Part 6 Timed CSP and Integrated Formal Modeling

Overview

- 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

Timed CSP

- Timed CSP extends CSP by introducing a capability to quantify temporal aspects of sequencing and synchronisation. Timing operators i.e., wait, timeout, timed-interrupt are added to CSP
  - S. Schneider. Concurrent and Real-time Systems: The CSP Approach, Wiley, 1999.
  - J. Davies, Specification and Proof in Real-Time CSP, Cambridge University Press, 1993.

Prefix

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

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

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.

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)

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

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

$$\begin{aligned} \textsc{Wait}\, t;\ P &= STOP \triangleright\{t\}\ P \\ a \xrightarrow{t} P &= a \rightarrow \textsc{Wait}\, t;\ P = a \rightarrow (STOP \triangleright\{t\}\ P) \end{aligned}$$

is used to represent P delayed by time t.

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 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.

The generic function ps (purge stale) can be defined as

$$[X]$$

$$ps : (\mathbb{T} \times \mathbb{F}(\mathbb{T} \times X)) \rightarrow \mathbb{F}(\mathbb{T} \times X)$$

$$\forall\, t : \mathbb{T};\ s : \mathbb{F}(\mathbb{T} \times X) \bullet ps(t, s) = \{(t_o, e) : s \mid t_o > t \bullet (t_o - t, e)\}$$

e.g. $ps(2, \{(1, a), (3, b), (7, c)\}) = \{(1, b), (5, c)\}$.

Solution

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

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

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

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

$[X]$

$$find\_oldest : \mathbb{P}_1(\mathbb{T} \times X) \rightarrow (\mathbb{T} \times X)$$

$$\begin{aligned} &\forall\, s : \mathbb{P}_1(\mathbb{T} \times X) \bullet \\ &\quad \exists (t, e) : s \bullet t = min(\mathrm{dom}\, s) \\ &\quad find\_oldest(s) = (t, e) \end{aligned}$$

Summary

For such an example Timed CSP is superior (to Object-Z) as a means of describing process control.

Timed CSP also handles the timing issues of delays and timeouts simply and elegantly. The allowed sequences of events are clearly and concisely determined by the CSP code, there is no need to calculate preconditions nor is any other form of deep reasoning required to understand the ways in which the timed-collection may evolve.

On the other hand, the syntactic treatment of internal state in the above is complex and unwielding, distracting strongly from the basically elegant treatment of the delay and timeout issues.

CSP still has no standard support for state modeling in the form of mathematical toolkits and libraries nor are there modular techniques for constructing and reasoning about complex internal state.

Revisiting Object-Z

$Buffer[X]$

$max : \mathbb{N}$
$items : \mathrm{seq}\, X$
Δ
$size : \mathbb{N}$

$\#items = size \land size \leq max$

INIT

$items = \langle\, \rangle$

$Join$

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

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

$Leave$

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

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

Two Linked Buffers (single thread)

$TwoBuffers[X]$

$b_1, b_2 : Buffer[X]$

$b_1 \neq b_2$

INIT

$b_1.\text{INIT} \land b_2.\text{INIT}$

$Join \mathrel{\widehat{=}} b_1.Join$

$Leave \mathrel{\widehat{=}} b_2.Leave$

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

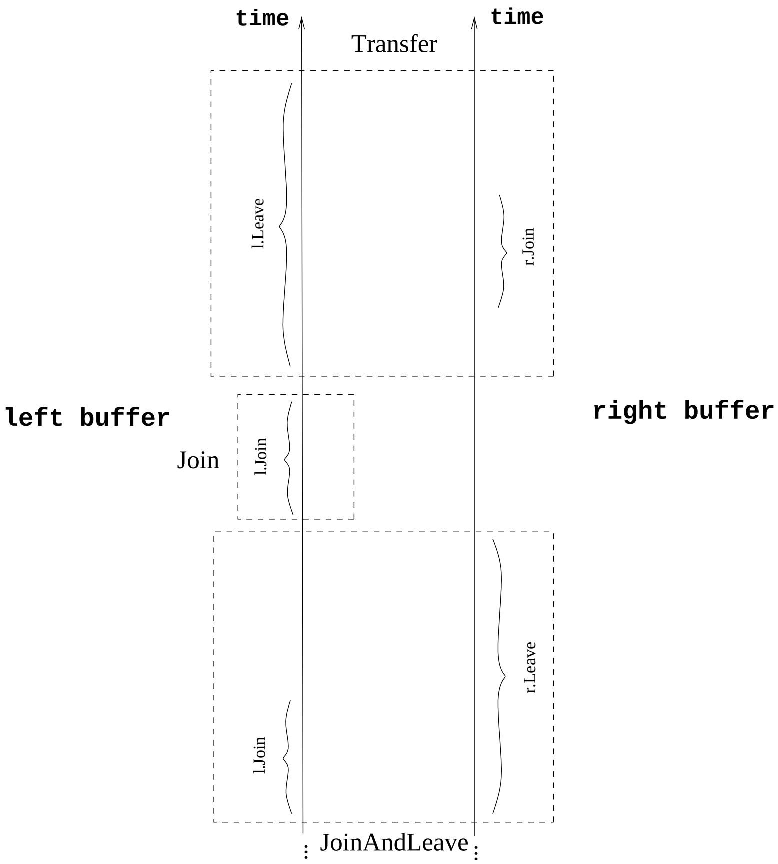


Figure 1: passive events

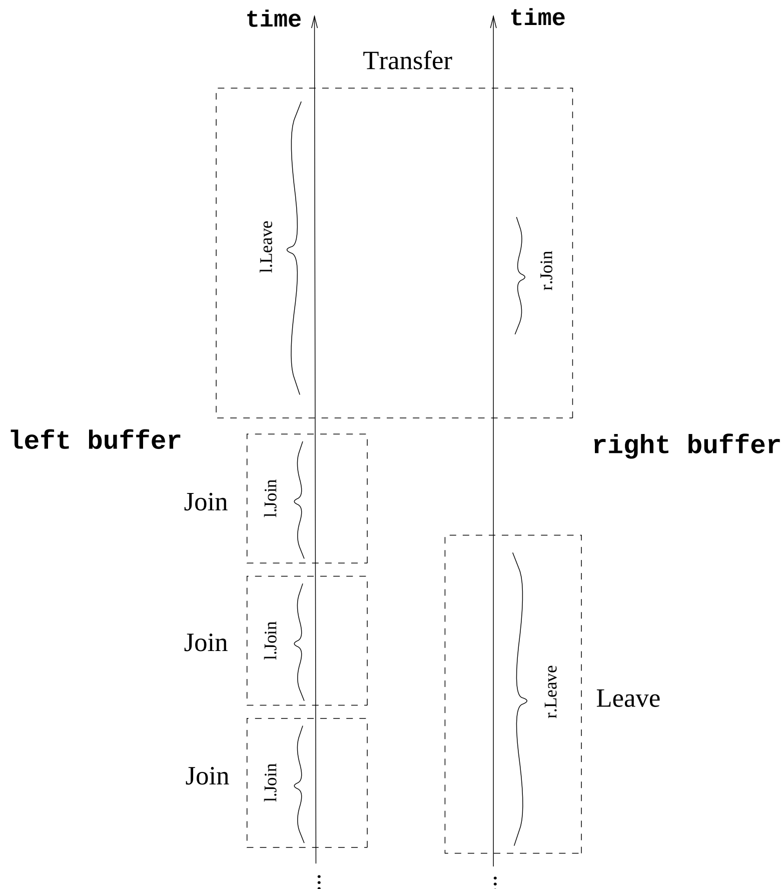


Figure 2: active events

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

$\Delta(items)$

$i? : X$

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

$Remove$

$\Delta(items)$

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

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

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

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

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.

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

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

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

$\ldots$

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

B

$a : A$
$\ldots$

$\ldots$

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

$\ldots$

A

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

$\ldots$

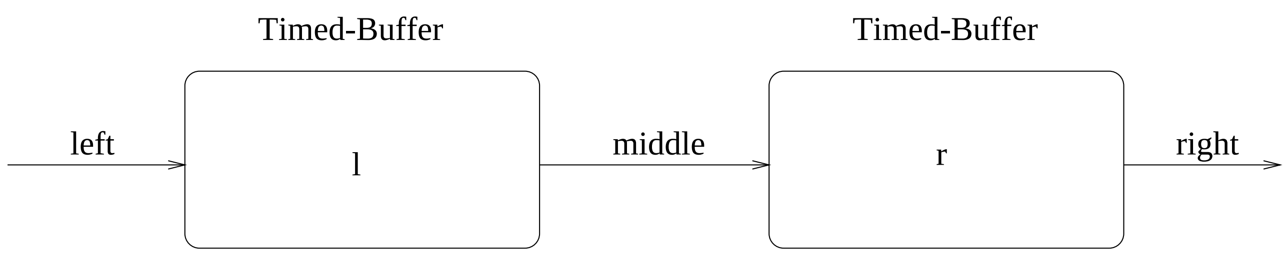
$OpA_1 \mathrel{\widehat=} \ldots$

$\ldots$

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

Identifying the object name with its 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



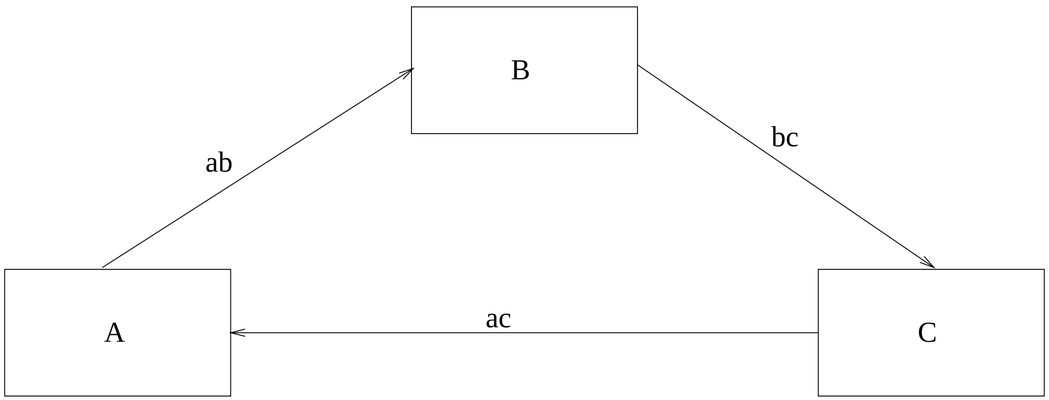
$TwoBuffers[X]$

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

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

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

Complex network topologies



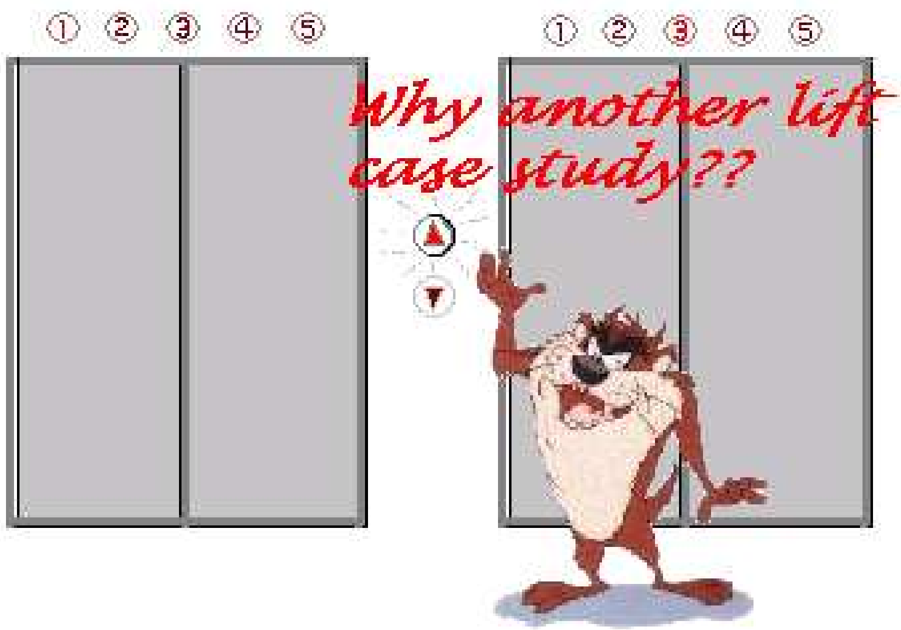
$$(A[bc'/bc] \,||[\, ab, ac \,]||\, (B[ac'/ac] \,||[\, bc \,]||\, C[ab'/ab]) \setminus ab, ac, bc)[ab, ac, bc/ab', ac', bc']$$

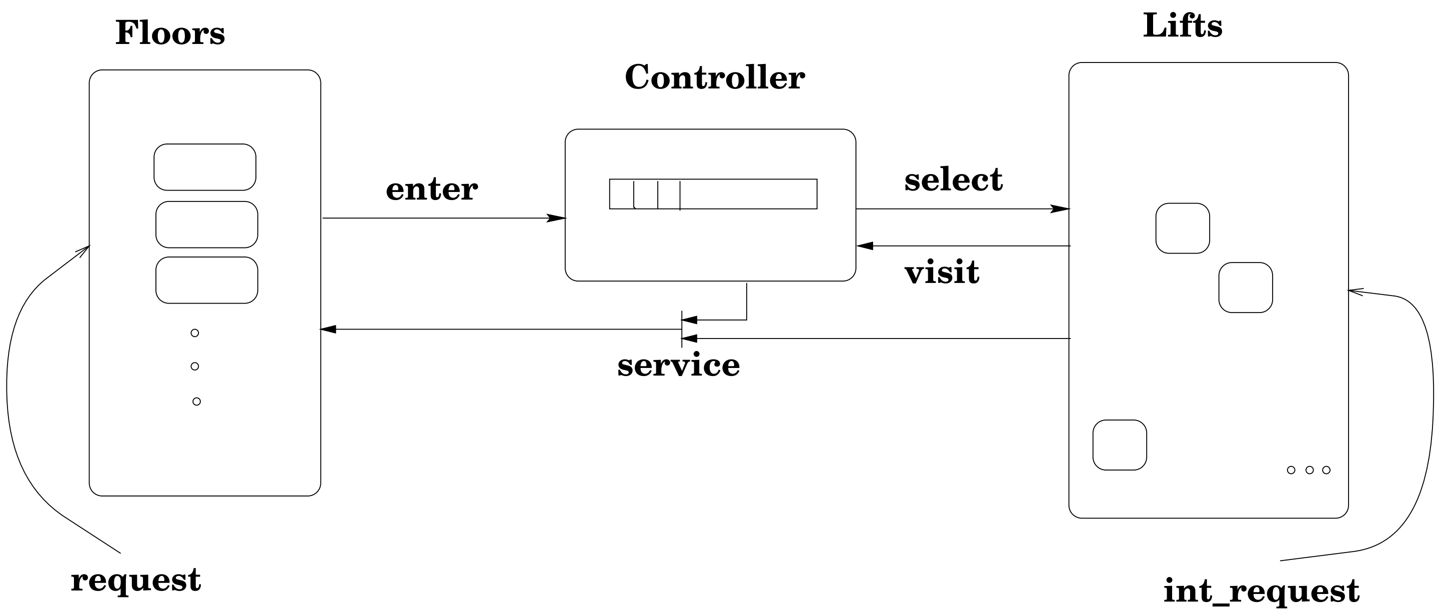
$$\left(\big|\big|\; v_1, v_2, v_3 \ldots \bullet\; v_1 \xleftrightarrow{ch_{12}} v_2,\, v_2 \xleftrightarrow{ch_{23}} v_3,\, v_3 \xleftrightarrow{ch_{13}} v_1, \ldots\right)(A, B, C)$$

$$\big|\big|(A \xleftrightarrow{ab} B, B \xleftrightarrow{bc} C, C \xleftrightarrow{ca} A) \quad or \quad \big|\big|(A \xleftrightarrow{ab} B \xleftrightarrow{bc} C \xleftrightarrow{ca} A)$$

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

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

INIT

$$downbutton.\mathrm{INIT}$$

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

BottomFloor

$upbutton : Button_{©}$

INIT

$upbutton.\mathrm{INIT}$

...

MiddleFloor

$TopFloor, BottomFloor$

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

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

Floors

$floors : \mathrm{seq}\, Floor_{©}$

...

$\mathrm{MAIN} \mathrel{\widehat{=}} \ |||\, f : \mathrm{ran}\, floor$

Lift door control

Door

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

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

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

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

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

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]

$\ldots$

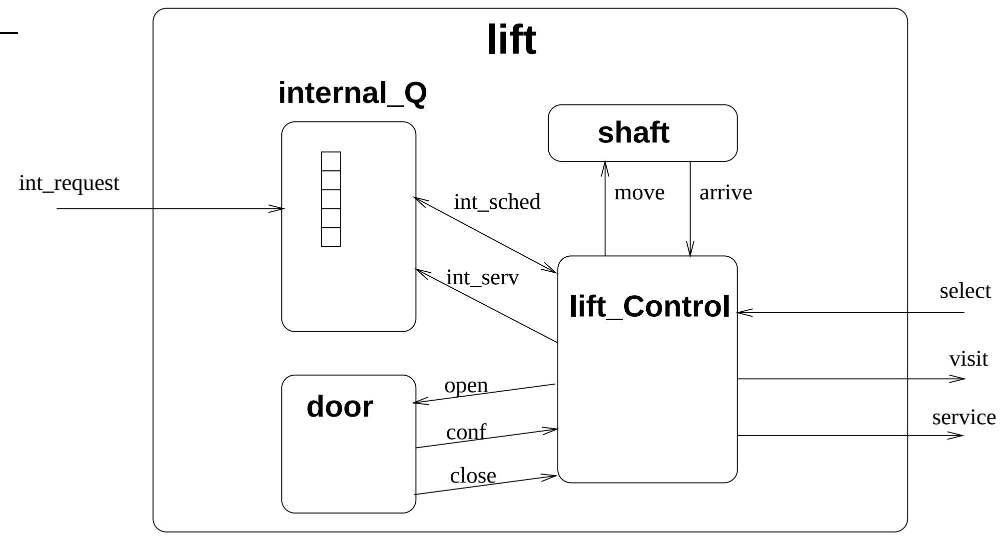
$\ldots$

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

Lift

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

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



Lifts

$$lifts : \mathbb{P}\, Lift_{\copyright}$$

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

Controller

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

Join

...

Remove

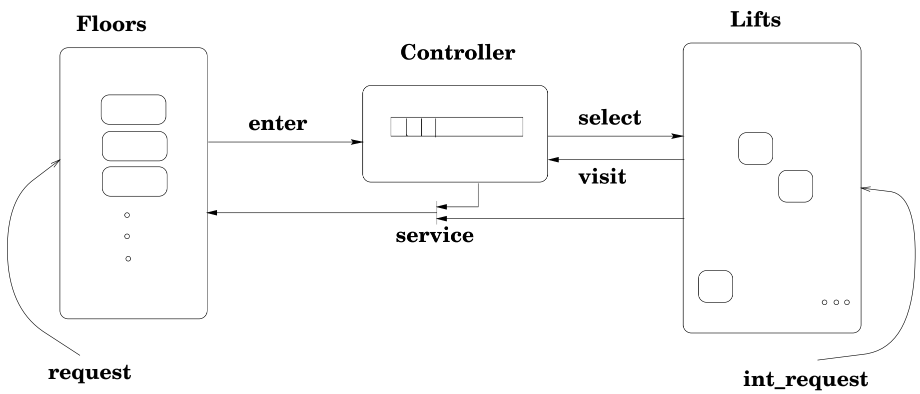
...

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

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

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

The Lift System

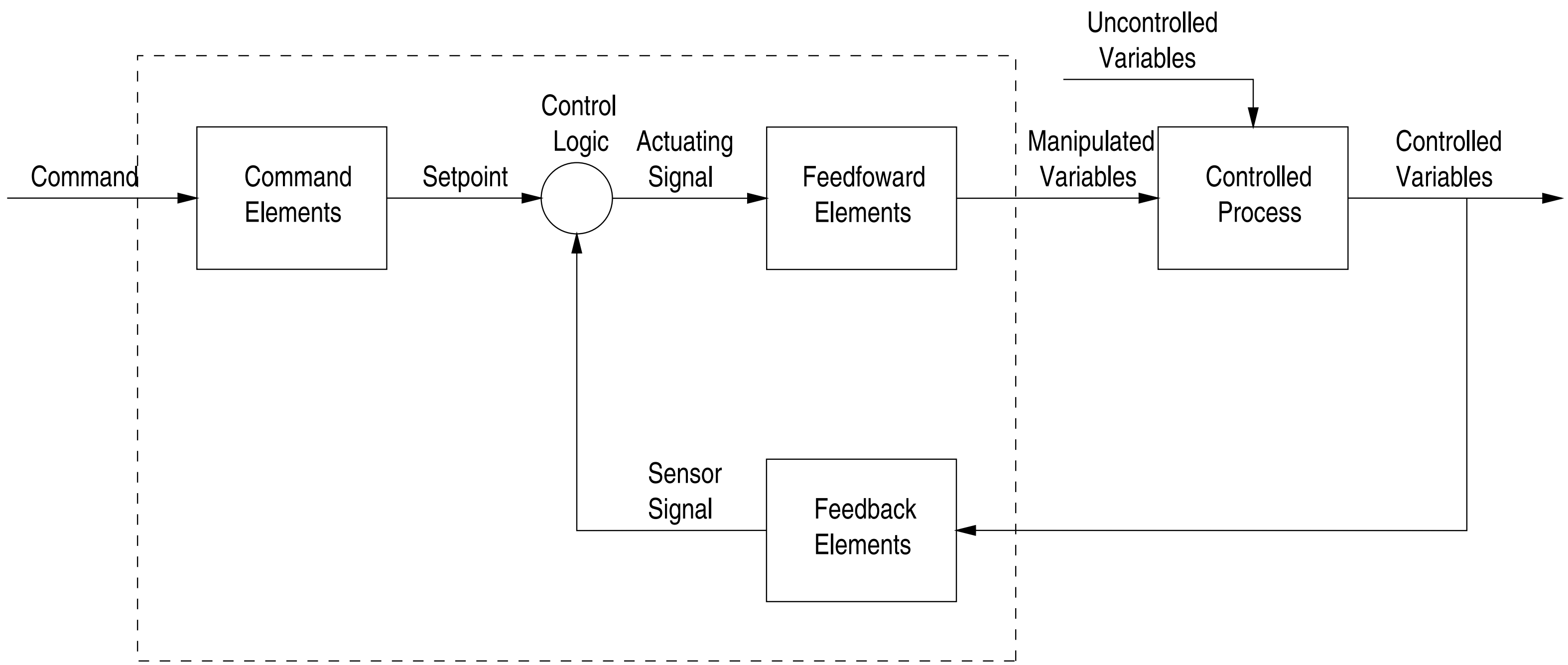


LiftSystem

$fs : Floors_{\copyright}$
$ls : Lifts_{\copyright}$
$contr : Controller_{\copyright}$

$$\textsc{Main} \mathrel{\widehat{=}} \| (fs \overset{enter}{\longleftrightarrow} contr \overset{select, check}{\longleftrightarrow} ls \overset{service}{\longleftrightarrow} fs)$$

Sensors and Actuators — Control Systems



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

Example: Digital Temperature Display

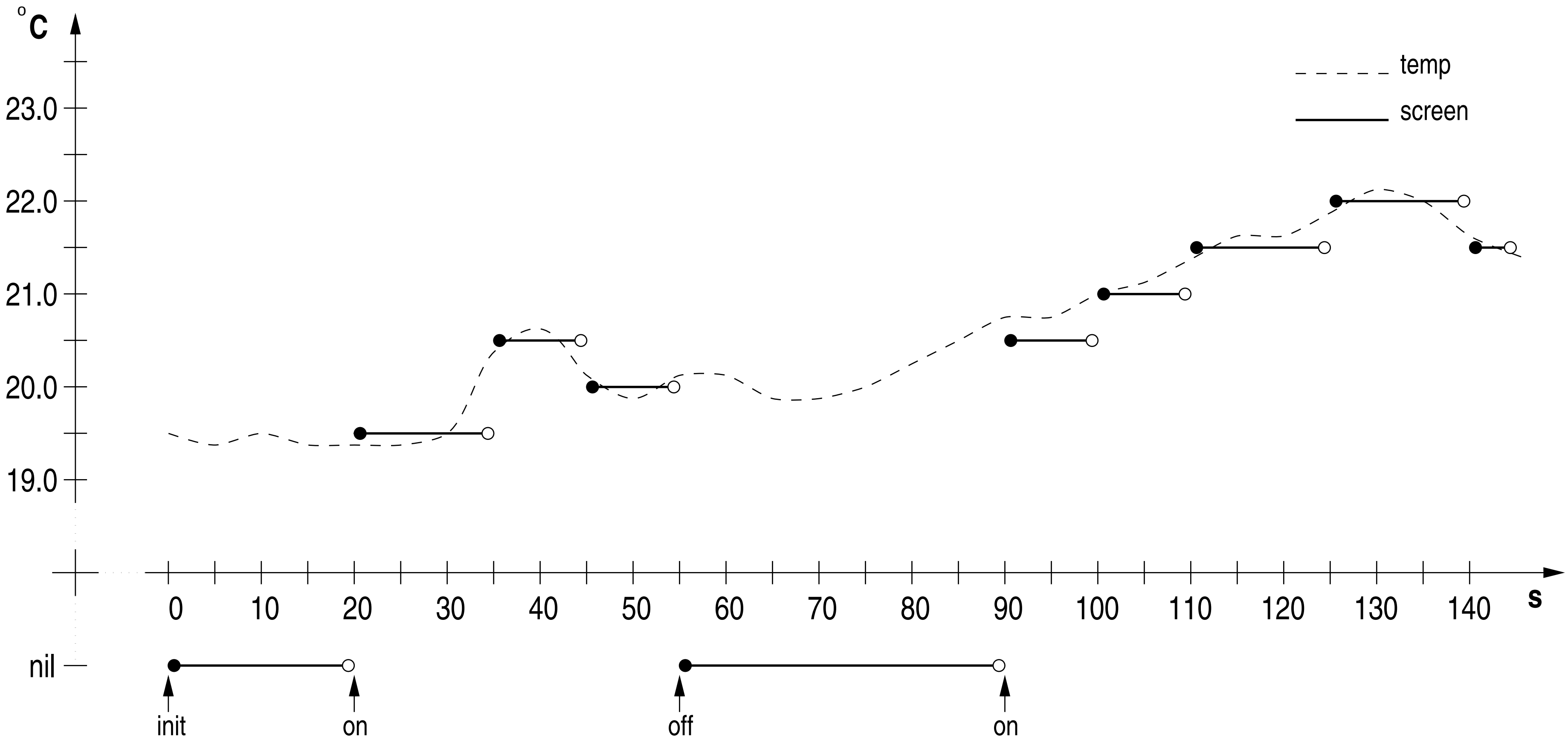


Figure 3: The office communication scenario.

$$temp : \mathbb{R}{}^\circ\mathsf{C}\,\mathbf{sensor}, \quad == \quad temp : \mathbb{R}\,\mathsf{s} \rightarrow \mathbb{R}{}^\circ\mathsf{C}\,.$$

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

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

where

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

The internal role is that of the local state variable.

DTD

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

SetScreen

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

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

INIT

$screen = nil$

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

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

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

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 ...

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

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

... [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

$\Delta(total)$

$total' = total + 1$

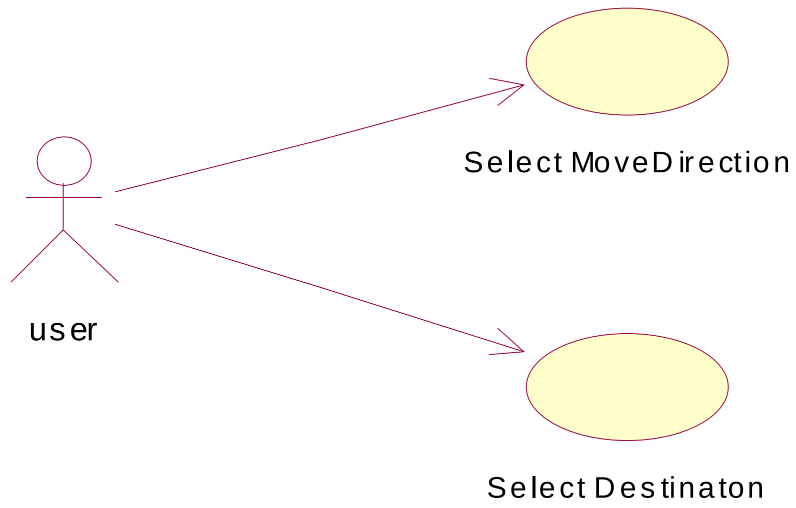
$\textsc{Main} \mathrel{\widehat=} \mu\, C \bullet Inc;\ \textsc{Deadline}\, gain;\ \textsc{WaitUntil}\, per;\ C$

UML

- UML stands for Unified (?) Modeling Language
- The UML combines/collects Data/Class Modeling concepts (extended ER diagrams), Object Modeling, Behaviour Modeling (statechart diagrams) 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 ...

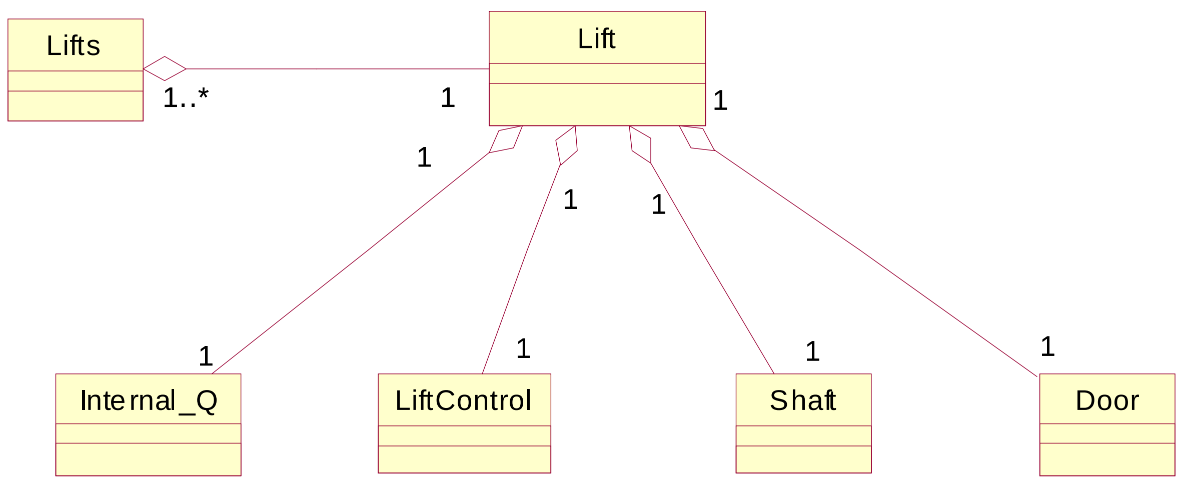
Use Case Diagram

- 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



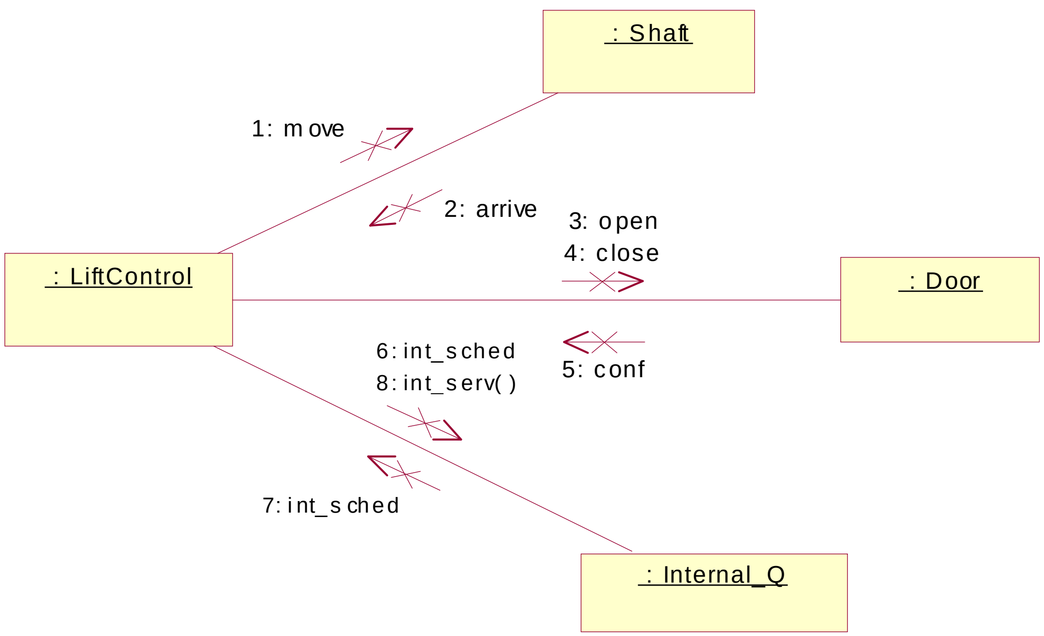
Class Diagram

- 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, association, aggregation, inheritance relationships, multiplicity ...



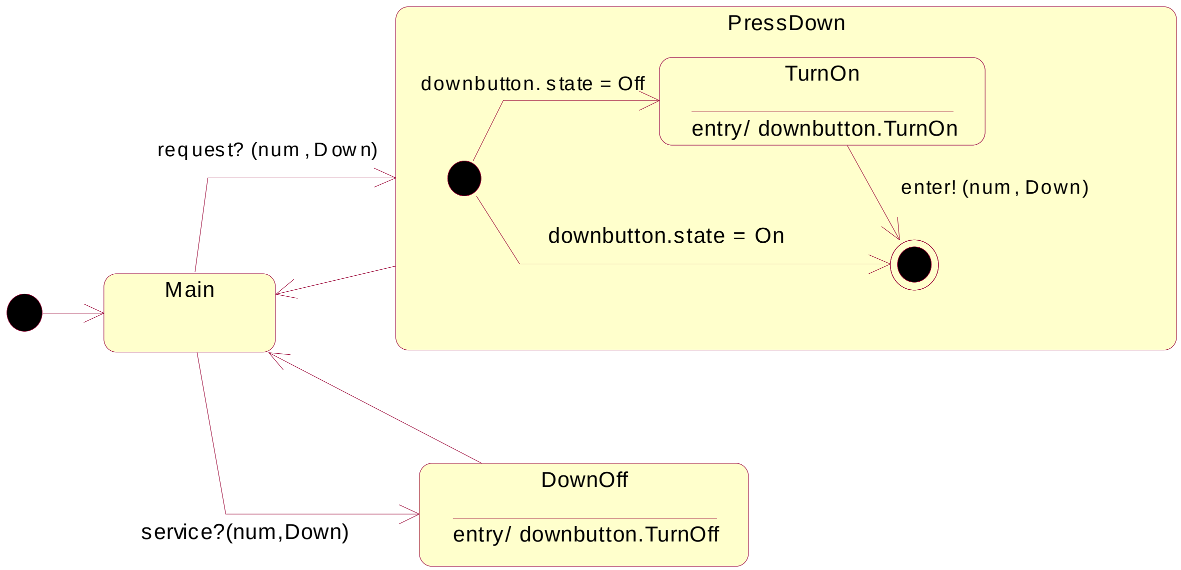
Collaboration Diagram – dynamic behavior, message-oriented

- 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, ... Action Semantics 2001. Therefore, the consistency between diagrams is problematic;
- There are limited capabilities for precisely modeling timed concurrency.

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 complete and coherent 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.

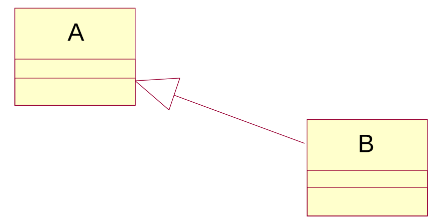
Class

B

A

......

......

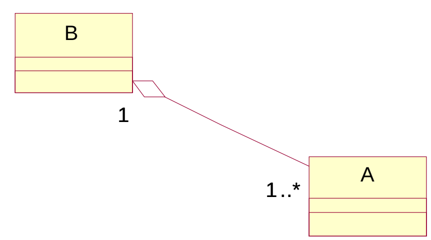


B

$a1, a2 : A$

......

......



Synchronisation

A

c : **chan**
......

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

B

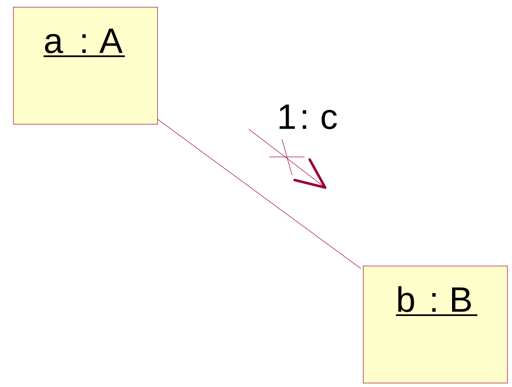
c : **chan**
......

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

AB

$a : A$
$b : B$
......

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



Asynchronisation

A

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

......

B

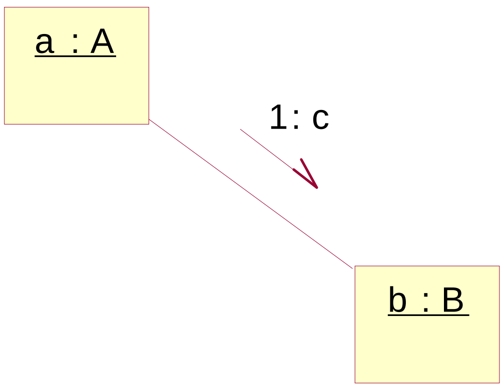
$c : \mathbb{N}$ **sensor**
......

......

AB

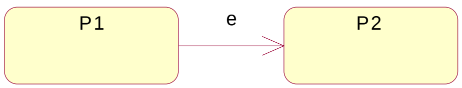
$a : A$
$b : B$
......

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

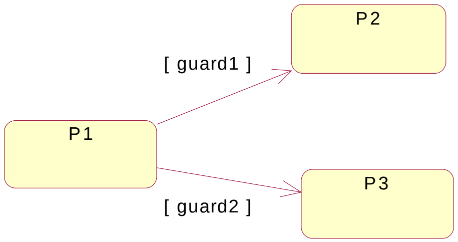


Dynamic Behavior

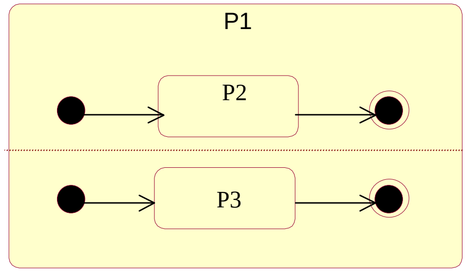
$P1;\ e \rightarrow P2$



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

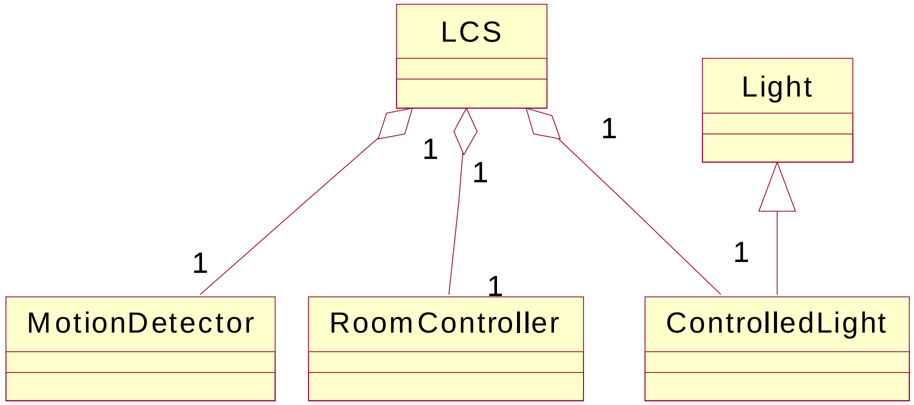


$P1 \mathrel{\widehat{=}} P2 \mid\mid\mid P3$



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.



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

MotionDetector

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

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

Light

$dim : Percent$ **actuator** [dim value]

$on : \mathbb{B}$

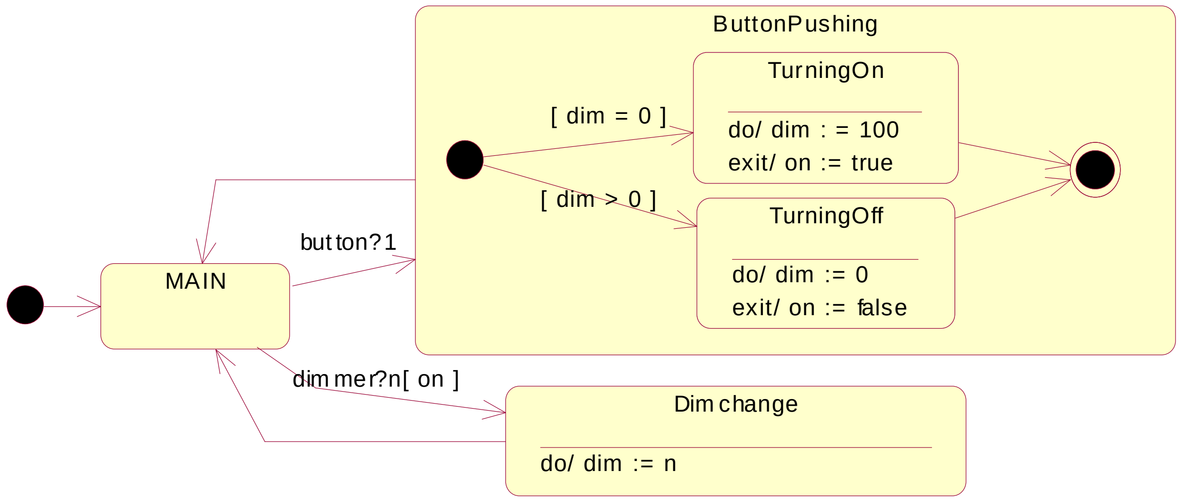
$$TurningOn \mathrel{\widehat=} dim := 100;\ on := true$$
$$TurningOff \mathrel{\widehat=} dim := 0;\ on := false$$

ControlledLight

$Light$

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

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


ButtonPushing
TurningOn
do/ dim : = 100
exit/ on := true
[dim = 0]
[dim > 0]
TurningOff
do/ dim := 0
exit/ on := false
button?1
MAIN
dimmer?n[on]
Dim change
do/ dim := n

$$satisfy : Percent \leftrightarrow Illumination$$

RoomController

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

Adjust

$dim! : Percent$ **on** $dimmer$

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

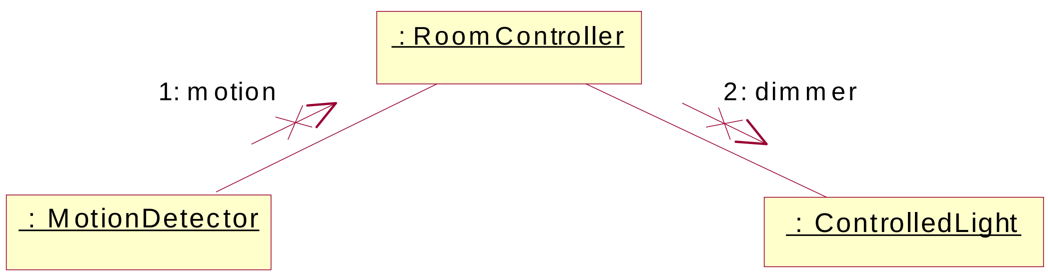
$$Ready \mathrel{\widehat{=}} motion?1 \rightarrow On$$

$$Regular \mathrel{\widehat{=}} \mu\, R \bullet [n : Illumination] \bullet$$
$$\quad odsensor?n \rightarrow olight := n;\ Adjust;\ dimmer!dim \rightarrow R$$
$$On \mathrel{\widehat{=}} Regular \bigtriangledown motion?0 \rightarrow OnAgain$$
$$OnAgain \mathrel{\widehat{=}} (motion?1 \rightarrow On) \triangleright \{absenT\}\ Off$$
$$Off \mathrel{\widehat{=}} dimmer!0 \rightarrow Ready$$
$$\textsc{Main} \mathrel{\widehat{=}} Off$$

LCS

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

$$\text{MAIN} \mathrel{\widehat{=}} \parallel (m \xleftrightarrow{motion} r \xleftrightarrow{dimmer} l)$$

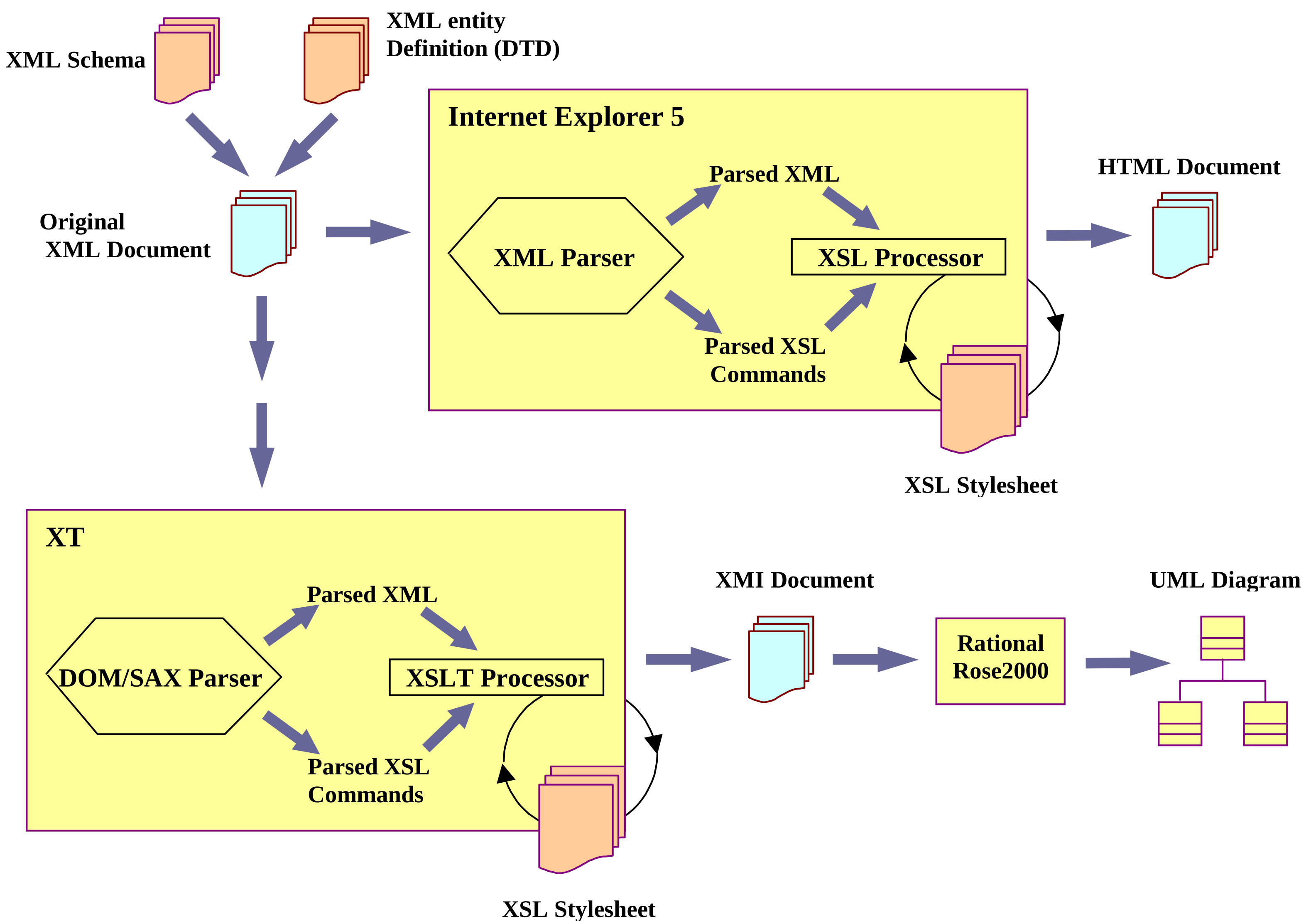


Z Family on the Web with their UML Photos

- Use eXtensible Markup Language (XML) to develop web environment for Z family languages
  - share design models
  - hyperlinks among models
  - advance browsing facilities

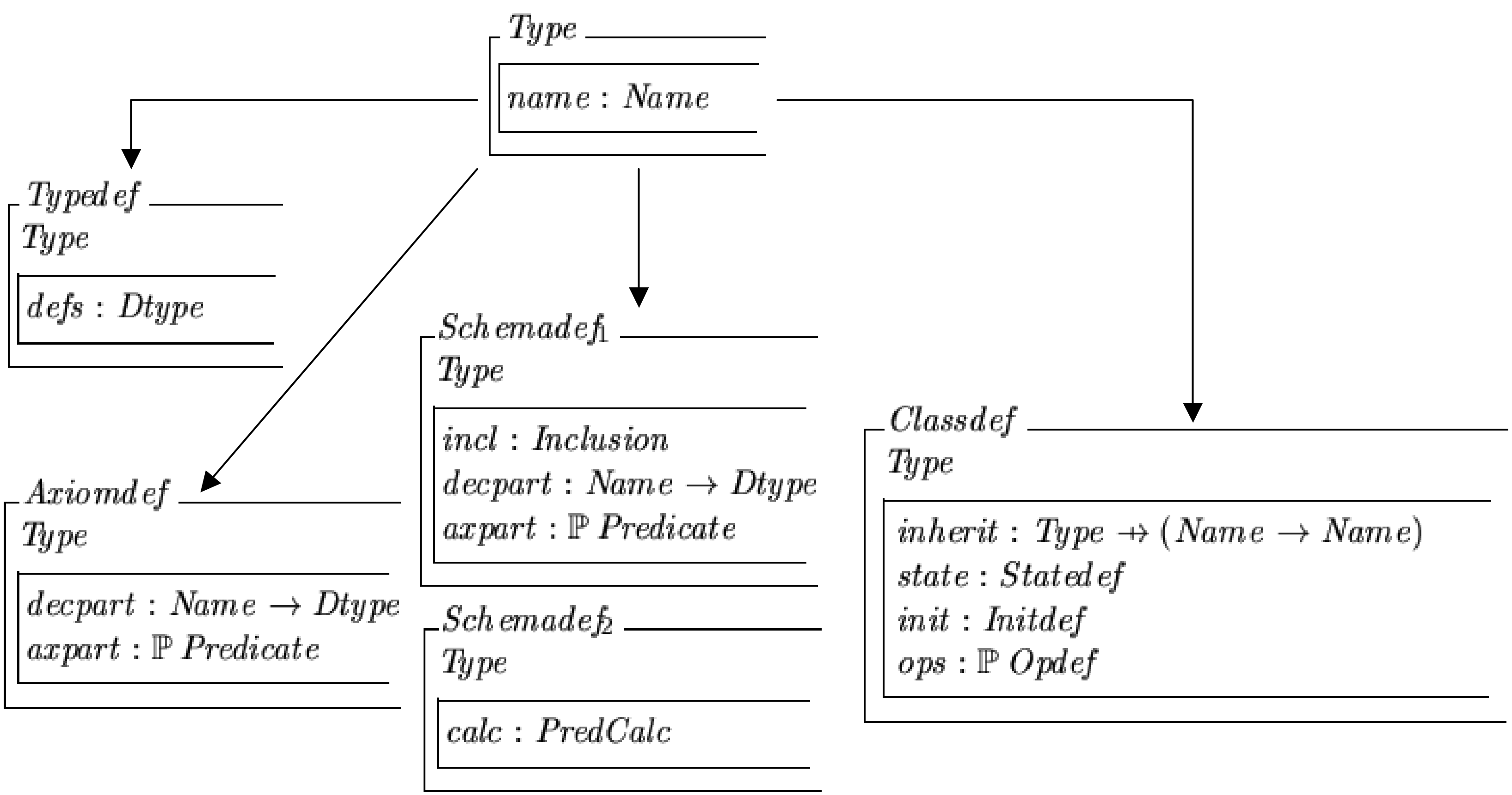
  `http://nt-appn.comp.nus.edu.sg/fm/zml/`

- Develop techniques for projecting (object-oriented) Z models to UML diagrams, based on XML Metadata Interchange (XMI).
- Education tool for helping students through the web to understand:
  - Z schema calculus
  - Object-Z inheritance
  - Relations between Object-Z/TCOZ with UML



XML Schema
XML entity
Definition (DTD)
Original
XML Document
Internet Explorer 5
XML Parser
Parsed XML
XSL Processor
Parsed XSL
Commands
HTML Document
XSL Stylesheet
XT
DOM/SAX Parser
Parsed XML
XSLT Processor
Parsed XSL
Commands
XSL Stylesheet
XMI Document
Rational
Rose2000
UML Diagram

Formal Object Design of ZML



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

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

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

Basic Implementation Ideas

- ZML: Define a customized XML for Z family languages for web-browsing/interchange purposes
- 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

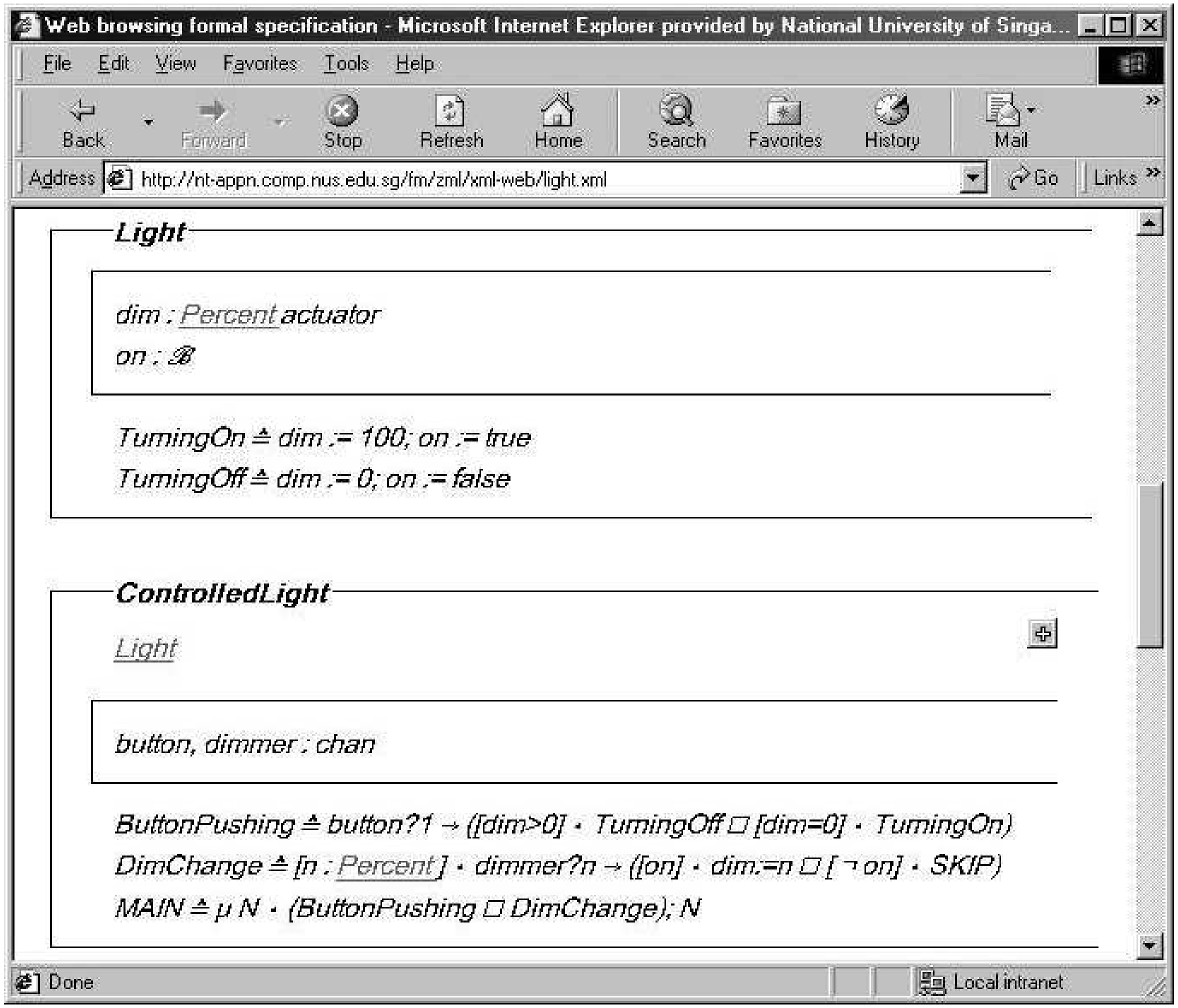
```
<ElementType name="op" content="eltOnly" order="seq">
  <element type="name" minOccurs="1" maxOccurs="1"/>
  <element type="delta" minOccurs="0" maxOccurs="1"/>
  <element type="decl" minOccurs="0" maxOccurs="*"/>
 <element type="predicate" minOccurs="0" maxOccurs="*"/>
  ...
</ElementType>
<ElementType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

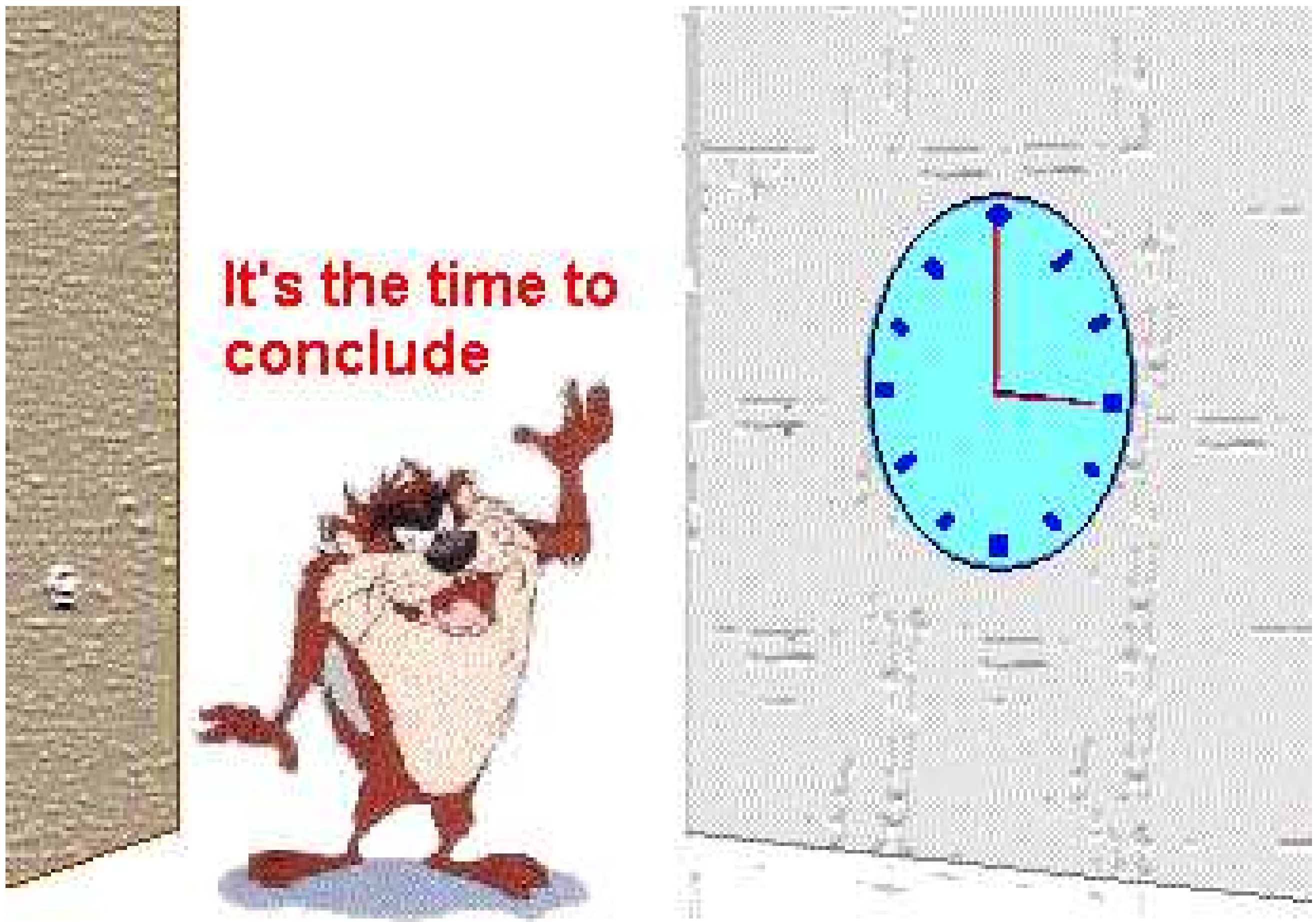
XSL Transformation

```
<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 Research/Studies

- 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/studies
  - applications to the specification of
    - software architectures
    - parallel distributed systems
  - tools support
  - 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, 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.
- S. C. Qin, J. S. Dong and W. N. Chin. A Semantic Foundation of TCOZ in Unifying Theory of Programming. FM'03. LNCS, Springer-Verlag, Pisa, Italy, Sep 2003.
- B. Mahony and J.S. Dong. Deep Semantic Links of TCSP and Object-Z: TCOZ Approach. Formal Aspects of Computing journal, 13:142-160, Springer, 2002.

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

Other integrated approaches (partial collection)

- J. Woodcock, A. Cavalcanti: The Semantics of Circus. ZB 2002: 184-203
- 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.
- 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