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

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:

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

↓ 3 (time passed)

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

↓ $a$ (event occur)

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

↓ 4 (time passed)

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

↓ $b$ (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

\[ \text{Transmit}(x) = \text{send}!x \rightarrow ((\text{ack} \rightarrow \text{STOP}) \triangleright \{5\} \text{Transmit}(x)) \]
Delay

A process which allows no communications for period $t$ then terminates is written $\text{WAIT } t$. The process

$$\text{WAIT } t; \ P \ = \ STOP \triangleright \{t\} \ P$$

$$a \overset{t}{\rightarrow} P \ = \ a \rightarrow \text{WAIT } t; \ P \ = \ a \rightarrow (STOP \triangleright \{t\} \ P)$$

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

$$ps : (T \times F(T \times X)) \rightarrow F(T \times X)$$

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

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

Solution

$TimedCollection \equiv TC_\emptyset$.

$TC_\emptyset \equiv left?e : X \rightarrow TC\{ (t_o, e) \}$

$TC\{ (t,a) \} \cup s \equiv$

$left?e : X \@ t_i \rightarrow TC_{ps}(t_i, \{ (t,a) \} \cup s) \cup \{ (t_o, e) \}$

$right!a \@ t_i \rightarrow TC_{ps}(t_i, s) \triangleright \{ t \} \ TC_{ps}(t, s)$

where $(t, a) = find_{\text{oldest}}(\{ (t, a) \} \cup s)$.

\[\begin{array}{|c|}
\hline
[X] \\
\hline
\end{array}\]

\[\begin{array}{|c|}
\hline
find_{\text{oldest}} : \mathbb{P}_1(\mathbb{T} \times X) \rightarrow (\mathbb{T} \times X) \\
\hline
\end{array}\]

\[\begin{array}{|c|}
\hline
\forall s : \mathbb{P}_1(\mathbb{T} \times X) \bullet \\
\exists(t, e) : s \bullet t = \text{min}(\text{dom } s) \\
find_{\text{oldest}}(s) = (t, e) \\
\hline
\end{array}\]
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\([X]\)

\[
\begin{align*}
\text{max} &: \mathbb{N} \\
\text{items} &: \text{seq } X \\
\Delta \\
\text{size} &: \mathbb{N} \\
\#\text{items} &= \text{size} \land \text{size} \leq \text{max}
\end{align*}
\]

\[
\begin{align*}
\text{Join} \\
\Delta(\text{items}) \\
i? &: X \\
\text{items}' &= \langle i? \rangle \hat{\cap} \text{items}
\end{align*}
\]

\[
\begin{align*}
\text{Leave} \\
\Delta(\text{items}) \\
i! &: X \\
\text{size} \neq 0 \land \text{items} &= \text{items}' \hat{\cap} \langle i! \rangle
\end{align*}
\]

\[
\begin{align*}
\text{Init} \\
\text{items} &= \langle \rangle
\end{align*}
\]
Two Linked Buffers (single thread)

TwoBuffers[$X$]

\[
\begin{align*}
&b_1, b_2 : \text{Buffer}[X] \\
&b_1 \neq b_2 \\
&\text{Join} \equiv b_1.\text{Join} \\
&\text{Leave} \equiv b_2.\text{Leave} \\
&\text{Transfer} \equiv b_1.\text{Leave} || b_2.\text{Join}
\end{align*}
\]

\[
\begin{align*}
&TWOBUFFERS[X] \\
&\text{INIT}\quad b_1.\text{INIT} \land b_2.\text{INIT}
\end{align*}
\]
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*: seq $X$
- *left*, *right*: chan

**Add**

- $\Delta(items)$
- $i? : X$
- $items' = \langle i? \rangle \cdot items$

**Remove**

- $\Delta(items)$
- $items' = items \seArrow last(items)$

**Join** $\triangleq [i : X] \cdot left?i \rightarrow Add; \ Deadline tj$

**Leave** $\triangleq [items \neq \langle \rangle] \cdot right!last(items) \rightarrow Remove; \ Deadline tl$

**Main** $\triangleq \mu Q \cdot (Join \Box Leave); \ Q$
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.

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

Identifying the object name with its MAIN process, e.g. if $ob_1$ and $ob_2$ are active object components, then $ob_1 || ob_2$ means $ob_1$.MAIN ||| $ob_2$.MAIN.
Two Communicating Timed Buffers

\[
\text{TwoBuffers}[X]
\]

\[
l : \text{TimedBuffer}[X][\text{middle/right}] \\
r : \text{TimedBuffer}[X][\text{middle/left}]
\]

\[
\text{MAIN} \defeq (l|[\text{middle}]| r \setminus \text{middle})
\]
Complex network topologies

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

\[\parallel v_1, v_2, v_3... \bullet v_1 \xleftarrow{ch_{12}} v_2, v_2 \xleftarrow{ch_{23}} v_3, v_3 \xleftarrow{ch_{13}} v_1, ... \parallel (A, B, C)\]

\[\parallel (A \leftrightarrow B, B \leftrightarrow C, C \leftrightarrow A) \quad \text{or} \quad \parallel (A \leftrightarrow B \leftrightarrow C \leftrightarrow A)\]
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)
Floors

Controller

Lifts

Detailed model can be found at:

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

<table>
<thead>
<tr>
<th>state : On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta(\text{state}) )</td>
<td>( \Delta(\text{state}) )</td>
</tr>
</tbody>
</table>
| \( \text{state}' = \text{On} \) | \( \text{state}' = \text{Off} \) 

**TopFloor**

| downbutton : Button\( \odot \) request, enter, service : chan |
|---|---|

\[ \text{PressDown} \equiv [\text{downbutton.state} = \text{Off}] \bullet \]
\[ \quad (\text{request?Down} \rightarrow \text{downbutton.TurnOn}); \ (\text{enter!Down} \rightarrow \text{Skip}) \]

\[ \text{DownOff} \equiv \text{service?Down} \rightarrow \text{downbutton.TurnOff} \]

\[ \text{MAIN} \equiv \mu T \bullet (\text{PressDown} \Box \text{DownOff}); \ T \]
BottomFloor

\[
\text{upbutton} : \text{Button}
\]

\[\text{Init}\]

\[\text{upbutton}.\text{INIT}\]

...

MiddleFloor

\[\text{TopFloor}, \text{BottomFloor}\]

\[
\text{Main} \equiv \mu M \bullet (\text{PressDown} \Box \text{DownOff} \Box \text{PressUp} \Box \text{UpOff}); M
\]

\[\text{Floor} \equiv \text{TopFloor} \cup \text{BottomFloor} \cup \text{MiddleFloor}\]

Floors

\[\text{floors} : \text{seq Floor}\]

\[\ldots\]

\[
\text{Main} \equiv \exists! f : \text{ran floor}
\]
Lift door control

Door

\[
\begin{align*}
open, \ conf, \ close, \ servo, \ sensor : \text{chan} \\
\text{OpenDoor}, \ \text{CloseDoor} \equiv \ldots \\
\text{CycleDoor} \equiv \text{OpenDoor}; \ conf \rightarrow \\
&\quad (\mu CD \bullet \text{Wait } t_o; \ \text{CloseDoor} \lor \{\text{sensor?(self, Interrupt)}\} \ \text{OpenDoor}; \ conf \rightarrow CD) \\
\text{Main} \equiv \mu D \bullet \text{open} \rightarrow \text{CycleDoor}; \ \text{close} \rightarrow D
\end{align*}
\]

Moving the lift

Shaft

\[
\begin{align*}
\text{move, arrive : chan} \\
\text{Main} \equiv \mu S \bullet \text{move?n} \rightarrow \text{Wait } |n| \ast \bar{t} + \text{delay}; \ \text{arrive} \rightarrow S
\end{align*}
\]
\[\text{Internal}_Q\]

\[
\begin{align*}
\text{panel} & : \text{seq Button} \\
\text{int}_\text{request}, \text{int}_\text{sched}, \text{int}_\text{serv} & : \text{chan}
\end{align*}
\]

\[\text{NextUp}, \text{NextDown}, \text{MAIN} \triangleq \ldots\]

\[\text{LiftControl}\]

\[
\begin{align*}
\text{fl} & : \mathbb{N} \\
\text{md} & : \text{MoveDirection} \\
\text{move}, \text{arrive} & : \text{chan} \\
& \quad \ldots
\end{align*}
\]

\[
\begin{align*}
\text{...} & \ldots \rightarrow \text{Internal}; \text{LC} \square \\
\text{...} & \rightarrow \text{External}; \text{LC}
\end{align*}
\]

\[\text{MAIN} \triangleq \mu \text{LC} \bullet\]

\[\text{...} \rightarrow \text{Internal}; \text{LC} \square\]

\[\text{...} \rightarrow \text{External}; \text{LC}\]
Lift

\[
iq : \text{Internal}_Q
\]
\[
lc : \text{LiftControl}
\]
\[
s : \text{Shaft}
\]
\[
d : \text{Door}
\]

\[\text{Main} \triangleq \fork\left(\begin{array}{l}
\text{lc} \xrightarrow{\text{move, arrive}} \text{s}; \\
\text{lc} \xrightarrow{\text{open, close, conf}} \text{d}; \\
\text{lc} \xrightarrow{\text{int sched, int serv}} \text{iq}\end{array}\right)\]

Lifts

\[
lifts : \text{P Lift}
\]

\[\text{Main} \triangleq \fork l : \text{lifts}\]
Controller

requests : seq(\mathbb{N} \times MoveDirection)  
enenter, select, visit, service : chan

\begin{align*}
\text{Join} & \cdots \quad \text{Remove} \cdots \\
\vdots & \cdots \quad \vdots & \cdots \\
\end{align*}

\( Dispatch \triangleq \cdots \bullet select!item \rightarrow Remove \)

\( CheckServ \triangleq \)
\[ [item : \mathbb{N} \times MoveDirection] \bullet visit?item \rightarrow \cdots \]

\( \text{MAIN} \triangleq \mu C \bullet (\text{Join} \Box Dispatch \Box CheckServ); \ C \)
The Lift System

LiftSystem

\[ \begin{align*}
fs &: \text{Floors} \\
ls &: \text{Lifts} \\
contr &: \text{Controller}
\end{align*} \]

\[
\text{MAIN} \equiv (fs \xrightarrow{\text{enter}} contr \xrightarrow{\text{select, check}} ls \xrightarrow{\text{service}} fs)
\]
Sensors and Actuators — Control Systems

× CSP channel mechanism is discrete
× CSP channel mechanism is synchronous
Example: Digital Temperature Display

No Wonder it's So Hot

37.6 C
On
Off
Figure 3: The office communication scenario.
\[ temp : \mathbb{R}^\circ C \textbf{sensor}, \quad \Rightarrow \quad temp : \mathbb{R}^s \rightarrow \mathbb{R}^\circ C. \]

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

\[ Display ::= \text{Temp}\langle\langle N \ast 0.5^\circ C\rangle\rangle \mid nil. \]

The internal role is that of the local state variable.
$\text{temp} : \mathbb{R}$ \texttt{sensor}  \\
$\text{screen} : \text{Display}$ \texttt{actuator}  \\
$\text{on, off} : \text{chan}$

\begin{align*}
\text{Init} & \equiv \text{screen} = \text{nil} \\

\text{SetScreen} & \equiv \Delta(\text{screen}) \\
& \equiv t? : \mathbb{R}^\circ C  \\
& \equiv \exists dt : \mathbb{N} \times 0.5^\circ C \bullet \\
& \quad \quad dt = t \pm 0.5^\circ C \land \\
& \quad \quad \text{screen}' = \text{Temp}(dt) \\

\text{Show} & \equiv ([t : \mathbb{R}^\circ C] \bullet \text{temp}?t \rightarrow \text{SetScreen}; \text{Deadline 5 s}; \text{WaitUntil 5 s}; \text{Show}) \\
& \quad \quad \quad \quad \downarrow \text{off} \rightarrow \text{NoShow} \\

\text{NoShow} & \equiv \text{screen} := \text{nil}; \text{on} \rightarrow \text{Show} \\
\text{Main} & \equiv \text{on} \rightarrow \text{Show}
\end{align*}
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 ...

\[ \text{CalendarTime} \equiv \mathbb{N}_{yr} \times \mathbb{N}_{mn} \times \mathbb{N}_{dy} \times \mathbb{N}_{hr} \times \mathbb{N}_{min} \times \mathbb{N}_{s}. \]

\[
\text{Convert} : \mathbb{N}_s \rightarrow \text{CalendarTime}
\]

... [detail of function omitted]
Solution

\[
\begin{align*}
\text{Clock} \\
\text{per} & \equiv 1 \text{s} \\
\text{gain} & \equiv 50 \text{ms} \\
\text{total} & \equiv \mathbb{N}s \\
\Delta & \\
\text{display} & \equiv \text{CalendarTime} \textbf{actuator} \\
\text{display} & \equiv \text{Convert(} \text{total} \text{)} \\
\text{Inc} & \\
\Delta(\text{total}) & \\
\text{total}' & \equiv \text{total} + 1 \\
\text{Main} & \equiv \mu \ C \cdot \text{Inc}; \ \text{Deadline gain}; \ \text{WaitUntil per}; \ \text{C}
\end{align*}
\]
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.

[Diagram]

- User
- Select Move Direction
- Select Destination
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 ...
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

\[ A \]
\[ c : \text{chan} \]
\[ \ldots \]
\[ Main \doteq \ldots c!\ldots \]

\[ B \]
\[ c : \text{chan} \]
\[ \ldots \]
\[ Main \doteq \ldots c?\ldots \]

\[ AB \]
\[ a : A \]
\[ b : B \]
\[ \ldots \]
\[ Main \doteq \ldots a \xleftarrow{c} \ b\ldots \]
Asynchronisation

\[ A \]
\[ c : N_{\text{actuator}} \]
\[ \ldots \]
\[ \ldots \]

\[ B \]
\[ c : N_{\text{sensor}} \]
\[ \ldots \]
\[ \ldots \]

\[ AB \]
\[ a : A \]
\[ b : B \]
\[ \ldots \]
\[ Main \equiv ...a \overset{c}{\longrightarrow} b... \]
Dynamic Behavior

\[ P_1; e \rightarrow P_2 \]

\[ P_1; ([\text{guard}1] \cdot P_2 \quad [\text{guard}2] \cdot P_3) \]

\[ P_1 \equiv P_2 \ || \ P_3 \]
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 lux
Percent == \{0\} \cup 10..100

\begin{align*}
\text{MotionDetector} \\
\begin{array}{|l|}
\hline
\text{motion} : \text{chan} \\
\text{md} : (\text{Move} \mid \text{NoMove}) \text{sensor} \\
\hline
\end{array}
\end{align*}

\begin{align*}
\text{NoUser} \triangleq \text{md?Move} \to \text{motion!1} \to \text{User} \\
\text{md?NoMove} \to \text{WAIT 1 s}; \text{NoUser} \\
\text{User} \triangleq \text{md?NoMove} \to \text{motion!0} \to \text{NoUser} \\
\text{md?Move} \to \text{WAIT 1 s}; \text{User} \\
\text{MAIN} \triangleq \text{NoUser}
\end{align*}
Light

\[
\begin{align*}
\text{dim} &: \text{Percent actuator} \\
\text{on} &: \mathbb{B}
\end{align*}
\]

\[
\begin{align*}
\text{TurningOn} &\equiv \text{dim} := 100; \text{on} := \text{true} \\
\text{TurningOff} &\equiv \text{dim} := 0; \text{on} := \text{false}
\end{align*}
\]

ControlledLight

Light

\[
\begin{align*}
\text{button, dimmer} &: \text{chan} \\
\text{control channels}
\end{align*}
\]

\[
\begin{align*}
\text{ButtonPushing} &\equiv \text{button}?1 \rightarrow \\
&\hspace{1em} ([\text{dim} > 0] \bullet \text{TurningOff} \quad \square \quad [\text{dim} = 0] \bullet \text{TurningOn}) \\
\text{DimChange} &\equiv [n : \text{Percent}] \bullet \text{dimmer}?n \rightarrow \\
&\hspace{1em} ([\text{on}] \bullet \text{dim} := n \quad \square \quad [\neg \text{on}] \bullet \text{SKIP}) \\
\text{MAIN} &\equiv \mu N \bullet (\text{ButtonPushing} \quad \square \quad \text{DimChange}); \ N
\end{align*}
\]
ButtonPushing

TurningOn
\[ \text{do/ } \dim := 100 \]
\[ \text{exit/ } \text{on} := \text{true} \]

TurningOff
\[ \text{do/ } \dim := 0 \]
\[ \text{exit/ } \text{on} := \text{false} \]

Dimchange
\[ \text{do/ } \dim := n \]
satisfy : Percent $\leftrightarrow$ Illumination

RoomController

\[ \text{dimmer, motion : chan} \]
\[ \text{odsensor : Illumination sensor} \]
\[ \text{absenT : } \top \]
\[ \text{olight : Illumination} \]

\[ \text{Ready } \equiv \text{motion?1 } \rightarrow \text{On} \]
\[ \text{Regular } \equiv \mu R \bullet [n : \text{Illumination}] \bullet \]
\[ \text{odsensor?n } \rightarrow \text{olight := n; Adjust; dimmer!dim } \rightarrow R \]
\[ \text{On } \equiv \text{Regular } \triangledown \text{motion?0 } \rightarrow \text{OnAgain} \]
\[ \text{OnAgain } \equiv (\text{motion?1 } \rightarrow \text{On}) \triangleright \{\text{absenT}\} \text{ Off} \]
\[ \text{Off } \equiv \text{dimmer!0 } \rightarrow \text{Ready} \]
\[ \text{MAIN } \equiv \text{Off} \]
\[ LCS \]

\[
m : MotionDetector
l : ControlledLight
r : RoomController
\]

\[
\text{Main} \cong \|(m \xRightarrow{\text{motion}} r \xRightarrow{\text{dimmer}} l)\|
\]
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
Formal Object Design of ZML
\textbf{UMLClass}  
\begin{align*}
\text{name} &: \text{String} \\
\text{attris} &: \text{String} \rightarrow \text{Dtype} \\
\text{ops} &: \mathbb{P} \text{String}
\end{align*}

\textbf{UMLDiagram}  
\begin{align*}
\text{classes} &: \mathbb{P} \text{UMLClass} \\
\text{inh, agg} &: \text{UMLClass} \leftrightarrow \text{UMLClass} \\
\text{dom(inh} \cup \text{agg) } \cup \text{ran(inh} \cup \text{agg)} &\subseteq \text{classes} \\
\forall h &: \text{classes} \bullet (h, h) \notin \text{inh}^+
\end{align*}

\text{project} : \mathbb{P} \text{Classdef} \rightarrow \text{UMLDiagram}

\begin{align*}
\forall (oz, uml) : \text{project} \bullet \\
\{ c : oz \bullet c.\text{name} \} &= \{ c : \text{uml.\text{classes}} \bullet c.\text{name} \} \bullet \\
\forall c_1, c_2 : oz \bullet \\
\exists_1 c' : \text{uml.\text{classes}} \bullet \\
& c'.\text{name} = c_1.\text{name} \\
& c'.\text{attris} = \{ \text{cls} : oz \bullet \text{cls.\text{name}} \} \triangleleft c_1.\text{state.\text{decpart}} \\
& c'.\text{ops} = \{ o : \text{Opdef} \mid o \in c_1.\text{ops} \bullet o.\text{name} \} \\
& c_2.\text{name} \in \{ t : \text{ran c_1.\text{state.\text{decpart}}} \bullet t.\text{name} \} \Rightarrow \\
& \exists_1 (c'_1, c'_2) : \text{uml.\text{agg}} \bullet c'_1.\text{name} = c_1.\text{name} \land c'_2.\text{name} = c_2.\text{name} \\
& c_2.\text{name} \in \{ \text{inh} : \text{dom c_1.\text{inherit}} \bullet \text{inh.\text{name}} \} \Rightarrow \\
& \exists_1 (c'_1, c'_2) : \text{uml.\text{inh}} \bullet c'_1.\text{name} = c_1.\text{name} \land c'_2.\text{name} = c_2.\text{name}
\end{align*}
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

```xml
<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>
Light

dim : Percent actuator
on : bool

TurningOn = dim := 100; on := true
TurningOff = dim := 0; on := false

ControlledLight

Light

button, dimmer : chan

ButtonPushing = button?i -> ([dim>0] • TurningOff □ [dim=0] • TurningOn)
DimChange = [n : Percent] • dimmer?n -> ([on] • dim:=n □ [¬ on] • SKIP)
MAIN = p N • (ButtonPushing □ DimChange), N
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


Most online versions can be found at: http://www.comp.nus.edu.sg/~dongjs
Other integrated approaches (partial collection)