LUNA: Hard Real-Time, Multi-Threaded, CSP-Capable Execution Framework

M. M. Bezemer
R. J. W. Wilterdink
J. F. Broenink

Control Engineering, University of Twente, The Netherlands
Outline

- Context and Introduction
- Framework architecture
  - Threading
  - Channels
  - CSP processes
  - Alternative
- Results
  - Measurements
  - Comparison
- Conclusions
Context

- Controlling embedded set ups / robots
  - Low resources
  - Custom build (Linux) Operating System
  - Guaranteed deadlines for updates for calculated motor signals

- Frameworks help with generic implementations / behaviour

- Multi core and/or distributed systems
  - Requires extra support from framework
  - CSP helps with organizing the execution flow

- Support multiple targets
  - Also requires extra support from framework
Controlling actual set ups requires different layers

- Loop control - Control the physical system
- Sequence control - Provide 'setpoints'
- Supervisory control - Complex tasks: planning, mapping, …
- User Interface - Connection with user
Controlling actual set ups requires real-time levels

- Hard real-time - must meet deadlines
- Soft real-time - should meet deadlines
- Non real-time - everything else
Introduction

- Requirements for an embedded control software framework
  - Hard real-time
  - Multi-platform
  - Thread support
  - Scalability

- Other 'handy features'
  - CSP execution engine
  - Low development time for framework user
  - Debugging and Tracing
Introduction

- Existing solutions do not meet all requirements
  - C++CSP2 not hard real-time
  - CTC++ not multi-threaded

- Develop a new framework to meet all the requirements

LUNA

LUNA is a Universal Networking Architecture
Architecture

1) Core Components
   - Platform support components + utility components

2) High-level Components
   - Platform independent components

3) Execution Engine Components
   - Components to determine the order of execution

- Core Components:
  - Threading
  - Mutexes
  - Semaphores
  - Others (timers, timing, sockets, ...)
  - OS abstraction

- High-level Components:
  - Networking (TCP/IP, ...)
  - Inspection
  - Device drivers
  - Others

- Execution Engine Components:
  - CSP
  - State Machine
  - Others
Threading

- Hybrid threading support
  - OS Threads – required for multi-core support
  - User Threads – fast(er) switching between threads
Threading

- CSP implementation with separation of concerns
  - Core components for platform-dependent threading components
  - Execution engine component for CSP algorithm implementation

![Diagram showing the components and relationships of the threading framework.]

- Core Components
- Execution Engine Components

- CSP functionality
- Sequential, Parallel, Recursion
- CSPConstruct
- CSProcess

- IThread
- Runnable
- IThreadBlocker

- OSThread
- OS Scheduler

- UThread
- UScheduler

- Scheduling mechanism
Channels

- Two types of channels
  1) Rendez-vous communication between 2 OS threads
     Blocks the complete OS thread, used for multi-core communication
  2) Rendez-vous communication between User Threads
     Faster and without blocking complete OS thread
     Complete CSP functionality: buffered, guarded
CSP Execution Engine

- CSP Process
  - Initialise process (pre run)
  - Perform main operations
  - Finalise (post run)

- Example of a sequential process
Alternative

- Naive Alternate implementation
  - Possibility of 'high-jacking' the channel, blocks GuardedReader

- Example: 1 GuardedReader, 1 'regular' Reader
Alternative

- Solution for the high-jacking problem
  - Added lock to channel, now Reader blocks
Results

- Context-switch speed
  - Switch as fast as possible between two threads

- Commstime
  - Determine CSP efficiency

- Real robotic set up
  - Performance in real life situations
Results

- **Context-switch speed**
  - Switch as fast as possible between two threads
  - 10,000 switches, average time

<table>
<thead>
<tr>
<th>Framework</th>
<th>OS thread (µs)</th>
<th>User thread (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC++ 'original'</td>
<td>-</td>
<td>4.275</td>
</tr>
<tr>
<td>C++CSP2</td>
<td>3.224</td>
<td>3.960</td>
</tr>
<tr>
<td>CTC++ QNX</td>
<td>3.213</td>
<td>-</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>3.226</td>
<td>1.569</td>
</tr>
</tbody>
</table>

- **OS thread switch speed is comparable**
- **User thread switch speed is fast!**
  - LUNA has virtually no management overhead
  - (high speeds only do not determine the framework efficiency)
Results

- Commstime Benchmark
  - Measure the efficiency of the CSP execution
  - 10,000 cycles, average time
Results

- Commstime Benchmark

<table>
<thead>
<tr>
<th>Framework</th>
<th>Thread type</th>
<th>Cycle time (µs)</th>
<th># Context-switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC++ 'original'</td>
<td>User</td>
<td>40.76</td>
<td>5</td>
</tr>
<tr>
<td>C++CSP2</td>
<td>OS</td>
<td>44.59</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>18.60</td>
<td>4</td>
</tr>
<tr>
<td>CTC++ QNX</td>
<td>OS</td>
<td>57.06</td>
<td>-</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>OS</td>
<td>34.03</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>9.34</td>
<td>4</td>
</tr>
</tbody>
</table>

- OS thread cycle time somewhat faster
  - Efficient way to block a OS thread (low management)

- User thread cycle time fast!
  - Mainly due to efficient context-switching

- Naive code generation results in bad performance
  - Design point of view versus execution point of view
Results

- Simple 2 DOF pan-tilt robotic set up

- Used for educational purposes
  - Practical assignments
    - Easy platform for experimenting
    - Vision-in-the-loop
    - Spot tracking
  - Courses
    - Real-time software development
    - Hardware/Software trade-offs
Results

- Real Robotic Set up
  - Performance in real life situations
  - Measurement runs of ~60 seconds

<table>
<thead>
<tr>
<th>Framework</th>
<th>Frequency (Hz)</th>
<th>Cycle time (ms)</th>
<th>Standard deviation (µs)</th>
<th>Processing time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>CTC++ 'original'</td>
<td>100</td>
<td>11.00</td>
<td>10.90</td>
<td>11.11</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1.18</td>
<td>0.91</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td>1000.15</td>
<td>1.00</td>
<td>0.91</td>
<td>1.10</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>100</td>
<td>10.00</td>
<td>9.93</td>
<td>11.00</td>
</tr>
<tr>
<td>(user threads)</td>
<td>1000</td>
<td>1.00</td>
<td>0.80</td>
<td>2.01</td>
</tr>
<tr>
<td></td>
<td>1000.15</td>
<td>1.00</td>
<td>0.79</td>
<td>1.21</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>100</td>
<td>10.00</td>
<td>9.97</td>
<td>11.00</td>
</tr>
<tr>
<td>(OS threads)</td>
<td>1000</td>
<td>1.00</td>
<td>0.96</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>1000.15</td>
<td>1.00</td>
<td>0.95</td>
<td>1.05</td>
</tr>
</tbody>
</table>
### Results

- **Real Robotic Set up**

<table>
<thead>
<tr>
<th>Framework</th>
<th>Frequency (Hz)</th>
<th>Cycle time (ms)</th>
<th>Standard deviation (µs)</th>
<th>Processing time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>CTC++ 'original'</td>
<td>100</td>
<td>11.00</td>
<td>10.90</td>
<td>11.11</td>
</tr>
<tr>
<td></td>
<td>1000.15</td>
<td>1.00</td>
<td>0.91</td>
<td>1.10</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>100</td>
<td>10.00</td>
<td>9.93</td>
<td>11.00</td>
</tr>
<tr>
<td>(user threads)</td>
<td>1000.15</td>
<td>1.00</td>
<td>0.79</td>
<td>1.21</td>
</tr>
<tr>
<td>LUNA QNX</td>
<td>100</td>
<td>10.00</td>
<td>9.97</td>
<td>11.00</td>
</tr>
<tr>
<td>(OS threads)</td>
<td>1000.15</td>
<td>1.00</td>
<td>0.95</td>
<td>1.05</td>
</tr>
</tbody>
</table>

- LUNA user threads are faster than CTC++

- LUNA OS threads are slightly slower than CTC++ (user threads!)
Conclusions

- LUNA meets all requirements
  - Hard real-time
  - Multi-platform
  - Multi-threaded
  - Scalable

- Fast and efficient compared to related frameworks

- Usable for controlling real robotic set ups

- Need model optimisation for code generation
Future work

- Develop controller for Production Cell with LUNA
  - To show that complex set ups can also controlled using LUNA

- Support Linux, RTAI and/or Xenomai
  - More drivers available to use webcams, joysticks, …

- Support for Windows
  - Well known by (starting) developers
  - Good (graphical) debugging facilities

- Graphical CSP modelling tool with code generation capabilities
  - Replacement for gCSP
  - Model optimisation algorithms included