A typical periodic object: a calendar clock ticks every second ...

$$CalendarTime == \mathbb{N}\,\mathsf{yr} \times \mathbb{N}\,\mathsf{mn} \times \mathbb{N}\,\mathsf{dy} \times \mathbb{N}\,\mathsf{hr} \times \mathbb{N}\,\mathsf{min} \times \mathbb{N}\,\mathsf{s}\,.$$

$$Convert : \mathbb{N}\,\mathsf{s} \rightarrow CalendarTime$$

$$\ldots \qquad \qquad \text{[detail of function omitted]}$$

Solution

Clock

$per == 1\,\mathsf{s}$
$gain == 50\,\mathsf{ms}$

$total : \mathbb{N}\,\mathsf{s}$
Δ
$display : CalendarTime$ **actuator**

$display = Convert(total)$

Inc

$\Delta(total)$

$total' = total + 1$

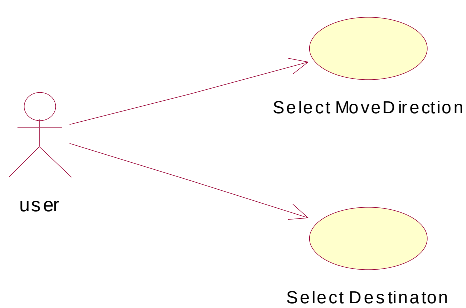
$\textsc{Main} \mathrel{\widehat=} \mu\, C \bullet Inc;\ \textsc{Deadline}\ gain;\ \textsc{WaitUntil}\ per;\ C$

UML

- UML stands for Unified (?) Modeling Language
- The UML combines/collects Data Modeling concepts (Entity Relationship Diagrams), Business Modeling (work flow), Object Modeling, and Component Modeling
- The UML is the OMG standard language for visualising, specifying, constructing, and documenting the artifacts of a software-intensive system
- UML consists of use case, class, statechart, collaboration diagrams ...

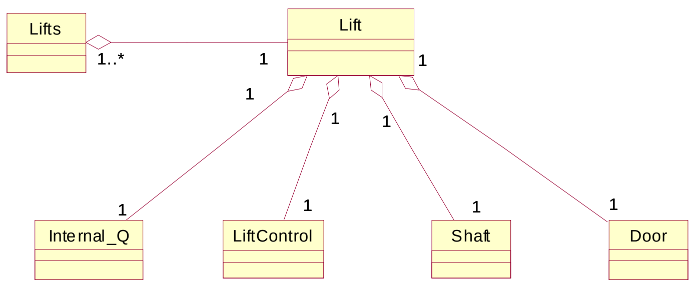
Use Case Diagram

- Use case is a pattern of behavior the system exhibits Each use case is a sequence of related transactions performed by an actor and the system in a dialogue. Actors are examined to determine their needs. Use case diagrams are created to visualise the relationships between actors and use cases



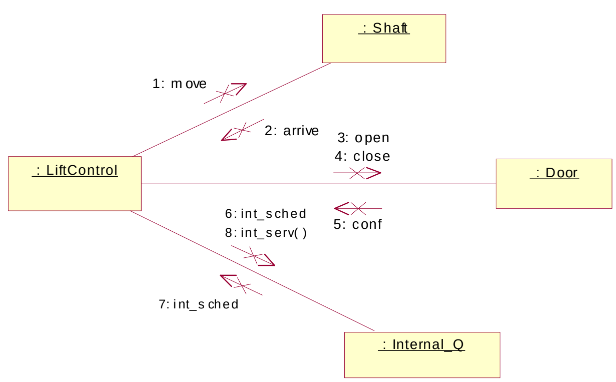
Class Diagram

- A class diagram shows the existence of classes and their relationships in the logical view of a system. It consists of classes and their structure and behavior, association, aggregation, dependency, and inheritance relationships, multiplicity and navigation indicators, and role names



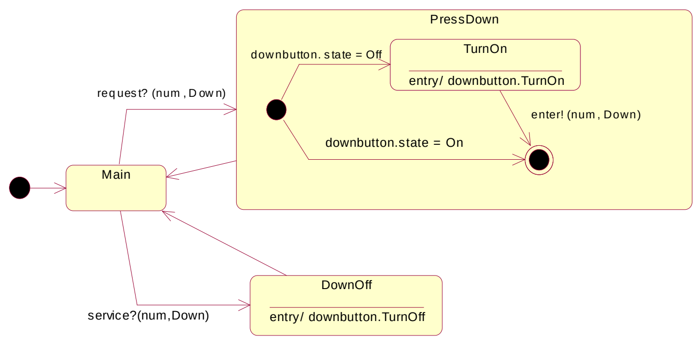
Collaboration Diagram – dynamic behavior, message-oriented

- A collaboration diagram displays object interactions organised around objects and their links to one another



Statechart Diagram – dynamic behavior, event-oriented

- A statechart diagram shows the life history of a given class, the events that cause a transition from one state to another, and the actions that result from a state change



Shortcomings of UML

- There is no unified formal semantics for all those diagrams. There are a few approaches to formalize a subset of UML, e.g. (Evans and Clark, 1998, Kim and Carrington, 1999) concentrated on class diagram semantics. Therefore, the consistency between diagrams is problematic; and

- There are limited capabilities for precisely modeling timed concurrency. For example, (in a new feature that has been added to the UML 1.3) synchronisation between concurrent substates of a single statechart diagram can be captured using a synch state link. However there is no facility to precisely model synchronous interactions between states in two different statechart diagrams.

Linking TCOZ and UML

- Syntactically, UML/OCL (Object Constraint Language) is extended with TCOZ communication interface types — chan, sensor and actuator. Upon that, TCOZ sub-expressions can be used (in the same role as OCL) in the statechart diagrams and collaboration diagrams.
- Semantically, UML class diagrams are identified with the signatures of the TCOZ classes. The states of the UML statechart are identified with the TCOZ processes (operations) and the state transition links are identified with TCOZ events/guards.
- Effectively, UML diagrams can be seen as the viewpoint visual projections from a formal TCOZ model.

Combination Process of TCOZ and UML

1. Firstly, the UML use-case models (user-case and collaboration diagrams) are used to analyse system requirements so that main classes and operations will be identified (e.g. classification of the boundary and control classes). Communication links of the collaboration diagrams guide the design of communication interfaces of the TCOZ model (synchronisation — channel, synchronisation — sensor/actuator).
2. Then, the UML class diagrams are used to capture the static structure of the system, in which class/object relationships can be captured.
3. Based on UML class diagrams, detailed TCOZ formal models are constructed in a bottom-up style. The states, timing and concurrent interactions of the system objects are captured precisely in the TCOZ models.
4. Finally, UML state diagrams are used to visualize the behaviors (process states and events) of essential components of the system, which are closely associated with the behavior parts of the TCOZ model.

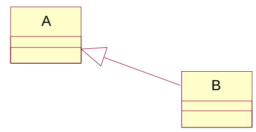
Class

B

A

......

......

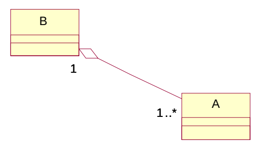


B

$a1, a2 : A$

......

......



Synchronisation

A

c : **chan**

......

$Main \mathrel{\widehat=} ...c!...$

B

c : **chan**

......

$Main \mathrel{\widehat=} ...c?...$

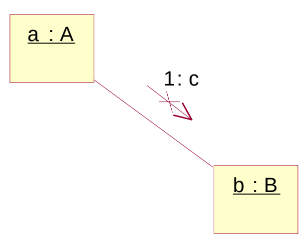
AB

$a : A$

$b : B$

......

$Main \mathrel{\widehat=} ...a \xleftrightarrow{c} b...$



Asynchronisation

A

$c : \mathbb{N}$ **actuator**

......

......

B

$c : \mathbb{N}$ **sensor**

......

......

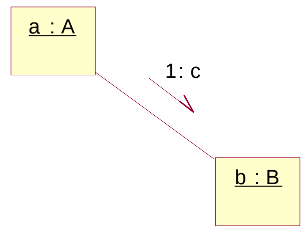
AB

$a : A$

$b : B$

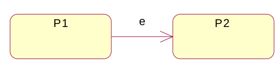
......

$Main \mathrel{\widehat=} \ldots a \overset{c}{\longleftrightarrow} b \ldots$

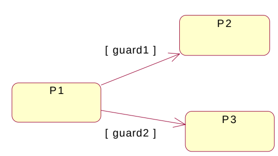


Dynamic Behavior

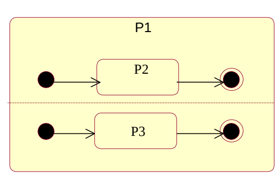
$P1;\ e \rightarrow P2$



$P1;\ ([guard1] \bullet P2 \;\square$
$[guard2] \bullet P3)$

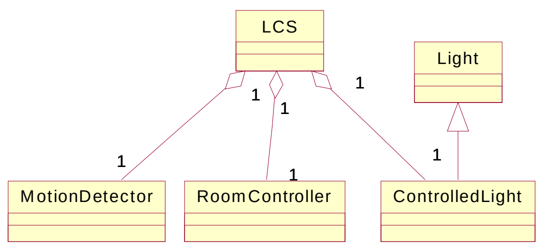


$P1 \mathrel{\widehat=} P2 \;|||\; P3$



Light Control System (LCS)

In most existing light control systems, all lights are controlled manually. Electrical energy is wasted by lighting rooms that are not occupied and by not adjusting light levels relative to need and daylight. LCS is an intelligent control system. It can detect the occupation of the building, then turn on or turn off the lights automatically. It is able to tune illumination in the building according to the outside light level. It gains input from sensors and actuators.



$Illumination == 1..10000 \text{ lux}$
$Percent == \{0\} \cup 10..100$

$MotionDetector$

$motion : \mathbf{chan}$
$md : (Move \mid NoMove)\ \mathbf{sensor}$ [motion sensor]

$NoUser \mathrel{\widehat=} md?Move \to motion!1 \to User\ \Box$
$\quad md?NoMove \to \textsc{Wait}\ 1\,\mathsf{s};\ NoUser$
$User \mathrel{\widehat=} md?NoMove \to motion!0 \to NoUser\ \Box$
$\quad md?Move \to \textsc{Wait}\ 1\,\mathsf{s};\ User$
$\textsc{Main} \mathrel{\widehat=} NoUser$

Light

$dim : Percent$ **actuator**	[dim value]
$on : \mathbb{B}$	

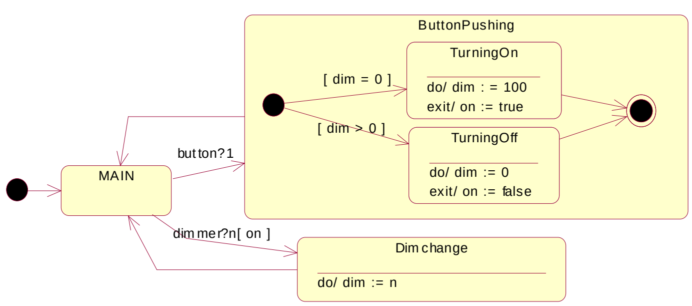
$TurningOn \mathrel{\widehat=} dim := 100;\ on := true$
$TurningOff \mathrel{\widehat=} dim := 0;\ on := false$

ControlledLight

$Light$

$button, dimmer :$ **chan**	[control channels]

$$ButtonPushing \mathrel{\widehat=} button?1 \rightarrow \\ \quad ([dim > 0] \bullet TurningOff \mathrel{\Box} [dim = 0] \bullet TurningOn)$$
$$DimChange \mathrel{\widehat=} [n : Percent] \bullet dimmer?n \rightarrow \\ \quad ([on] \bullet dim := n \mathrel{\Box} [\neg\ on] \bullet \textsc{Skip})$$
$$\textsc{Main} \mathrel{\widehat=} \mu\, N \bullet (ButtonPushing \mathrel{\Box} DimChange);\ N$$



$satisfy : Percent \leftrightarrow Illumination$

RoomController

$dimmer, motion : \textbf{chan}$
$odsensor : Illumination\ \textbf{sensor}$
$absenT : \mathbb{T}$
$olight : Illumination$

Adjust

$dim! : Percent\ \textbf{on}\ dimmer$

$dim!\ \underline{satisfy}\ olight$

$Ready \mathrel{\widehat{=}} motion?1 \rightarrow On$

$Regular \mathrel{\widehat{=}} \mu\, R \bullet [n : Illumination] \bullet$
$\quad odsensor?n \rightarrow olight := n;\ Adjust;\ dimmer!dim \rightarrow R$

$On \mathrel{\widehat{=}} Regular\ \nabla\ motion?0 \rightarrow OnAgain$

$OnAgain \mathrel{\widehat{=}} (motion?1 \rightarrow On) \triangleright\{absenT\}\ Off$

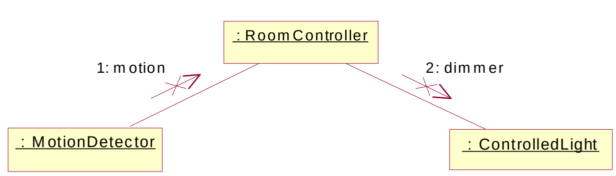
$Off \mathrel{\widehat{=}} dimmer!0 \rightarrow Ready$

$\textsc{Main} \mathrel{\widehat{=}} Off$

LCS

$m : MotionDetector$
$l : ControlledLight$
$r : RoomCtrller$

$\textsc{Main} \mathrel{\widehat{=}} \|(m \xleftrightarrow{motion} r \xleftrightarrow{dimmer} l)$



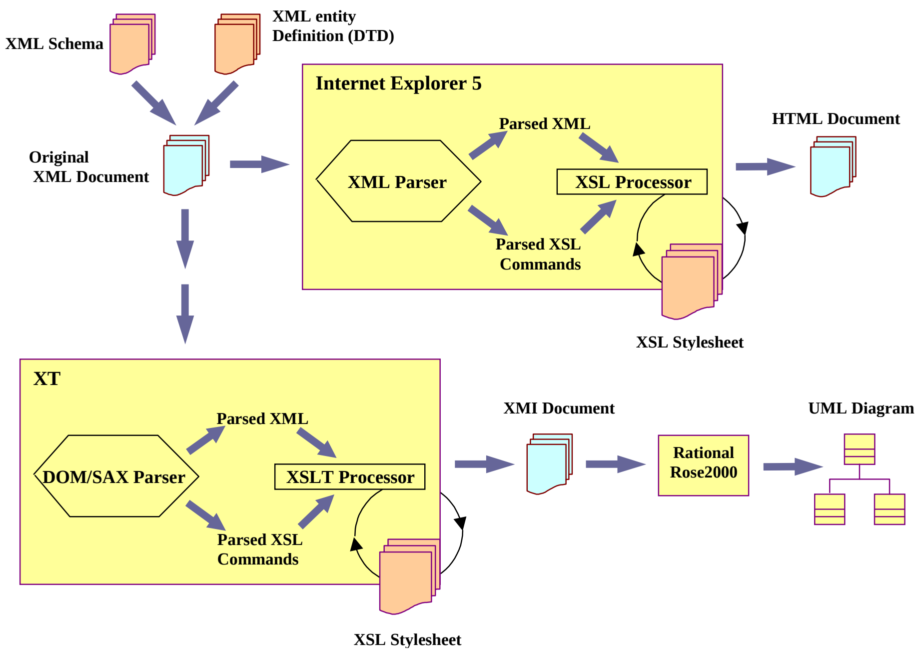
Z Family on the Web with their UML Photos

- Use eXtensible Markup Language (XML) to develop web environment for Z family languages

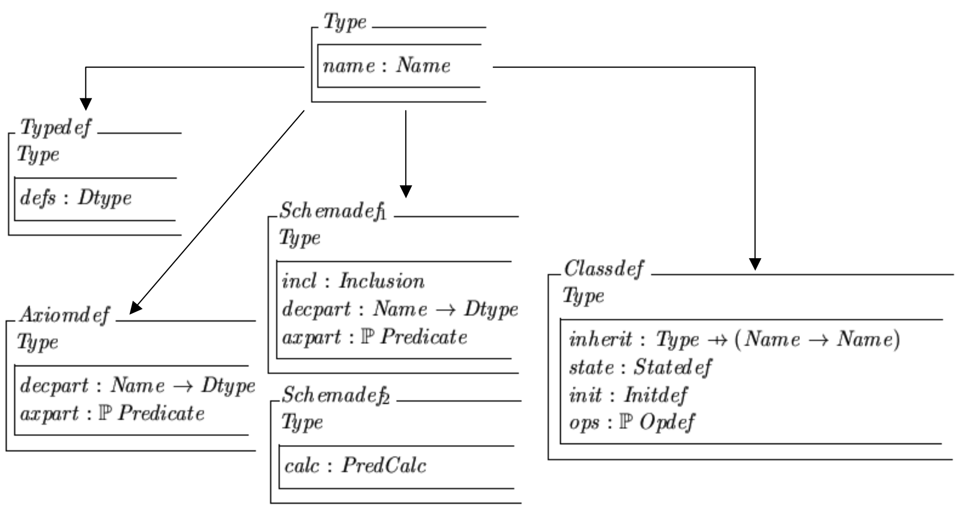
  - share design models
  - hyperlinks among models
  - advance browsing facilities

  `http://nt-appn.comp.nus.edu.sg/fm/zml/`

- Develop techniques for projecting (object-oriented) Z models to UML diagrams, based on XML Metadata Interchange (XMI).
- Use Object-Z to specify and design the essential functionalities of the ZML environment



Formal Object Design of ZML



UMLClass

$$\begin{array}{l} name : String \\ attris : String \rightarrow Dtype \\ ops : \mathbb{P}\, String \end{array}$$

UMLDiagram

$$\begin{array}{l} classes : \mathbb{P}\, UMLClass \\ inh, agg : UMLClass \leftrightarrow UMLClass \\ \hline \mathrm{dom}(inh \cup agg) \cup \mathrm{ran}(inh \cup agg) \subseteq classes \\ \forall\, h : classes \bullet (h, h) \notin inh^{+} \end{array}$$

$$\begin{array}{l} project : \mathbb{P}\, Classdef \rightarrow UMLDiagram \\ \hline \forall (oz, uml) : project \bullet \\ \quad \{c : oz \bullet c.name\} = \{c : uml.classes \bullet c.name\} \bullet \\ \quad \forall\, c_1, c_2 : oz \bullet \\ \qquad \exists_1\, c' : uml.classes \bullet \\ \qquad\quad c'.name = c_1.name \\ \qquad\quad c'.attris = \{cls : oz \bullet cls.name\} \lhd c_1.state.decpart \\ \qquad\quad c'.ops = \{o : Opdef \mid o \in c_1.ops \bullet o.name\} \\ \qquad c_2.name \in \{t : \mathrm{ran}\, c_1.state.decpart \bullet t.name\} \Rightarrow \\ \qquad\quad \exists_1 (c_1', c_2') : uml.agg \bullet c_1'.name = c_1.name \wedge c_2'.name = c_2.name \\ \qquad c_2.name \in \{inh : \mathrm{dom}\, c_1.inherit \bullet inh.name\} \Rightarrow \\ \qquad\quad \exists_1 (c_1', c_2') : uml.inh \bullet c_1'.name = c_1.name \wedge c_2'.name = c_2.name \end{array}$$

Basic Implementation Ideas

- ZML: Define a customized XML for Z family languages for web-browsing purpose
- UML tool: Rational Rose 2000 supports XMI import/export according to UML.DTD
- Translation rules are applied using XSLT techniques to automatically translate Object-Z/TCOZ model(XML) to UML diagrams(XMI) and vice versa

Syntax definition

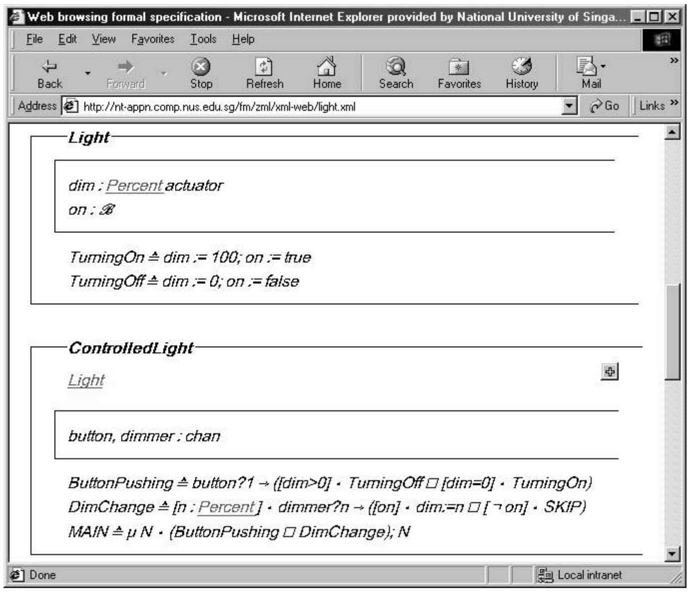
```
<ElementType name="op" content="eltOnly" order="seq">
  <element type="name" minOccurs="1" maxOccurs="1"/>
  <element type="delta" minOccurs="0" maxOccurs="1"/>
  <element type="decl" minOccurs="0" maxOccurs="*"/>
 <element type="predicate" minOccurs="0" maxOccurs="*"/>
  ...
</ElementType>
<ElementType name="classdef" content="eltOnly">
  <element type=state" minOccurs=1" maxOccurs=1"/>
  <element type=init" minOccurs="0" maxOccurs=1"/>
  <element type="op" minOccurs="0" maxOccurs="*"/>
  ...
</ElementType>
```

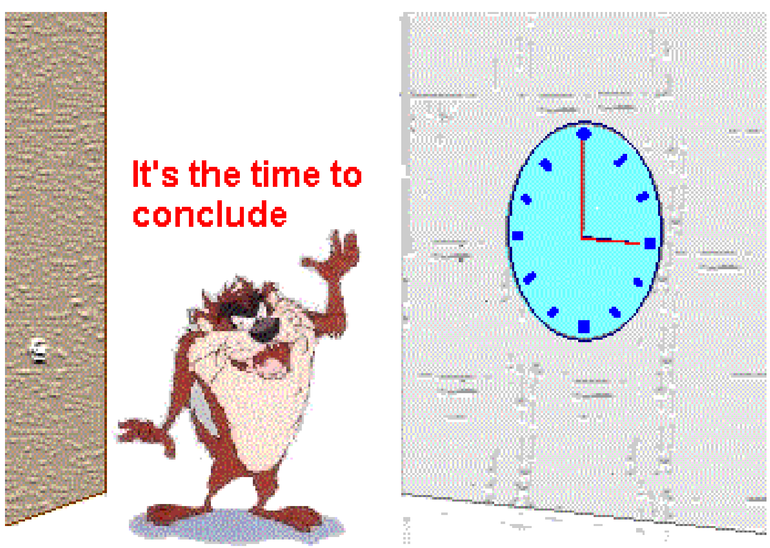
XSL Transformation

```
<xsl:template match="classdef[@layout='simpl'] classdef[@layout='gen']">
<html>
 ...
 <a><xsl:attribute name="name"><xsl:value-of select="name"/></xsl:attribute></a>
 ...
 <xsl:apply-templates select="state"/>
 <xsl:apply-templates select="init"/>
 <xsl:apply-templates select="op"/>
 ...
</html>
</xsl:template>
```


Light example

```
<classdef layout="simpl" align="left">
  <name>Light</name>
  <state>
      <decl>
          <name>dim</name>
          <dtype><type>Percent</type><type>&actuator;</type></dtype></decl>
      <decl>
         <name>on</name>
         <dtype><type>&bool;</type></dtype></decl>
  </state>
  <op layout="calc">
      <name>TurningOn</name>
      <predicate>dim := 100; on := true</predicate> </op>
      ...
</classdef>
```


Web browsing formal specification - Microsoft Internet Explorer provided by National University of Singa...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://nt-appn.comp.nus.edu.sg/fm/zml/xml-web/light.xml
Go Links
Light
dim : Percent actuator
on : 𝔹
TurningOn ≙ dim := 100; on := true
TurningOff ≙ dim := 0; on := false
ControlledLight
Light
button, dimmer : chan
ButtonPushing ≙ button?1 → ([dim>0] • TurningOff □ [dim=0] • TurningOn)
DimChange ≙ [n : Percent] • dimmer?n → ([on] • dim:=n □ [¬ on] • SKIP)
MAIN ≙ μ N • (ButtonPushing □ DimChange); N
Done
Local intranet


It's the time to conclude

Conclusion and Further Work

- State-based (Object-Z), Event-based (Timed CSP), Graph-based (UML)
- TCOZ
  - combines the modelling powers from Object-Z and Timed CSP
  - distinguishes the notion of *active* and *passive* objects
- Further research
  - applications to the specification of
    - software architectures
    - parallel distributed systems
  - tools support
  - Hoare and He's UTP to TCOZ semantics
  - TCOZ refinement rules

TCOZ papers

* J. Sun, J.S. Dong, J. Liu and H. Wang. Object-Z Web Environment and Projections to UML. *WWW-10: 10th International World Wide Web Conference*, ACM Press, May 2001.

• J. Liu, J.S. Dong and J. Sun. TRMCS in TCOZ, *10th International Workshop on S/W Specification and Design (ISSWD'00)*, San Diego, USA, IEEE Press, Nov, 2000.

* J. Liu, J.S. Dong, B. Mahony and K. Shi. Linking UML with Integrated Formal Techniques, *UML: Systems Analysis, Design, and Development Issues*, 2000.

* B. Mahony and J.S. Dong. Timed Communicating Object Z.*IEEE Transactions on Software Engineering*, 26(2):150-177, Feb 2000.

• J.S. Dong, B. Mahony and N. Fulton, Capturing Concurrent Interactions of Mission Computer Tasks, *The 6th Asia-Pacific S/E Conference (APSEC'99)*, IEEE Press, Dec, 1999.

* B. Mahony and J.S. Dong. Sensors and Actuators in TCOZ. *World Congress on Formal Methods (FM'99)*, Lecture Notes in Computer Science, Springer-Verlag, Toulouse, France, Sep 1999.

• B. Mahony and J.S. Dong. Overview of the Semantics of TCOZ. *Integrated Formal Methods (IFM'99)*, Springer-Verlag, York, UK, June 1999.

* J.S. Dong and B. Mahony. Active Object in TCOZ. *the IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, pages 16-25, IEEE Press, Brisbane, Dec 1998.

* B. Mahony and J.S. Dong. Network Topology and a Case Study in TCOZ. *the 11th International Conference of Z Users (ZUM'98)*, LNCS, pp 308-327, Springer-Verlag, Berlin, Sep 1998.

* B. Mahony and J.S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. *the 20th International Conference on S/E (ICSE'98)*, pages 95-104, IEEE Press, Kyoto, April 1998.

Most online versions can be found at: `http://www.comp.nus.edu.sg/~dongjs`

Other integrated approaches (partial collection)

- G. Smith and J. Derrick. Specification, refinement and verification of concurrent systems - an integration of Object-Z and CSP, Formal Methods in System Design, 2001. (To appear.)
- H. Treharne and S. Schneider. How to Drive a B Machine, ZB 2000 , Lecture Notes in Computer Science. Springer-Verlag, 2000.
- C. Fischer. Combination and implementation of processes and data: from CSP-OZ to Java. PhD thesis. University of Oldenburg, 2000.
- H. Wehrheim. Data Abstraction for CSP-OZ. In FM'99: World Congress on Formal Methods, Lecture Notes in Computer Science. Springer-Verlag, 1999.
- M. Butler. csp2B: A Practical Approach to Combining CSP and B. In FM'99: World Congress on Formal Methods, Lecture Notes in Computer Science. Springer-Verlag, 1999.
- C. Bolton, J. Davies and J. Woodcock. On the Refinement and Simulation of Data Types and Processes. In Integrated Formal Methods (IFM'99). Springer-Verlag, 1999.
- C. Suhl. RT-Z: An Integration o Z and timed CSP. In Integrated Formal Methods (IFM'99). Springer-Verlag, 1999.
- K. Taguchi and K. Araki. The State-Based CCS Semantics for Concurrent Z Specification ICFEM'97. IEEE Press, 1997
- A. Galloway and W. Stoddart. An operational semantics for ZCCS. ICFEM'97. IEEE Press, 1997