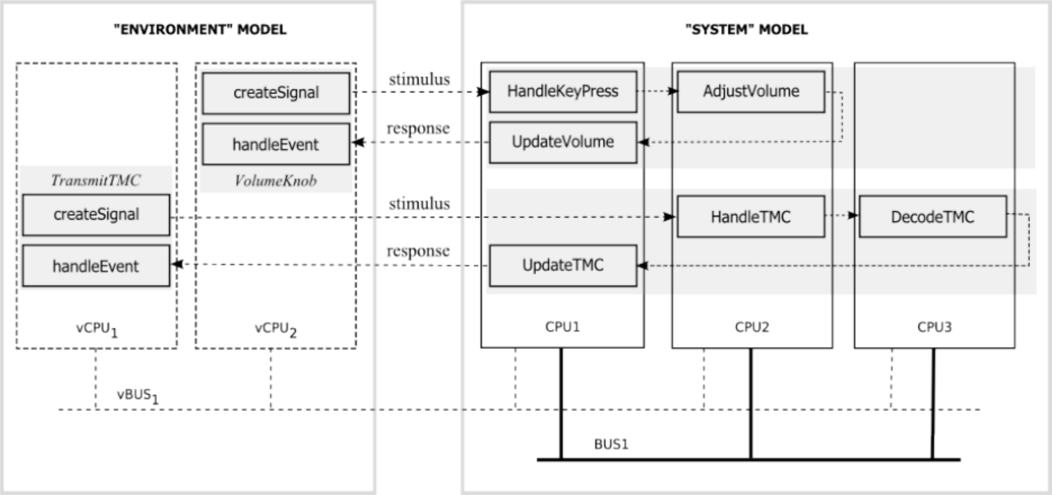
Preparing Overture for the Future – VDM10++ / VDM1X

suggestions for possible language extensions

expressiveness versus analysability

(1) Two steps forward – One step back

- virtual CPUs for all environment stimuli
  (as originally proposed in FM06 paper)
- Required in order to specify multiple periodic environment processes with *independent* timing behaviors



(2) Referencing Time - A

```
class ExampleOne

operations
    op1: () ==> ()
    op1 () == skip

sync
    per op1 => time > 500

end ExampleOne
```

(2) Referencing Time - B

(reminder: #req - #act - #fin → *when* did they happen?)

```
class ExampleTwo

operations
    public static async isr : () ==> ()
    isr () == skip

    -- specify the ISR response time latency
    pre time(#act(isr)) – time(#req(isr)) < 50

    -- specify the ISR maximum elapse time
    post time(#fin(isr)) – time (#req(isr)) < 150

end ExampleTwo
```

(3) Specifying Sporadic Threads

- Periodic threads are now specified with 4-tuple (*p*, *j*, *d*, *o*)
- How to specify sporadic threads? Use (**nil**, **nil**, *d*, **nil**)?

```
class ExampleThree
operations
    op1 : () ==> ()
    op1 () == skip

threads
    sporadic (100) op1

end ExampleThree
```

(4) Thread Specifications - A

- Allow multiple thread definitions per class?

- Allow initializing parameters passed to thread operation?

- Thread operation must be (or implicitly is) asynchronous?

(4) Thread Specifications - B

```
class ExampleFour

operations
    public async ptr: nat ==> ()
    ptr (x) == …

threads
    periodic (1000, 10, 10, 0) ptr (0);
    sporadic (500) ptr (15)

end ExampleFour
```

(5) duration and cycles - A

- Allow general expressions instead of literals

```
class ExampleFive

operations
    public async op1: nat ==> ()
    op1 (x) == duration (10 * x) skip

end ExampleFive
```

(5) duration and cycles - B

- Allow specification of intervals
- Non-deterministic choice from interval on elaboration
- Possibly overruled by simulator global setting
  (i.e. normal, exponential distribution with parameters)

```
class ExampleSix

operations
  public async op1 : nat ==> ()
  op1 (x) == duration (10 * x, 20 * x) skip

end ExampleSix
```

(5) duration and cycles - C

```
class ExampleSeven

instance variables
    invoked : nat := 0

operations
    op1: nat * nat ==> nat
    op1 (x, y) ==
      (     invoked := invoked + 1;
            return if (y – invoked > x) then y – invoked else x );

    public async op2: nat ==> ()
    op2 (x) == duration (10, 100, op1) skip

end ExampleSeven
```

Not covered (but interesting!)

- Dynamic deployment (cf. Nielsen)
- Multiple communication paths between CPUs
- Faulty communications (message loss)
- Message broadcasting (multiple receivers)
- Configurable communication buffer depths
- Configurable scheduling protocols