Towards A Full Semantics for Real-Time VDM++

(working title – *Revision* : 1.1 – April 4, 2006)

Marcel Verhoef [1], Peter Gorm Larsen [2], and Jozef Hooman [3]

[1] Chess Information Technology B.V., P.O. Box 5021, 2000 CA Haarlem, The Netherlands.*
http://www.chess.nl email: `Marcel.Verhoef@chess.nl`
[2] University College of Aarhus, Dalgas Avenue 2, DK-8000 Århus C, Denmark.
http://www.iha.dk email: `pgl@iha.dk`
[3] Embedded Systems Institute, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands.*
http://www.esi.nl email: `Jozef.Hooman@esi.nl`

Abstract. Currently, this is very much a working document, collecting thoughts and ideas. The aim is to write a technical report and derive a paper from the technical report. Publication target is FM'06 (*http://fm06.mcmaster.ca*), submission date is 24 February 2006.

1 The "Grand Idea"

Here a list of thoughts and ideas that would probably take 50 years of research to complete. Obviously we need to scale down the ambition level but I personally like to reason from the "big picture" first, to get the story straight.

1. Use Ed Lee's IEEE Computer column [1] to introduce the problem of embedded software development.
2. Introduce notion of system engineering and (lack) of inter discipline communication in practice.
3. Our grand idea: deliver a solution for engineering of embedded systems from UML model to code and across all involved disciplines. Many partial solutions exist already, this work should provide the "super glue" to stick everything together.
4. Primary idea is to provide a suitable notation for specifying timing properties of software. In our opinion, it's the core part that's missing.
5. The notation shall be easy to use and understand by software engineers working in an industrial environment.
6. Break the typical abstraction barriers that make practical analysis difficult, introduce the notion of "deployment", "scheduling" and "communication" at the model level.
7. Use VDM++ notation as a starting point. Extend the semantics to include proper notion of time, preferably an operational semantics.
8. Show that useful analysis can still be performed (actually: can **only** be performed if you do it like this) although we have broken typical abstraction barriers.
9. Explore possibility to extract models from the VDM++ specification to do exhaustive analysis (using model checkers or proof tools).
10. Explore possibility to integrate models with typical control engineering simulators to explore dynamic system properties by simulation. Likely candidates are Ptolemy (which includes Giotto), TrueTime (which is based on Matlab/Simulink).
11. Explore how the model can be used as a basis for synthesis (e.g. code generation).
12. buzzword compliance: model driven architecture, model based engineering.

---

* and Radboud University Nijmegen, Institute for Computer and Information Sciences, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands. *http://www.cs.ru.nl/ita/*. This work has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project was partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

2 Inventory of related work

1. The Parallel Object-Oriented Specification Language (*http://www.es.ele.tue.nl/poosl/*). They already go a long, long way. Why do we need VDM++ still? Is VDM++ closer to the notations currently used by software engineers?
2. Giotto (*http://www-cad.eecs.berkeley.edu/~fresco/giotto/*).
3. Kopetz work on the Time-Triggered Architecture (*http://www.ttagroup.org*).
4. For analysis of timed systems, UPPAAL (*http://www.uppaal.com*) and for analysis of scheduling problems (of uni-processor systems), TIMESTOOL (*http://www.timestool.org*). But there are many, many more.
5. Work performed at Mälardalen University in Sweden (see ISOLA proceedings and their web site *http://www.mrtc.mdh.se/*).
6. Ptolemy, (*http://ptolemy.eecs.berkeley.edu/*).
7. OMEGA, (*http://www-omega.imag.fr/*) and in particular the recent papers co-authored by Jozef (obviously :-).
8. The survey paper by Wang [2] gives an extensive overview of languages for specifying and analysis of timed systems.
9. The recent work by Lamport, "Real-time is Really Simple" (see *http://www.lamport.org*).
10. The work on TrueTime at KTH's control engineering laboratory (see the web site of Dan Hendrikssen at *http://www.control.lth.se/~dan/truetime/*).
11. The work performed by Laci Posta and Natalya Mulyar at ESI, integrating Matlab/Simulink with Rational Rose Real-Time.
12. Obviously there is the work of Mok on *Real-time Logic* and the work of Hansen and Chaochen on *Duration Calculus*. A nice overview is actually provided in [3].
13. Do not forget: architecture definition languages such as Koala (Philips), AADL (SEI, see *http://www.aadl.info/*) and so on (i.e. Garlan stuff from the Urbino summerschool).
14. I have the feeling that I've barely scratched the surface!

3 Notes from the brainstorm session with Evert

The following list was composed after a brainstorm meeting with Evert (who is not a VDM++ expert, neither does he have practical experience with VDMTools). So, many things are already available in some form or other.

1. The semantics of the specification language should be close to (or mapped onto) the semantics of industry accepted implementation languages and operating systems, such as for example C, C++ and VxWorks.
2. Support for multi-CPU systems development where the CPU's are interconnected through a communication bus, such as CAN, TCP/IP etcetera.
3. Estimation of the CPU time of a language construct directly from the complexity of the statement.
4. The specification language shall be able to specify deadlines
5. The tools shall be able to determine whether these deadlines can be met and if not, under which circumstances.
6. Support for round-robin and priority based preemptive scheduling.
7. Support analysis of deadlock, livelock, starvation, race conditions
8. Support analysis of schedulability
9. Dealing with uncertainty (probabilistic behavior, specify jitter as a probabilistic variable?)
10. Checking of constraints and invariants
11. Integration with continuous time simulation. Bridge the discrete event / continuous time hurdle
12. Good visualization possibilities, such as sequence diagrams and Gantt charts, preferably in parallel with the continuous variables.

4 A few observations of my own

Here are few observations that should be seen in addition to the "On the Use of VDM++ for Specifying Real-time Systems" paper.

1. Primary concern is to extend the semantics and the interpreter to deal with multi-processing. So not only pseudo-parallel execution, but also truly parallel execution of tasks. This is needed, for example, to specify the behavior of the environment independently from the system. This is an absolute must for embedded systems.
2. It shall then be possible to reason about deployment, which active object is executed on what computing resource. Obviously, the scheduling parameters of this computing resource should be specified, not as a simulation parameter as it is now, but as a property of the computation resource.
3. The notion of interrupts needs to be integrated into the language; these concepts are common in embedded systems, but are seldom supported at the language level. I believe that much can be gained here.
4. Similarly, it shall be possible to explicitly specify the hardware/software interface, in particular interfaces to communication resources (and the interconnections between the computation components, just like in SystemC and ROOM for example, where channels are used for that).
5. Basically, the HW/SW interface and the inner workings of the operating system (or scheduler) should become part of the system model. Only then can we reason about timing, response times etc at the system level. Interestingly, Lamport provides these kinds of definitions in his TLA+ book...
6. Simulation is the main modus operandi of the current VDMTools implementation and I think we should keep that intact, including the round-trip engineering philosophy (going from VDM++ to UML and back). Maybe we should consider SysML?
7. We should consider integrating with other simulation platforms rather than improving the VDMTools interpreter. Similarly, figure out whether we can derive abstract specifications from our more detailed models that can be fed into powerful analysis tools such as Uppaal, Times, SMV and so on.

5 Ideas for case studies

1. The trip sensor specification from the "Timing Tolerances in Safety-Critical Software" paper by Alan Wassyng et al (see FM'05 proceedings).
2. The production cell case study of Lewerentz and Lindner (see also LNCS volume 891, 1995)
3. The "Countermeasures" example from the VICE project (VDM++ In a Constraint Environment)
4. The NASA Safer backpack, which is well-published and covers all domains
5. The railway crossing used by Jinfeng Huang (from the POOSL group of Jeroen Voeten)
6. (parts of) Jan Beckers paper path simulation model, nice direct link to Boderc but a potential publication nightmare wrt confidentiality.
7. The in-car radio navigation case study of Marcel (as described in his ISOLA/STTT paper), also fits the general theme of Marcel's PhD thesis.
8. The "Timed Philosophers" example from the Overture workshop paper. Is interesting because it already includes the notational extensions we want to investigate.

6 Outline work plan

1. Describe the current semantics of VDM++ using VDM++ in an operational (executable) style. Use [4] and in particular [3] as our starting point. Use the current VDMTools implementation to test and prototype the operational semantics. Visualization can be taken care of by the *ShowVice* tool that I already have build for [5], possibly slightly modified. The workflow is described in Figure 1.

2. Extend the semantics as proposed in [5].
3. Take an interesting case study, describe it using the "new" extended semantics
4. What can we analyze better now than what we did before?
5. Can we now create simulation models directly from our semantics models?

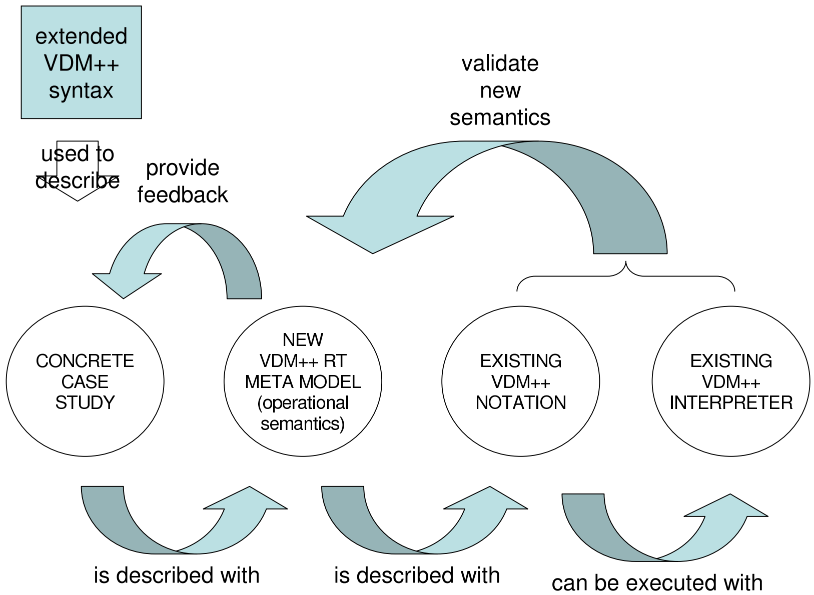

Fig. 1. The proposed workflow for this work

References

1. Lee, E.A.: Absolutely positively on time: What would it take? IEEE Computer (2005) 85–87
2. Wang, F.: Formal verification of timed systems: A survey and perspective. Proc. of the IEEE **92** (2004) 1283–1305
3. Mukherjee, P., Bousquet, F., Delabre, J., Paynter, S., Larsen, P.G.: Exploring timing properties using VDM++. (1999)
4. Lano, K.: Logic specification of reactive and real-time systems. Journal of Logic and Computation **8** (1998) 679–711
5. Verhoef, M.: On the use of VDM++ for specifying real-time systems. Proc. First Overture workshop (2005)
6. Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object–oriented Systems. Springer, New York (2005)
7. Douglas, B.P.: Real Time UML – Advances in the UML for real-time systems. Third edn. Addison-Wesley (2004)
8. Lamport, L.: Real time is really simple. Technical Report MSR-TR-2005-30, Microsoft Research (2005)
9. Lamport, L.: Real-time model checking is really simple. In: Proceedings of Charme 2005, Springer (2005)
10. Axlog: Avionics architecture description language. *http://www.axlog.fr/R_d/aadl/seminar.html* (2002) Seminar hand outs.
11. Hendriksson, D., Cervin, A.: TrueTime 1.13 – Reference Manual. Lund Institute of Technology – Department of Automatic Control. (2003)

Acknowledgments

The authors wish to thank Evert van de Waal for providing useful comments to early drafts of this document.

7 In-car Radio Navigation case-study

7.1 Event classes

```
class Event

instance variables
  val : nat

operations
  public Event: nat ==> Event
  Event (pv) == val := pv;

  public getEvent: () ==> nat
  getEvent () == return val

end Event

class InterruptEvent
  is subclass of Event

operations
  public InterruptEvent: nat ==> InterruptEvent
  InterruptEvent (pne) == Event(pne)

end InterruptEvent

class NetworkEvent
  is subclass of Event

operations
  public NetworkEvent: nat ==> NetworkEvent
  NetworkEvent (pne) == Event(pne)

end NetworkEvent
```

7.2 Task classes

```
class AbstractTask

instance variables
  name : seq of char := [];
  events : seq of NetworkEvent := [];
  interrupts : seq of InterruptEvent := [];
  dispatcher : EventDispatcher

operations
  public AbstractTask: seq of char * EventDispatcher ==> AbstractTask
  AbstractTask (pnm, ped) == atomic ( name := pnm; dispatcher := ped; );

  public getName: () ==> seq of char
  getName () == return name;

  public setEvent: Event ==> ()
  setEvent (pe) ==
```

```
    if isofclass(NetworkEvent,pe)
    then events := events ^ [pe]
    else interrupts := interrupts ^ [pe];

  protected getEvent: () ==> Event
  getEvent () ==
    if len interrupts > 0
    then ( dcl res: Event := hd interrupts;
           interrupts := tl interrupts;
           return res )
    else ( dcl res: Event := hd events;
           events := tl events;
           return res );

  protected handleEvent: Event ==> ()
  handleEvent (-) == is subclass responsibility;

  protected sendMessage: seq of char * nat ==> ()
  sendMessage (pnm, pid) == dispatcher.SendNetwork(getName(), pnm, pid);

  protected raiseInterrupt: seq of char * nat ==> ()
  raiseInterrupt (pnm, pid) == dispatcher.SendInterrupt(getName(), pnm, pid)

sync
  mutex (setEvent, getEvent);
  per getEvent => len events > 0 or len interrupts > 0

end AbstractTask

class BasicTask
  is subclass of AbstractTask

operations
  public BasicTask: seq of char * EventDispatcher ==> BasicTask
  BasicTask (pnm, ped) == AbstractTask(pnm, ped);

thread
  while (true) do
    handleEvent(getEvent())

end BasicTask

class EnvironmentTask
  is subclass of AbstractTask

instance variables
  -- unique identifier for each generated event
  static num : nat := 0;

  -- administration for the event traces
  protected outl : map nat to nat := {|->};
  protected inl : map nat to nat := {|->}

operations
```

```
  public getNum: () ==> nat
  getNum () ==
    ( dcl res : nat := num;
      num := num + 1;
      return res );

  public Run: () ==> ()
  Run () == is subclass responsibility;

  public updateTime: nat ==> ()
  -- updateTime (delta) == mtime := mtime + delta;
  updateTime (delta) == skip;

  public logOutEvent: nat ==> ()
  logOutEvent (pev) == outl := outl munion {pev |-> time};

  public logInEvent: nat ==> ()
  logInEvent (pev) == inl := inl munion {pev |-> time}

sync
  mutex (getNum);

end EnvironmentTask
```

7.3 Application tasks

```
class MMIHandleKeyPressOne
  is subclass of BasicTask

operations
  public MMIHandleKeyPressOne: EventDispatcher ==> MMIHandleKeyPressOne
  MMIHandleKeyPressOne (pde) ==  BasicTask("HandleKeyPress",pde);

  public HandleKeyPress: () ==> ()
  HandleKeyPress () == duration (100) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( HandleKeyPress();
      sendMessage("AdjustVolume", pe.getEvent()) )

end MMIHandleKeyPressOne

class MMIHandleKeyPressTwo
  is subclass of BasicTask

operations
  public MMIHandleKeyPressTwo: EventDispatcher ==> MMIHandleKeyPressTwo
  MMIHandleKeyPressTwo (pde) == BasicTask("HandleKeyPress",pde);

  public HandleKeyPress: () ==> ()
  HandleKeyPress () == duration (100) skip;

  handleEvent: Event ==> ()
```

```
  handleEvent (pe) ==
    ( HandleKeyPress();
      sendMessage("DatabaseLookup", pe.getEvent()) )

end MMIHandleKeyPressTwo

class MMIUpdateScreen
  is subclass of BasicTask

operations
  public MMIUpdateScreen: EventDispatcher ==> MMIUpdateScreen
  MMIUpdateScreen (pde) == BasicTask("UpdateScreen",pde);

  public UpdateScreen: () ==> ()
  UpdateScreen () == duration (500) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( UpdateScreen();
      raiseInterrupt("VolumeKnob", pe.getEvent()) )

end MMIUpdateScreen

class RadioAdjustVolume
  is subclass of BasicTask

operations
  public RadioAdjustVolume: EventDispatcher ==> RadioAdjustVolume
  RadioAdjustVolume (pde) == BasicTask("AdjustVolume",pde);

  public AdjustVolume: () ==> ()
  AdjustVolume () == duration (100) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( AdjustVolume();
      sendMessage("UpdateScreen", pe.getEvent()) )

end RadioAdjustVolume

class RadioHandleTMC
  is subclass of BasicTask

operations
  public RadioHandleTMC: EventDispatcher ==> RadioHandleTMC
  RadioHandleTMC (pde) == BasicTask("HandleTMC",pde);

  public HandleTMC: () ==> ()
  HandleTMC () == duration (1000) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( HandleTMC();
      sendMessage("DecodeTMC", pe.getEvent()) )
```

```
end RadioHandleTMC

class NavigationDatabaseLookup
  is subclass of BasicTask

operations
  public NavigationDatabaseLookup: EventDispatcher ==> NavigationDatabaseLookup
  NavigationDatabaseLookup (pde) == BasicTask("DatabaseLookup",pde);

  public DatabaseLookup: () ==> ()
  DatabaseLookup() == duration (5000) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( DatabaseLookup();
      sendMessage("UpdateScreen", pe.getEvent()) )

end NavigationDatabaseLookup

class NavigationDecodeTMC
  is subclass of BasicTask

operations
  public NavigationDecodeTMC: EventDispatcher ==> NavigationDecodeTMC
  NavigationDecodeTMC (pde) == BasicTask("DecodeTMC",pde);

  public DecodeTMC: () ==> ()
  DecodeTMC () == duration (5000) skip;

  handleEvent: Event ==> ()
  handleEvent (pe) ==
    ( DecodeTMC();
      sendMessage("UpdateScreen", pe.getEvent()) )

end NavigationDecodeTMC
```

7.4 Environment tasks

```
class VolumeKnob
  is subclass of EnvironmentTask

operations
  public VolumeKnob: EventDispatcher ==> VolumeKnob
  VolumeKnob (ped) == AbstractTask("VolumeKnob",ped);

  handleEvent: Event ==> ()
  handleEvent (pev) == duration (0) logInEvent(pev.getEvent())
  post forall pr in set dom inl &
         exists1 ps in set dom outl &
           pr = ps => outl(ps) - inl(pr) <= 1500;

  createSignal: () ==> ()
  createSignal () ==
    duration (0)
```

```
    ( dcl num : nat := getNum();
      updateTime(1000);
      logOutEvent(num);
      raiseInterrupt("HandleKeyPress", num) );

  public Run: () ==> ()
  Run () == start(self)

thread
  periodic (1000) (createSignal)

end VolumeKnob

class InsertAddress
  is subclass of EnvironmentTask

operations
  public InsertAddress: EventDispatcher ==> InsertAddress
  InsertAddress (ped) == AbstractTask("InsertAddress",ped);

  handleEvent: Event ==> ()
  handleEvent (pev) == duration (0) logInEvent(pev.getEvent())
  post forall pr in set dom inl &
         exists1 ps in set dom outl &
           pr = ps => outl(ps) - inl(pr) <= 2000;

  createSignal: () ==> ()
  createSignal () ==
    duration (0)
      ( dcl num : nat := getNum();
        updateTime(1000);
        logOutEvent(num);
        raiseInterrupt("HandleKeyPress", num) );

  public Run: () ==> ()
  Run () == start(self)

thread
  periodic (1000) (createSignal)

end InsertAddress

class TransmitTMC
  is subclass of EnvironmentTask

operations
  public TransmitTMC: EventDispatcher ==> TransmitTMC
  TransmitTMC (ped) == AbstractTask("TransmitTMC", ped);

  handleEvent: Event ==> ()
  handleEvent (pev) == duration (0) logInEvent(pev.getEvent())
  post forall pr in set dom inl &
         exists1 ps in set dom outl &
           pr = ps => outl(ps) - inl(pr) <= 100000;
```

```
  createSignal: () ==> ()
  createSignal () ==
    duration (0)
      ( dcl num : nat := getNum();
        updateTime(1000);
        logOutEvent(num);
        raiseInterrupt("HandleTMC", num) );

  public Run: () ==> ()
  Run () == start(self)

thread
  periodic (1000) (createSignal)

end TransmitTMC

class Logger

types
  string = seq of char

instance variables
  static io : IO := new IO();
  static num : nat := 0;

operations
  public printNetworkEvent: seq of char * seq of char * nat ==> ()
  printNetworkEvent (psrc, pdest, pid) ==
    ( def - = io.writeval[seq of (seq of char | nat)]
        (["network", psrc, pdest, pid, time]) in num := num + 1;
      def - = io.fwriteval[seq of (seq of char | nat)]
        ("mytrace.txt", ["network", psrc, pdest, pid, time], <append>) in skip );

  public printInterruptEvent: seq of char * seq of char * nat ==> ()
  printInterruptEvent (psrc, pdest, pid) ==
    ( def - = io.writeval[seq of (seq of char | nat)]
        (["interrupt", psrc, pdest, pid, time]) in num := num + 1;
      def - = io.fwriteval[seq of (seq of char | nat)]
        ("mytrace.txt", ["interrupt", psrc, pdest, pid, time], <append>) in skip );

  static public wait: () ==> ()
  wait () == skip

sync
  per wait => num > 30

end Logger
```

7.5 Event dispatching

```
class AbstractTaskEvent

instance variables
```

```
  public abstask: AbstractTask;
  public ev : Event

operations
  public AbstractTaskEvent: AbstractTask * Event ==> AbstractTaskEvent
  AbstractTaskEvent (pat, pev) == (abstask := pat; ev := pev);

  public getFields: () ==> AbstractTask * Event
  getFields () == return mk_ (abstask, ev)

end AbstractTaskEvent

class EventDispatcher
  is subclass of Logger

instance variables
  queues : map seq of char to AbstractTask := {|->};
  -- messages : seq of (AbstractTask * Event) := [];
  messages : seq of AbstractTaskEvent := [];
  interrupts: seq of AbstractTaskEvent := []
  -- interrupts: seq of (AbstractTask * Event) := []

operations
  public Register: AbstractTask ==> ()
  Register (pat) ==
    queues := queues munion { pat.getName() |-> pat }
    pre pat.getName() not in set dom queues;

  setEvent: AbstractTask * Event ==> ()
  setEvent (pat, pe) ==
    if isofclass(NetworkEvent,pe)
    -- then messages := messages ^ [mk_(pat,pe)]
    then messages := messages ^ [new AbstractTaskEvent(pat,pe)]
    else interrupts := interrupts ^ [new AbstractTaskEvent(pat,pe)];
    -- else interrupts := interrupts ^ [mk_(pat,pe)];

  getEvent: () ==> AbstractTask * Event
  getEvent () ==
    if len interrupts > 0
    -- then ( dcl res : AbstractTask * Event := hd interrupts;
    --        interrupts := tl interrupts;
    --        return res )
    then ( dcl res : AbstractTaskEvent := hd interrupts;
           interrupts := tl interrupts;
           return res.getFields() )
    -- else ( dcl res : AbstractTask * Event := hd messages;
    --        messages := tl messages;
    --        return res );
    else ( dcl res : AbstractTaskEvent := hd messages;
           messages := tl messages;
           return res.getFields() );

  public SendNetwork: seq of char * seq of char * nat ==> ()
  SendNetwork (psrc, pdest, pid) ==
```

```
    duration (0)
     ( dcl pbt: BasicTask := queues(pdest);
       printNetworkEvent(psrc, pdest, pid);
       setEvent(pbt, new NetworkEvent(pid)) )
    pre pdest in set dom queues;

  public SendInterrupt: seq of char * seq of char * nat ==> ()
  SendInterrupt (psrc, pdest, pid) ==
    duration (0)
     ( dcl pbt: BasicTask := queues(pdest);
       printInterruptEvent(psrc, pdest, pid);
       setEvent(pbt, new InterruptEvent(pid)) )
    pre pdest in set dom queues;

  handleEvent: AbstractTask * Event ==> ()
  handleEvent (pat, pe) == pat.setEvent(pe)

thread
  duration (0)
    while (true) do
      def mk_ (pat,pe) = getEvent() in
        handleEvent(pat,pe)

sync
  mutex(setEvent, getEvent);
  per getEvent => len messages > 0 or len interrupts > 0

end EventDispatcher
```

7.6 The RadNav system – top-level specification

```
class RadNavSys

instance variables
  dispatch : EventDispatcher := new EventDispatcher();
  appTasks : set of BasicTask := {};
  mode : nat

operations
  RadNavSys: nat ==> RadNavSys
  RadNavSys (pi) ==
   ( mode := pi;
     cases (mode) :
       1 -> ( addApplicationTask(new MMIHandleKeyPressOne(dispatch));
              addApplicationTask(new RadioAdjustVolume(dispatch));
              addApplicationTask(new MMIUpdateScreen(dispatch));
              addApplicationTask(new RadioHandleTMC(dispatch));
              addApplicationTask(new NavigationDecodeTMC(dispatch)) ),
       2 -> ( addApplicationTask(new MMIHandleKeyPressTwo(dispatch));
              addApplicationTask(new NavigationDatabaseLookup(dispatch));
              addApplicationTask(new MMIUpdateScreen(dispatch));
              addApplicationTask(new RadioHandleTMC(dispatch));
                   addApplicationTask(new NavigationDecodeTMC(dispatch)) )
     end;
```

```
    startlist(appTasks); start(dispatch) )
  pre pi in set {1, 2};

 addApplicationTask: BasicTask ==> ()
 addApplicationTask (pbt) ==
  ( appTasks := appTasks union {pbt};
    dispatch.Register(pbt) );

 addEnvironmentTask: EnvironmentTask ==> ()
 addEnvironmentTask (pet) ==
  ( dispatch.Register(pet);
    pet.Run() );

 public Run: () ==> ()
 Run () ==
  ( cases (mode):
      1 -> ( addEnvironmentTask(new VolumeKnob(dispatch));
             addEnvironmentTask(new TransmitTMC(dispatch)) ),
      2 -> ( addEnvironmentTask(new InsertAddress(dispatch));
             addEnvironmentTask(new TransmitTMC(dispatch)) )
    end;
    Logger`wait() )

end RadNavSys
```

8 The new stuff

```
system RadNavSys

instance variables
  -- create an MMI class instance
  static public mmi : MMI := new MMI();
  -- define the first CPU with fixed priority scheduling and 22E6 MIPS performance
  CPU1 : CPU := new CPU (CPU‘FP, 22E6);

  -- create an Radio class instance
  static public radio : Radio := new Radio();
  -- define the second CPU with fixed priority scheduling and 11E6 MIPS performance
  CPU2 : CPU := new CPU (CPU‘FP, 11E6);

  -- create an Navigation class instance
  static public navigation : Navigation := new Navigation();
  -- define the third CPU with fixed priority scheduling and 113 MIPS performance
  CPU3 : CPU := new CPU (CPU‘FP, 113E6);

  -- create a communication bus that links the three CPU’s together
  BUS1 : BUS := new BUS (BUS‘CSMACD, 72E3, CPU1, CPU2, CPU3)

operations
  public RadNavSys: () ==> ()
  RadNavSys ()
    ( -- deploy mmi on CPU1
      CPU1.deploy(mmi);
      CPU1.setPriority(HandleKeyPress,100);
      CPU1.setPriority(UpdateScreen,90);
      -- deploy radio on CPU2
      CPU2.deploy(radio);
      CPU2.setPriority(AdjustVolume,100);
      CPU2.setPriority(DecodeTMC,90);
      -- deploy navigation on CPU3
      CPU3.deploy(navigation);
      CPU3.setPriority(DatabaseLookup, 100);
      CPU3.setPriority(DecodeTMC, 90)
      -- starting the CPUs and BUS is implicit )

  static public wait: () ==> ()
  wait () == skip;

  sync
    per wait => mmi.cnt > 30

end RadNavSys

class MMI

instance variables
  public cnt : nat := 0

operations
```

```
  async public HandleKeyPress: nat ==> ()
  HandleKeyPress (pn) ==
    ( duration (1E5) cnt := cnt + 1;
      cases (pn):
        1 -> RadNavSys`radio.AdjustVolume(),
        2 -> RadNavSys`navigation.DatabaseLookup()
      end );

  async public UpdateScreen: nat ==> ()
  UpdateScreen (-) ==
    duration (5E5) skip;

end MMI

class Radio

operations
  async public AdjustVolume: () ==> ()
  AdjustVolume () ==
    ( duration (1E5) skip;
      RadNavSys`mmi.UpdateScreen(1) );

  async public HandleTMC: () ==> ()
  HandleTMC () ==
    ( duration (1E6) skip;
      RadNavSys`navigation.DecodeTMC() )

end Radio

class Navigation

operations
  async public DatabaseLookup: () ==> ()
  DatabaseLookup () ==
    ( duration (5E6) skip;
      RadNavSys`mmi.UpdateScreen(2) )

  async public DecodeTMC: () ==> ()
  DecodeTMC () ==
    ( duration (5E6) skip;
      RadNavSys`mmi.UpdateScreen(3) )

end Navigation

class VolumeKnob

thread
  periodic (p,j,d)
    duration (0)
      while (true)
        RadNavSys`mmi.HandleKeyPress(1)

end VolumeKnob

class InsertAddress
```

```
thread
  periodic (p,j,d)
    duration (0)
      while (true)
        RadNavSys`mmi.HandleKeyPress(2)

end InsertAddress

class TransmitTMC

thread
  periodic (p,j,d)
    duration (0)
      while (true)
        RadNavSys`radio.HandleTMC()

end TransmitTMC

class World

operations
  public RunScenario1 : () ==> ()
  RunScenario1 () ==
    ( start(new RadNavSys());
      startlist(new VolumeKnob(), new TransmitTMC());
      RadNavSys`wait() );

  public RunScenario1 : () ==> ()
  RunScenario1 () ==
    ( start(new RadNavSys());
      startlist(new InsertAddress(), new TransmitTMC());
      RadNavSys`wait() );

end World

new World().RunScenario1()
```