Predictable Execution with IEC 61499

Li Hsien Yoong
The University of Auckland
Sequence of presentation

- What has been achieved:
  - Deterministic behaviour of centralized IEC 61499 systems

- Current goal:
  - Deterministic behaviour and predictable timing of distributed IEC 61499 systems

- Industrial application:
  - Video and tool demonstration
What is the IEC 61499?

- An open standard of the International Electrotechnical Commission (IEC)

- Component-oriented approach for designing distributed industrial-process control systems to meet future requirements of intelligent automation

- Model-based development for industrial control software
  - Graphical – *function blocks*
  - Target-independent
  - Supports reuse of IEC 61131
  - System-level design of distributed systems
An IEC 61499 example

- Model of a cruise control system
  - Each block encapsulates a sub-component
  - Clearly defined event/data flow between components
An IEC 61499 example

- Function blocks consist of input/output interface:
  - Events
  - Data
- Three types of function blocks:
  - Basic
  - Composite
  - Service interface
Design artifacts in IEC 61499

- Hierarchy of design artifacts:
  - Function block – encapsulates a functional unit of software
  - Resource – independent unit of software made up of a network of function blocks
  - Device – programmable controller that executes function blocks
  - System – a collection of devices implementing the desired control function
Some ambiguities...

- Execution semantics of function blocks is not fully defined in standard.
- **Two main deficiencies:**
  - *Lack of any notion of time* – What is the lifetime of events? Are events simultaneous?
  - *Lack of any notion of composition* – How do blocks communicate? Is there a (partial) order for block execution?
- **Different solutions from different vendors:**
  - Function Block Run-Time (FBRT)
  - 4DIAC Run-Time Environment (FORTE)
  - ISaGRAF
Problem 1: Transition evaluation in an ECC

- **Lifetime of events**
  - E1 occurs, C1 and C2 are true.
  - Should transition to State4 be taken?

- **Eventless transitions**
  - E1 occurs, C2 is true, C1 and C3 are false.
  - Will State5 or State6 ever be reached?
Problem 2: Composition of blocks in a network

- **Race conditions** – FB1 may be triggered by FB2 before it can complete its execution.

- **Starvation** – FB3 may be left unattended while FB1 and FB2 monopolize the execution.
Key issues addressed

- Formal model for function block systems
  - Globally asynchronous locally synchronous paradigm for distributed IEC 61499 systems

- Software synthesis
  - Automated generation of efficient code without run-time environment or middleware

- Abstract communication patterns for distribution
  - Specify communication semantics using known patterns
## Mapping function blocks to Esterel

<table>
<thead>
<tr>
<th>Function block element</th>
<th>Esterel feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function blocks</td>
<td>Modules</td>
</tr>
<tr>
<td>Events</td>
<td>Pure interface signals</td>
</tr>
<tr>
<td>Data</td>
<td>Value-only interface signals</td>
</tr>
<tr>
<td>Internal variables</td>
<td>Value-only local signals</td>
</tr>
<tr>
<td>EC states</td>
<td>Demarcated by <code>pause</code> statements</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Instantaneous modules</td>
</tr>
<tr>
<td>Transition conditions</td>
<td><code>await</code> statements</td>
</tr>
<tr>
<td>Function block network</td>
<td>Parallel composition of modules</td>
</tr>
</tbody>
</table>
Handling transition evaluations

// State1
pause;
await
case immediate ?C1 do
  // State2
  pause;
  await immediate E1 and ?C2;
  // State4
  pause;
  case immediate E1 and ?C2 do
    // State3
    pause;
    await immediate E1 and ?C2 do
      // State5
      pause;
      case immediate ?C1 do
        // State5
        pause;
        case immediate ?C3 do
          // State6
          pause;
          end await
        end await
      end await
Handling composition of function blocks

```
| await immediate EI1; emit EO1; emit EO2; pause; |
| ECC (fragment) |
|________________|
| FB1            |
| EI1            |
| EI2            |
| S1             |
| FB2            |
| EI3            |
| EO3            |
| ECC (fragment) |
|________________|
| FB3            |
| EI4            |
| EO4            |
| DO3            |
| DI4            |
| State          |

||
| await immediate S2; emit S1; emit EO2; pause; |
| ||
| await immediate S1; emit S2; pause; |

||
| await immediate pre(S2); emit S1; emit EO2; pause; |
| ||
| await immediate pre(S1); emit S2; pause; |
```
Translating function blocks to Esterel

module CruiseControl:
  input cclock, set, off, resume, quickAccel, quickDecel;
  // other inputs omitted
  output speed, speedSet, throttleChg, regulOff;
  // other outputs omitted
  signal s0 : value signed<[16]>, s1, s2, s3, s4, ... in
  run CruiseManager [...] 
  ||
  run Throttle [...] 
  ||
  run SpeedGauge [...] 
... 
end signal
end module
GALS model for distributed systems

- IEC 61499 is a standard for distributed control systems. Concurrency arises from 2 sources:
  - Parallelism in the controlled environment (*logical* parallelism) – disciplined synchronization
  - Distribution of the control systems (*literal* parallelism) – communicate only when necessary

- GALS model for distributed IEC 61499 systems
  - *Resources* are synchronous islands.
  - Resources communicate with each other using *communication function blocks*.
  - Communication function blocks encapsulate various communication patterns.
Communication function blocks

Client-server function blocks

Publish-subscribe function blocks
Example of a distributed system
Experimental results – speed

![Bar chart showing time (in ms) for various processes with logarithmic scale.](image)
Experimental results – normalized speed
Experimental results – size

The graph shows the object code size (in kB) on a logarithmic scale for various systems and scenarios. The y-axis represents the size in kilobytes, ranging from 1 to 128.00. The x-axis lists different systems and scenarios, including LED flasher, Drill station, Temperature controller, Speed regulator, Railway, Cruise control 1, Cruise control 2, Water monitor, Distributing station, and Baggage handling station. Each scenario is represented by different colors: FBC-Cinst, FBC-C, FBC-Strl, FBRT, and FORTE.
Experimental results – normalized size
Towards a multi-rate framework

- Multi-rate framework is based on the *synchronous approach*. Consists of several clocks derived as a rational multiple of some base clock.

- **Why multi-rate?**
  - Multi-rate systems ensure determinism and facilitates verification of global behaviour.
  - Amenable to static timing analysis — *real-time systems*.
  - When used with time-triggered networks, deterministic distributed real-time systems can be achieved.
Basic idea to ensure timing determinism

- Assume execution platform to be always fast enough:
  - Language provides semantics to express timing constraints of environment (reactivity).
  - Compiler ensures that timing constraints are met on platform (schedulability).
- Similar abstraction to synchronous approach, but for multi-rate systems, every module requires timing-deterministic I/O operations.
Proposed approach for multi-rate software

- Goals:
  - Facilitate analysis of global behaviour, e.g., by simulation of formal verification.
  - Implementable using a simple static priority preemptive scheduler.
  - Efficient use of computing resources, while ensuring equivalence with single-task approach.
Proposed approach for multi-rate software

- **Scheduling:**
  - Rate monotonic preemptive scheduling will be used.
  - All task periods are multiples of the base period. The base period must be a common factor of all task periods.

- **Timing constraints:**

  \[
  WCET(P_0) + \text{other}_0 < T_0
  \]

  \[
  \sum_{i=0}^{n-1} \left[ N_i \times (WCET(P_i) + \text{other}_i) \right] + WCET(P_n) + \text{other}_n < T_n
  \]

  where \( N_i = \left\lfloor \frac{T_n}{T_i} \right\rfloor \)
Proposed approach for multi-rate software

**Issues:**

- Determinism means producing the same output sequence for a given input sequence at specific instants of time.
- For synchronous programs, execution time may vary as long as:
  \[ \text{WCET}(P) + \text{other} < T \]
- For multi-rate programs in a multi-tasking scheme, I/O operations must remain timing-deterministic even with variations in execution time.
Slow-to-fast resource communication

- Illustration: assume $T_R = 2T_r$
- Case 1: $R$ computes slowly
Slow-to-fast resource communication

- Case 2: R computes faster than usual

- To maintain determinism, R must communicate to r at the beginning of the next fast cycle, where R starts its next slow cycle.
Fast-to-slow resource communication

- Communication from the fast to slow resource can occur instantly after the completion of the fast resource.
- But, communication must not happen during the computation of the slow resource.
If the slow resource gets delayed by an intermediate resource, the original data from the fast resource must not get overwritten.
General rule for communication

- From slow to fast: use delayed communication
- From fast to slow: sample and hold communication
- For modules of same speed: use delayed communication

Implications for implementation:
- Outputs must be scheduled as separate non-preemptible tasks.
Extensions to distributed systems

- Similar determinism is achievable using time-triggered networks.
- TDMA cycle is divided into separate communication slots.
- Order of slots will be the same as priority derived using rate-monotonic scheduling.
Industrial impact

- Glidepath (airport baggage handling system)
- Powerplants (greenhouse controller)
- Integration with nxtControl Studio (commercial IDE)
- Auckland UniServices (IDE – editor, compiler, timing analyzer, and module checker for function blocks)