UML Modelling Case Study: A GSM BTS

Sorin Manolache, `sorma@ida.liu.se`
Linköping University, IDA/ESLAB

Motivation

- Investigate the versatility of UML as an implementation independent specification language for complex hardware/ software real-time systems
- Possibly perform architecture exploration on the model

Therefore...

Specification Requirements

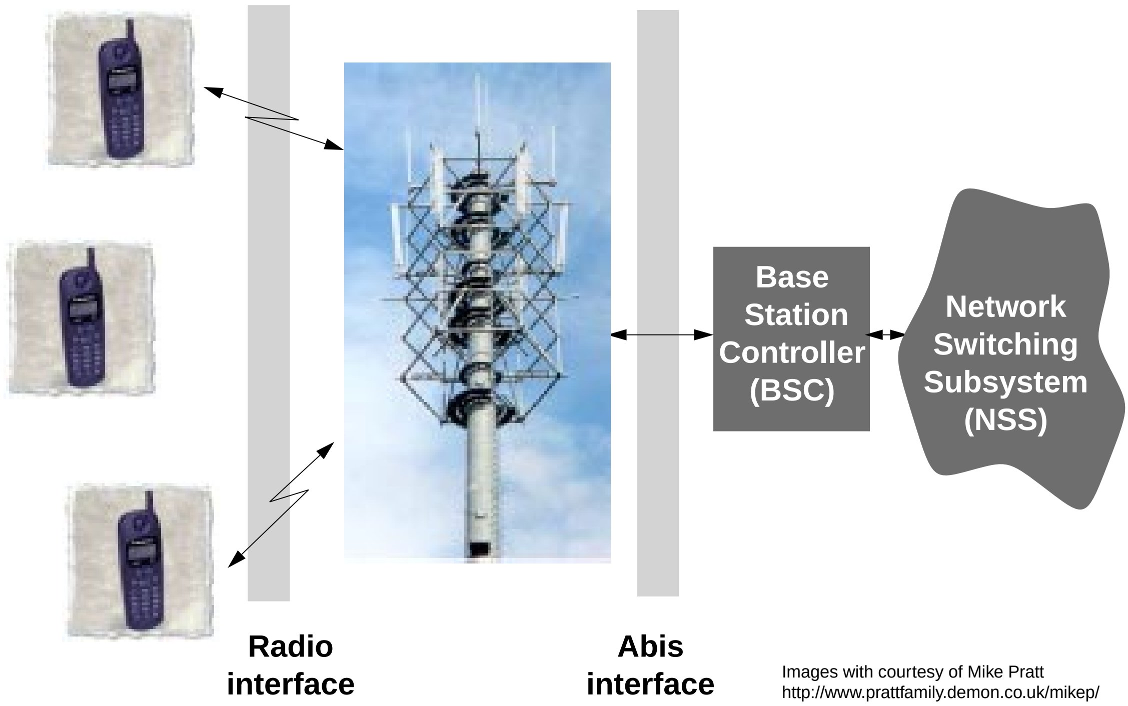
- executable specification
- (explicit) specification of time constraints and execution time
- specification of concurrency
- modelling of resources (hw: processors, comm. links, memory, etc.; sw: shared data, message queues, locks, etc.)
- specifying the binding
- scheduling policies and scheduling units
- etc. [UML Profile for Scheduling, Performance and Time — `http://www.omg.org/cgi-bin/doc?ad/99-03-13`]

Application Characteristics

Application implies:

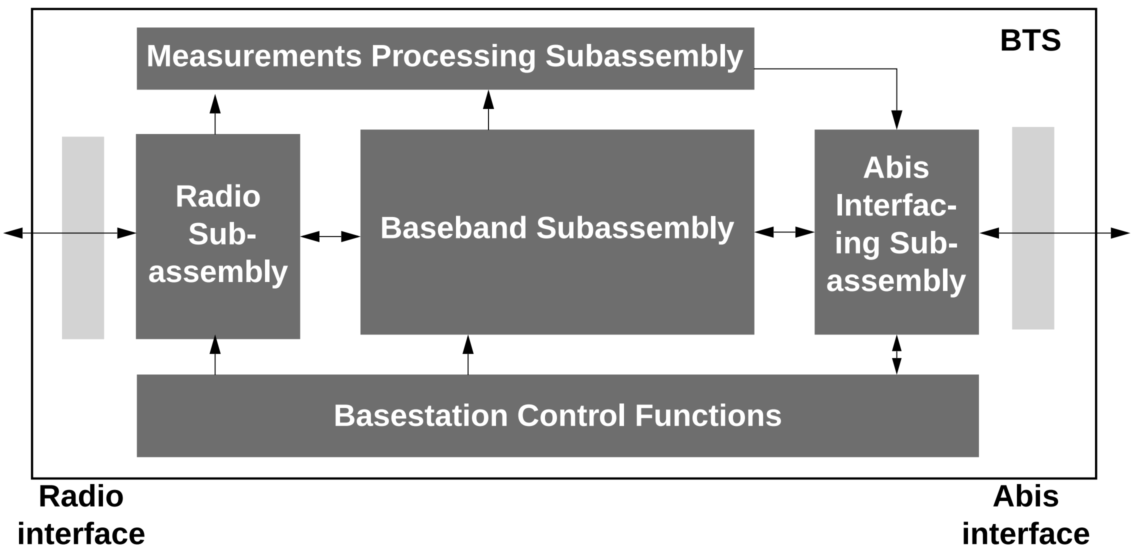
- data oriented processing
- control oriented processing
- protocol processing
- analog part
- concurrency

The GSM BTS (1)

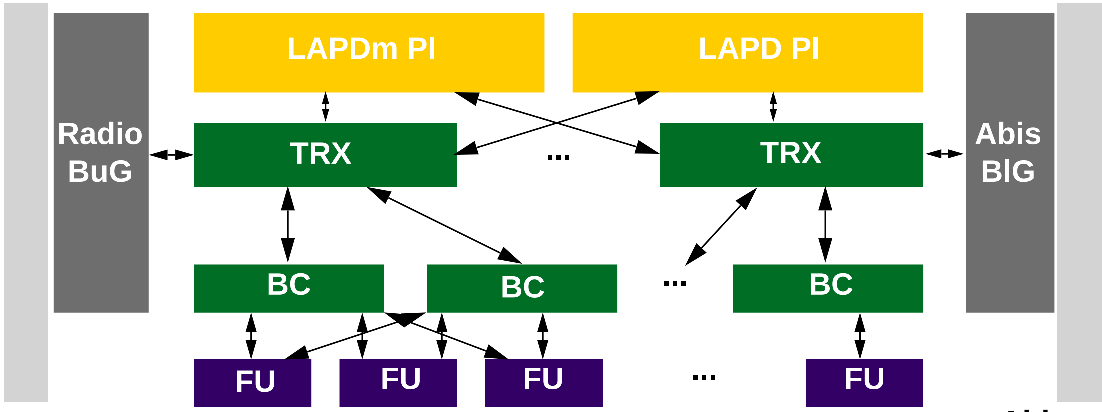


Images with courtesy of Mike Pratt
http://www.prattfamily.demon.co.uk/mikep/

The GSM BTS (2)



Model Architecture



Radio interface

FU – functional unit
BC – baseband controller
TRX – transmitter-receiver

PI – protocol interpreter
BuG – burst generator
BIG – block generator

Abis interface

Protocol processing
Control oriented
Data oriented
Specified at a high level of abstraction

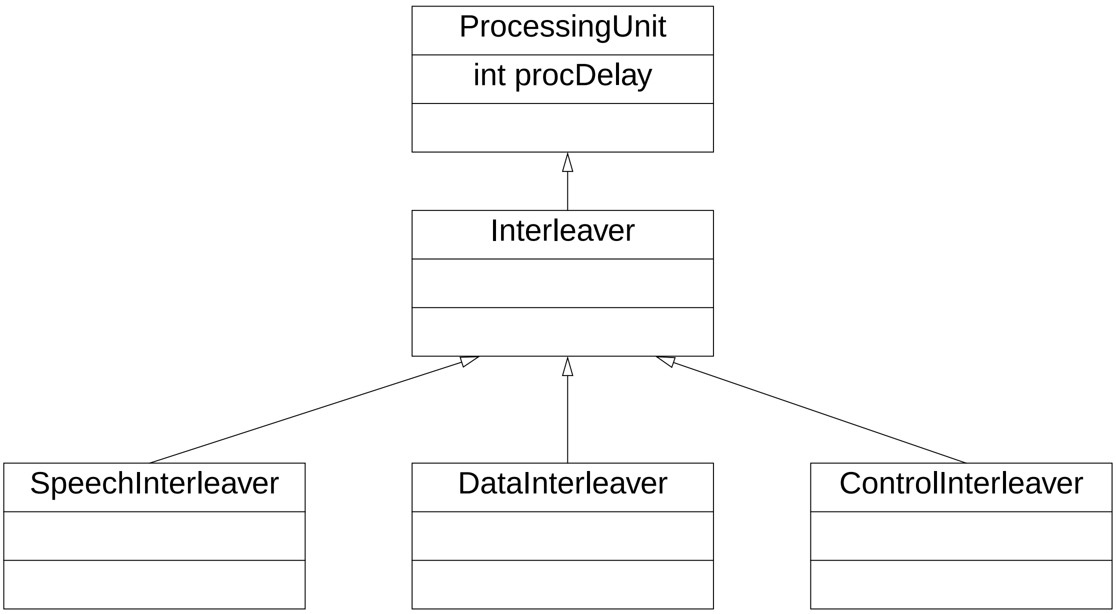
Main Packages of the Model

- FunctionalUnits
- BasebandCtrl
- TRX
- PIs
- Abis
- Radio
- Globals
- Shell

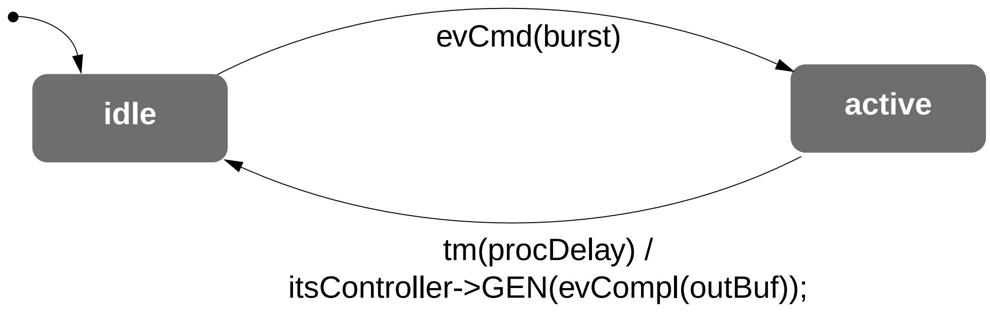
The *FunctionalUnits* Package

- operates on the data path
- the DSP algorithms are specified in C++
- no control functions (very primitive statechart)
- simple interface (**in** command, **out** notification) ⇒ event-driven
- instrumented for timed simulation
- examples of units:
  - CRC encoders, CRC decoders
  - burst interleavers, burst deinterleavers
  - convolutional encoders, convolutional decoders

FunctionalUnits Class Diagram



FunctionalUnits Behaviour

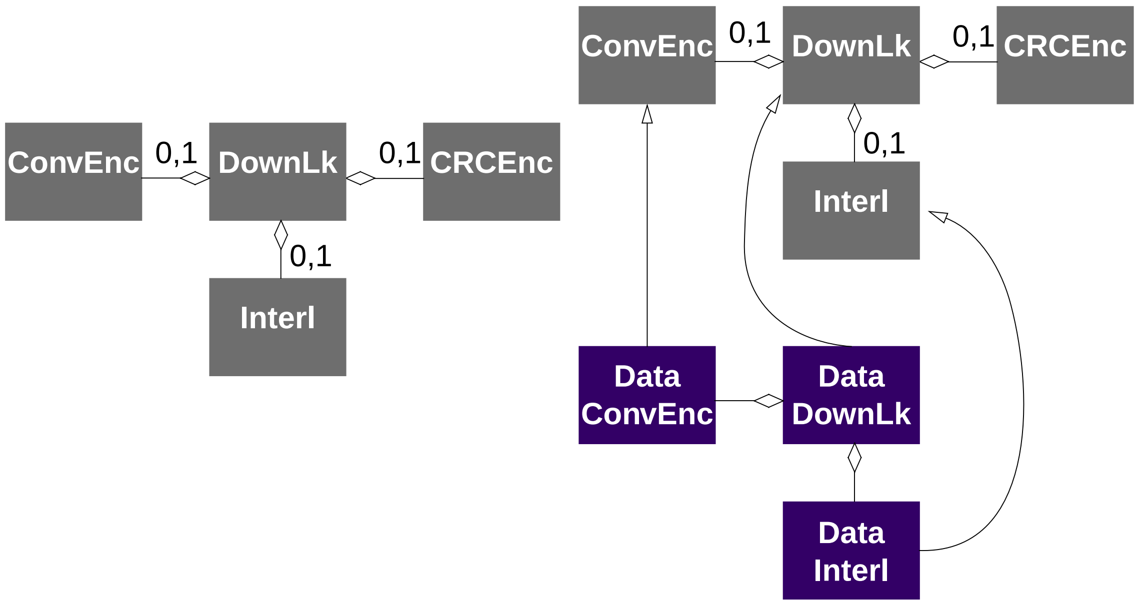



```
for (k = 0; k < NORMAL_BLOCK; k++) {
    val = block->getBitAt(k);
    b = ((4 * currBlock + (k % 8)) % 8);
    j = 2 * ((49 * k) % 57) + (k % 8) / 4;
    outBuf[b]->setBitAt(j, val);
}
```

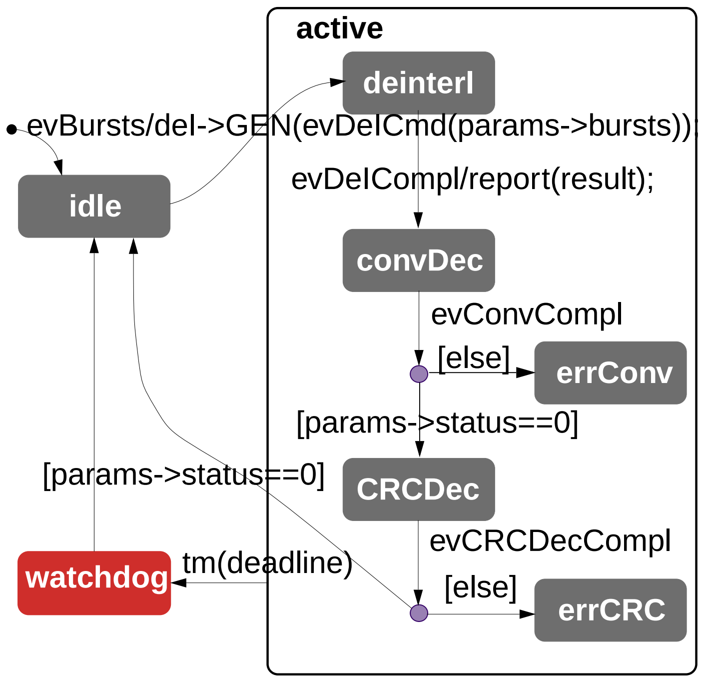

The *BasebandCtrl* Package

- control oriented
- instrumented for timed simulation (execution deadlines of managed functional units)
- example of baseband controllers:
  - SpeechDownLkCtrl
  - DataUpLkCtrl
  - CtrlDownLkCtrl

BasebandCtrl **Class Diagram**



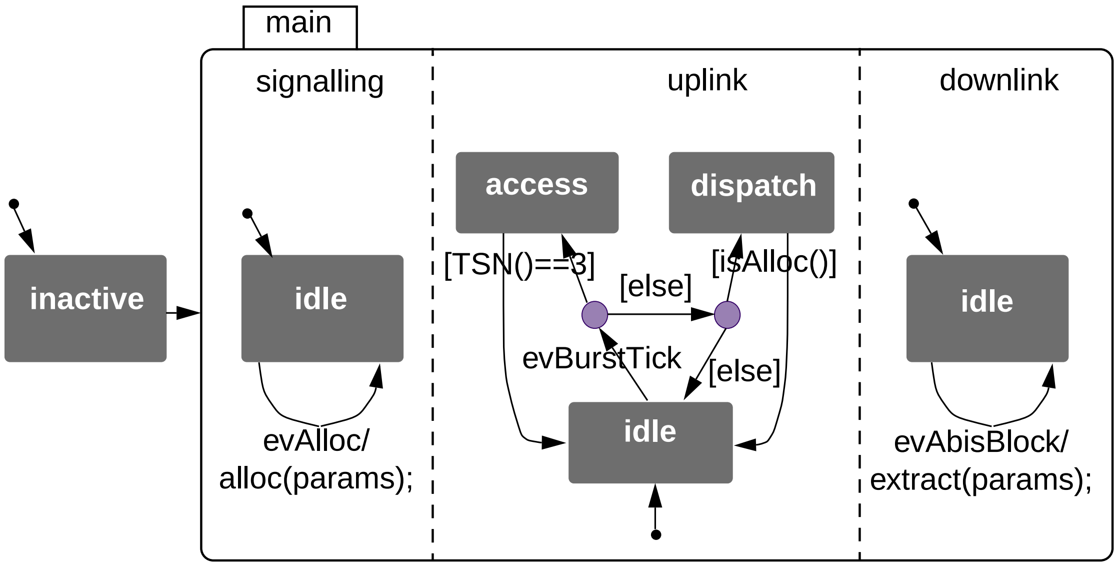
BasebandCtrl **Statechart**



The *TRX* Package

- control oriented
- controls signalling, uplink and downlink transmission
- allocated and deallocated channels
- “database” (keeps track of allocated channels and their types)
- time-triggered

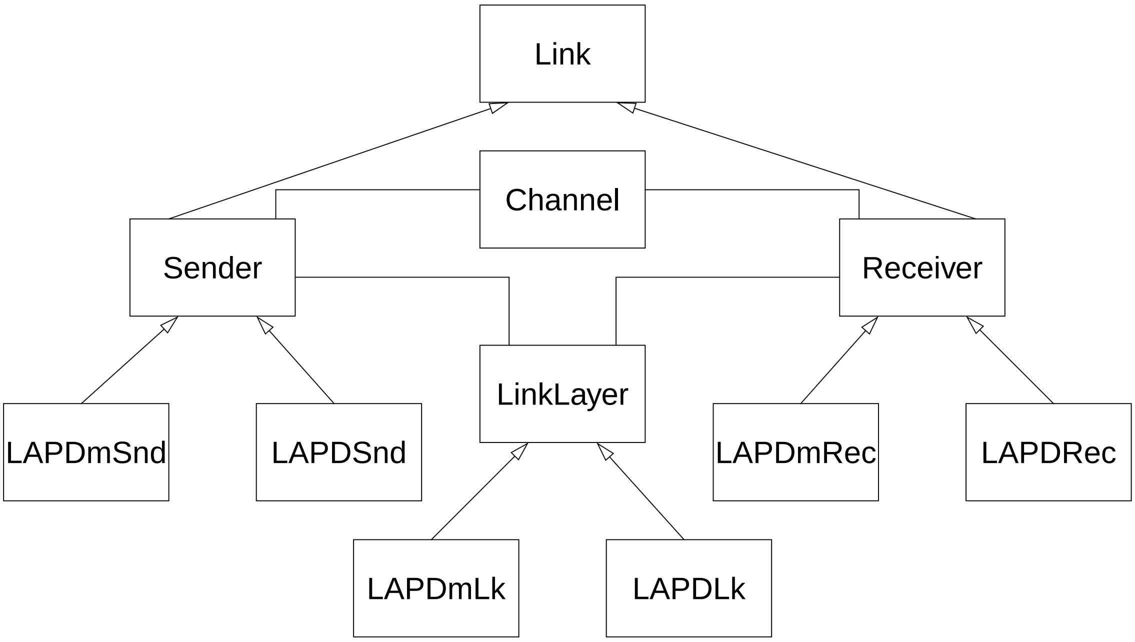
The TRX Statechart



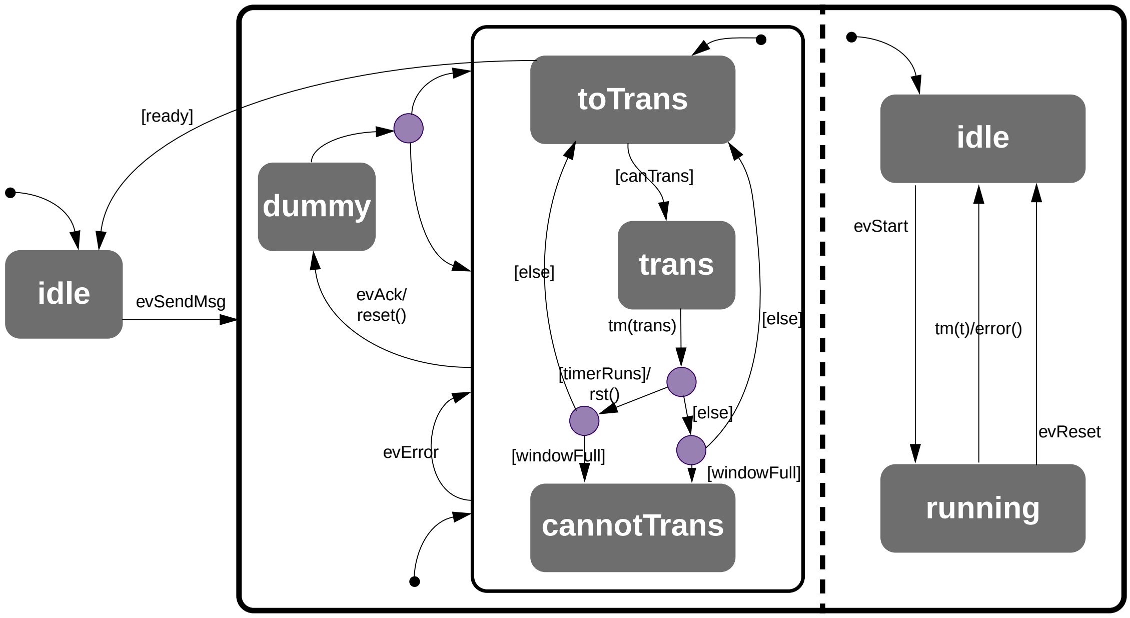
The *Pls* Package

- protocol processing on the datalink layer
- aspects:
  - sliding window and transmit–wait–acknowledge concepts
  - segmenting and reassembling ⇒ packets may arrive out of order
  - channel loss

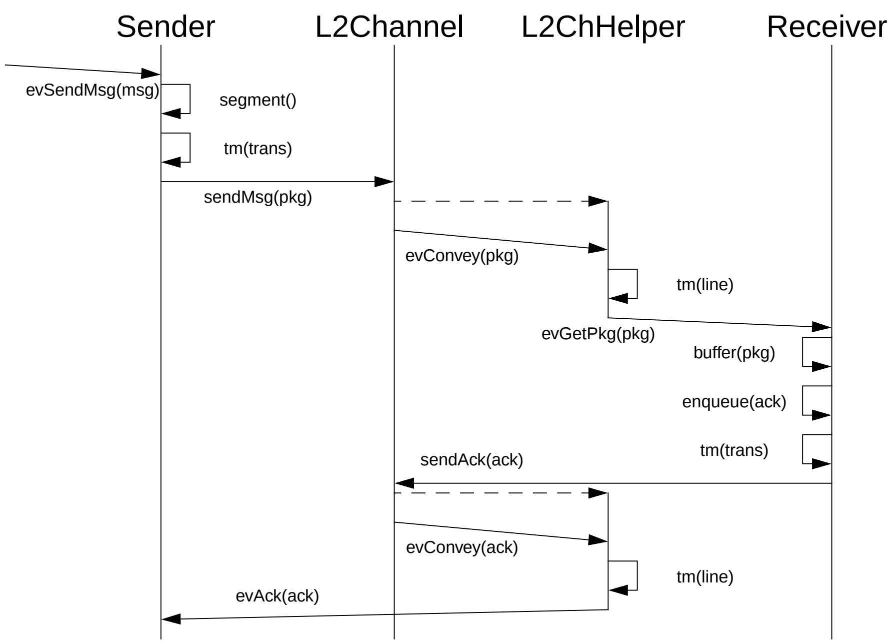
Pls Class Diagram



Sender Statechart



Layer 2 Message Transmission



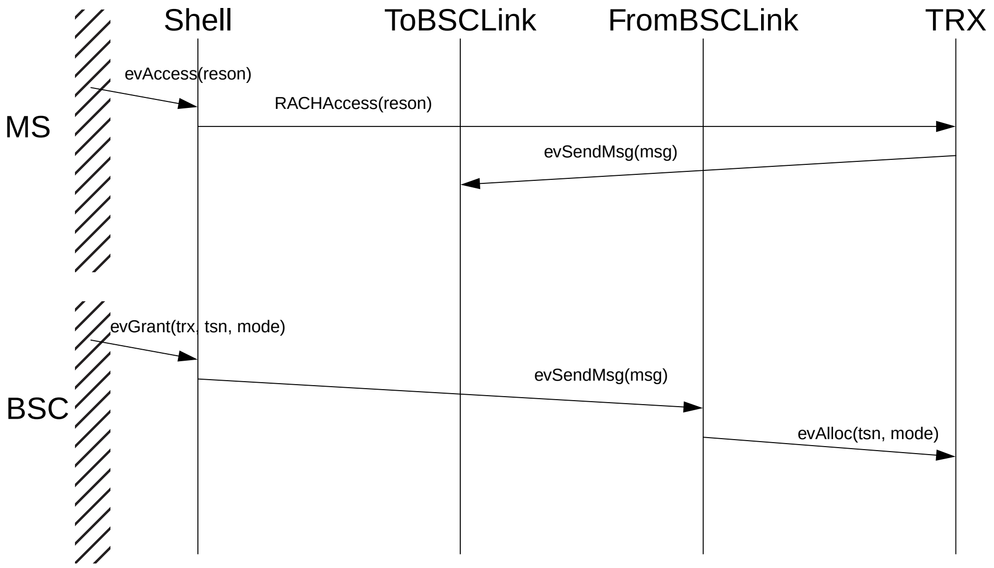
The *Globals* Package

- contains various utility classes and functions
- examples:
  - classes: BitAddressableBuffer, Queue, BTSConfig (singleton), GlobalClock (singleton)
  - functions: getTSN(), getFrame()
- *GlobalClock* implements the subscriber pattern

The *Shell* Package

- for interfacing the designer with the model
- provides a means to feed “coarse-grained” events from the outside (MS or BSC)

Channel Allocation



Rhapsody's Simulation Engine

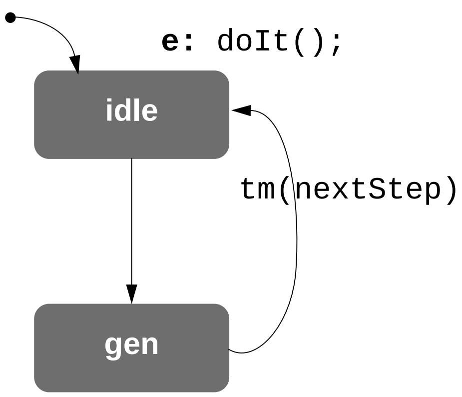
- the objects of the model are clustered in one or several OS threads (under the user's control)
- an event queue and a timeout pool are associated to every thread
- the time is advanced to the value of next due timeout when all threads are idle, that is, when
  - the objects wait for timeouts to be due, or
  - the event queue is empty
- we observed that the result of the timed simulation depends on how the objects are grouped in threads ⇒ probably because of the OS the simulator relies upon

Simulation

- implied a small modification of the tool simulation engine in order to have access to its clock
- a list of events to be fed to the model can be specified in a file ⇒ possibility of specifying test scenarios
- for telecom applications automatic generation of large set of events with various probability distributions would be helpful ⇒ an idea would be to define an test scenario specification language
- impossibility at the moment to feed events to the system at explicit moment in time ⇒ we adopted two approaches

Feeding Test Vectors to the Model (1)

- "wrap" the model in an "eventGenerator"




```
switch (time) {
...
case 48000:
 sprintf(msgBuf, "Car passes on
            track 2 at %d", t);
 cout << msgBuf << endl;
 insertMessage(msgBuf);
 itsSensor[5]->GEN(evCar());
 nextStep = 1000;
 break;
case 49000:
 sprintf(msgBuf, "Car passes on
            track 2 at %d", t);
...
}
```

Feeding Test Vectors to the Model (2)

- GUI that allows feeding of events $\Rightarrow$ dynamic scenario generation
- applicable when
  - coarse-grained events (e.g. *evAllocate*, *evGrant*)
  - not necessarily interested when an input event arrives but rather in the response time

Conclusions

- an industrial-scale application has been modelled in UML
- it comprised heterogeneous aspects: protocol processing, DSP, control oriented, real-time requirements
- UML leverages the quality of the specification and the ease of understanding by the visualization of the various relationships it provides
- the timeout transition is currently the only means for dealing with real-time aspects
- manual instrumentation is required for correctness checking
- software or hardware generation from UML is equivalent to synthesis from statecharts or from the language of the imported/generated code
- limited possibility for architecture exploration