The egg and the hen
Or,
Which comes first:
the problem or the solution?

Tullio Vardanega

Outline of the talk

- Challenges of embedded real-time systems
  - Design for verifiability
  - Problem modelling, expressive power of the solution
  - Determinism versus inflexibility
- Desirable characters of the solution
  - Expressiveness, scalability, verifiability,
- The Ravenscar profile
  - Motivation, features, coverage

Challenges (1/6)

- Embedded real-time systems control and interact with a surrounding physical environment
  - Their interactive nature demands accurate modelling of the physical reality
  - Their real-time nature demands timeliness and responsiveness of control activities

Challenges (2/6)

- Timeliness
  - Control (by avoidance or minimisation) of release jitter
  - Assurance of completion within specified time bounds (deadline)
- Responsiveness
  - Minimisation of activation latency

Recall of basics

1. Activity becomes eligible for execution (ready)
2. Latest allowed completion (deadline)
3. Activation request
4. Impacted by activation latency
5. Ready period
6. Impacted by release jitter

Recall of basics

- Requests for activation can be periodic (regularly repeated time event) or aperiodic (irregular event)
  - Periodic activities need a reliable and accurate time reference
  - Aperiodic activities need characterisation of maximum frequency of arrival upon which they become sporadic
  - Both need low activation latency and controlled release jitter
Recall of basics

- The latency of activation is a function of the performance of the runtime scheduling mechanisms
  - The more elaborate, the greater the latency
    - Hence we prefer simple but not simplistic schedulers
- The release jitter is a function of the interference caused by other activities
  - Execution priority is the key to jitter control

Recall of basics

- Deadline as factor of utility function

Challenges (3/6)

- Embedded real-time systems model real-world entities, which are inherently concurrent
  - Multiple activation requests
    - Some fully independent of one another
  - Multiple sources
    - Time, external interrupts, software events
  - Diverse processing needs
    - Some require collaboration
      - Typically in a producer-consumer fashion

Challenges (4/6)

- To build embedded real-time systems we need:
  - Expressive means to accurately model the physical reality
  - Runtime mechanisms to ensure efficient and predictable implementation of concurrency
  - Analytical devices to assess the satisfaction of real-time requirements

Challenges (5/6)

- Accurate modelling of physical reality
  - We want a solution that fits the problem
    - Not a (degenerated) problem representation that fits a prefabricated solution
  - Efficient and predictable runtime
    - Not all problems allow all scheduling decisions to be made off line without losing value
      - The solution must warrant determinism (i.e., predictable behaviour)
      - The solution should not inflict inflexibility

Challenges (6/6)

- Static verification
  - To accept an implementation (design + code) we must be able to assess whether it meets the real-time requirements of the problem
    - We seek correctness by construction
  - We cannot afford to defer the assessment to the operation phase
    - Dynamic testing is best suited for functional requirements
    - Static analysis is far more practical and superior for real-time requirements
Desirable characters (1/6)

• **Expressive power (1/2)**
  - We should be able to:
    - Model concurrency with periodic and sporadic activities
    - Capture external (i.e., interrupt) and internal (i.e., software) events in addition to the passage of time
    - Support collaborative processing
      - Precedence of activation
      - (Data-oriented) synchronisation
    - Resource sharing
    - Assign cohesive functions to activities

Desirable characters (2/6)

• **Expressive power (2/2)**
  - We must ensure that:
    - The design determines the implementation
    - The implementation corresponds to the design, so that they can be consistently analysed
      - A powerful form of fault avoidance
  - We must enable:
    - Feedback from design to specification
    - Feedback from implementation to design and specification
      - Understanding and requirements evolve during development

Design vs. Implementation

- The concurrency of system components must be an explicit dimension of design
- The design method must offer a coherent set of abstractions and relations to represent concurrency
- The implementation language must offer a range of concurrency constructs that correspond, semantically, to those used for design

Design feedback

- The design and implementation dictionary, which captures
  - The real-time attributes and the activation characteristics of the system components
    - E.g.: periodic, interrupt / software sporadic
  - The runtime execution model that underpins the system design
    - E.g.: with / without pre-emption, priority
  - The means of communication and synchronisation availed to system components
    - E.g.: protected regions, signals, guards, wait queues

Computational model

- The design and implementation dictionary, which captures
  - The real-time attributes and the activation characteristics of the system components
  - E.g.: periodic, interrupt / software sporadic
  - The runtime execution model that underpins the system design
    - E.g.: with / without pre-emption, priority
  - The means of communication and synchronisation availed to system components
    - E.g.: protected regions, signals, guards, wait queues

Seminario: un approccio linguistico al controllo ...
Desirable characters (3/6)

- **Flexibility**
  - To tolerate development feedback
    - We must contemplate late design changes
  - Design can hardly be fixed at specification time
- To favour modularity and scalability of design
  - We want to enable loosely-coupled development of components
- To achieve scalability of system
  - We need a system concept that scales to needs efficiently
  - Efficiency is inversely proportional to the # of wasted cycles

Desirable characters (4/6)

- **Runtime efficiency**
  - Deterministic behaviour
    - Predictability
    - Time-bound services
  - Performance
    - Simple on-line scheduling decisions
  - Flexible scheduling criteria
    - Fixed priority, permanent attribute to reflect urgency of service
    - Pre-emption, to reflect priority

Desirable characters (5/6)

- Statically verifiable
  - The scheduling algorithm and its effects must be mathematically representable
    - Period of activation, minimum inter-arrival time, \( T \)
    - Worst-case computation time, \( c \)
    - Worst-case blocking time, \( B \)
    - Deadline, \( D \)
    - Priority, \( P \)
    - Response time, \( R \) \( (R \leq D) \)
  - \( T,C,D \) are real-time attributes of the application
  - \( R \) is a runtime function of \( P,C,B \)

Desirable characters (6/6)

- **Response Time Analysis**
  - Response time for thread \( i \)
    \[ R_i = B_i + C_i + \sum_{j \in HP(i)} \left( \frac{t}{T_i} \right) C_j \]
  - Interference from higher-priority threads
    \[ I_i = \sum_{j \in HP(i)} \left( \frac{t}{T_i} \right) C_j \]
  - Interference from interval timer
    \[ K_i = \sum_{j \in HP(i) \text{ cyclic}} \left( \frac{t}{T_i} \right) (\text{Clock Int} + \text{Ready} + \text{Select}) \]
  - Worst-case blocking maximum = \( B_i = \) interference from lower-priority threads

The Ravenscar profile (1/10)

- A concurrent language runtime subset with
  - Adequate expressive power
  - Designed to meet the requirements of high-integrity embedded real-time systems
  - Minimal footprint
  - Strip away all disallowed services
  - High efficiency
  - Simple and predictable scheduling decisions
  - Statically verifiable behaviour
  - Based on sound scheduling theory
  - Certifiable runtime code
The Ravenscar profile (2/10)

- Requires:
  - Single activation event per thread of control
    - ✓ Time, external interrupt, software synchronisation
      - Rationale: to comply with the power of the associated scheduling theory (e.g. Response Time Analysis)
  - Non-suspending execution within activation
    - ✓ Only suspension for next activation event
      - Rationale: ditto
    - No termination, no dynamic creation

The Ravenscar profile (3/8)

- Requires (cont’d):
  - Data-oriented synchronisation via protected objects with priority ceiling emulation
    - ✓ To enable concurrent collaborative processing
      - Rationale: to bound priority inversion while controlling blocking effects
    - Simple synchronisation via suspension objects
      - ✓ To enable very-low-overhead activation
      - Rationale: to give users access to low-level high-efficiency private semaphore P and V

The Ravenscar profile (4/10)

- Requires (cont’d):
  - Single-position entry queues
    - ✓ Fully deterministic synchronisation service
      - Rationale: to avoid non-deterministic waiting time upon task queues forming on entry and to permit simpler and leaner runtime
    - At most one entry per protected object
      - ✓ No two barriers simultaneously open
      - Rationale: to avoid non-deterministic select policy and permit simpler and leaner runtime

The Ravenscar profile (5/10)

- Requires
  - Absolute time delay
    - ✓ Exclusive use of high-precision time type package
      - Rationale: to attain monotonic, accurate, fine-grained time base
  - Prohibits:
    - ✓ All other concurrency features (a whole load of them!)
      - Sophistication that raises expressive power but detracts from predictability and static verification
    - Potentially blocking protected operations

The Ravenscar profile (6/10)

- Best placed in a concurrent language with compile-time and run-time conformance checks
  - So much preferable to manual checks!
    - ✓ Facilitates enforcement of design and coding rules
  - You code activities and not the scheduler
  - You tell the scheduler what you want by:
    - ✓ Defining the activation event of tasks
    - Setting the priority level of tasks

The Ravenscar profile (7/10)

- Inherently avoids deadlock
  - On single CPU, priority ceiling emulation prevents circularities in ownership of and contention for locks
  - Is in perfect match with HRT-HOOD
    - Which provides specification-to-implementation support for the RP computational model
Ravenscar example

```
Ravenscar example

```

Ravenscar example

```
Ravenscar example

```

Ravenscar example

```
Ravenscar example

```

Ravenscar example

```
Ravenscar example

```

The Ravenscar profile (8/10)

```
The Ravenscar profile (8/10)

```

Seminario: un approccio linguistico al controllo ...
The Ravenscar profile (9/10)

- With the Ravenscar profile we have:
  - The expressive power to represent the entities of the problem domain accurately
  - A highly efficient and predictable runtime
  - A computational model directly amenable to static analysis
  - High-level means to control jitter and minimise latency
  - Effective means for modular and scalable design

The Ravenscar profile (10/10)

<table>
<thead>
<tr>
<th>Runtime Primitive</th>
<th>Executed by runtime for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter_Protected_Object</td>
<td>All threads</td>
</tr>
<tr>
<td>Leave_Protected_Object</td>
<td>All threads</td>
</tr>
<tr>
<td>Leave_Interrupt_Wait</td>
<td>Protected interrupt handlers</td>
</tr>
<tr>
<td>Leave_Semaphore_Wait</td>
<td>Protected interrupt handlers</td>
</tr>
<tr>
<td>Leave_Semaphore_Queue</td>
<td>Sporadic threads</td>
</tr>
<tr>
<td>Select_from_Ready_Queue</td>
<td>All threads</td>
</tr>
<tr>
<td>Switch_Running_Context</td>
<td>All threads</td>
</tr>
<tr>
<td>Insert_in_Delay_Queue</td>
<td>Periodic threads</td>
</tr>
<tr>
<td>Handle_Interval_Time_Interrupt</td>
<td>Periodic threads</td>
</tr>
<tr>
<td>Handle_External_Queue</td>
<td>Periodic interrupt handlers</td>
</tr>
<tr>
<td>Select_PnSpinlock</td>
<td>Runtime structures</td>
</tr>
</tbody>
</table>

Conclusion (1/4)

- The Ravenscar profile happens to be a natural restriction of standard Ada tasking
- It could equally well find a home in real-time Java
- The profile allows us to design solutions for embedded real-time system problems
- It delivers us from finding problem representations that fit invariant solutions

Conclusion (2/4)

- Standardisation status
  - Profile outline in
  - Profile rationale in
    - "Guide for the use of the Ada Ravenscar Profile in High Integrity Systems" ISO/IEC TR being finalised

Conclusion (3/4)

- Standardisation status (cont’d)
  - Profile definition in
    - Ada Issue 249
    - Will become an official amendment in the forthcoming language revision
  - 2 official implementations, more to come
    - Aomix/Object Ada RAVEN
      - Proprietary, for PowerPC, Intel, ERC32 targets
    - GNAT/ORK
      - Open source, for ERC32, Intel targets

Conclusion (4/4)

- Relation to Real-Time Java Specification
  - NIST requirements spec [www.nist.gov/rt-java]
    - Too open-ended, not tight enough, soft RT
  - J Consortium [www.j-consortium.org]
    - Aims at an ISO PAS (no standard) – declining interest
    - Captures the RP as a Real-Time Core Profile
  - Real time with Java flavour
  - Sun’s Real-Time Expert Group [www.rj.org]
    - Values JVM compatibility more than meeting HRT
    - Java with real-time flavour