When Hard Realtime Matters: Software for Complex Mechatronic Systems

Berthold Bäuml and Gerd Hirzinger

Institute of Robotics and Mechatronics
DLR – German Aerospace Center
berthold.baeuml@dlr.de
Mechatronic Systems

- “Mechanics in the loop”
- demands hard *realtime* (= short latencies and determinism)
- to prevent *injury* and *damage*!
New Level of Complexity

- complex single robotic component (DLR-Hand-II: 13DOF, 100 sensors)
- compound systems with tightly interacting parts (Justin: 43 DOF, 250 sensors)
- rates of up to 3kHz over all DOF (up to 10kHz in next generation)
- sophisticated control algorithms (e.g. gravity compensation, impedance)
- tight coupling to non-realtime modules for user interaction, perception and planning

- Heterogenous team of experts developing software (~ 10 members)
- Iterative, flexible and adaptive development flow (esp. for research prototypes)

Challenges:
- “modern” software paradigms for complex systems:
  - modularity (“component based software engineering”)
  - concurrent execution (“services”)
  - distributed execution (“multi core, multi CPU, networked clusters”)
  - but under **hard realtime** constraints.
“Decentral net of functional blocks and communication links”

Software concept has to provide
an abstract, **functional view** and the **mapping** to a concrete hardware setup.
# Robot Middleware

<table>
<thead>
<tr>
<th>Middleware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORCA (Brooks et al. 2005)</td>
<td>used in mobile robotics soft realtime in the 100Hz range</td>
</tr>
<tr>
<td>MARIE (Cote et al. 2004)</td>
<td></td>
</tr>
<tr>
<td>MIRO (Utz et al. 2002)</td>
<td></td>
</tr>
<tr>
<td>Player (Vaughan et al. 2003)</td>
<td></td>
</tr>
<tr>
<td>MS Robotics Studio (msdn.microsoft.com/robotics)</td>
<td>WindowsXP -&gt; non realtime, WindowsCE ?</td>
</tr>
<tr>
<td>OROCOS (<a href="http://www.orocos.org">www.orocos.org</a>)</td>
<td>hard realtime, &gt;500Hz for two 6DOF arms more complex systems, distributed computation ?</td>
</tr>
<tr>
<td>OpenHRP (Kanehiro et al. 2004)</td>
<td>RT-Middleware, based on CORBA in applications hard realtime part in monolithic modules with proprietary communication</td>
</tr>
<tr>
<td>MCA (<a href="http://www.mca2.org">www.mca2.org</a>)</td>
<td>used in humanoid robots ARMAR hard realtime on one node, but TCP/IP</td>
</tr>
<tr>
<td>YARP (Metta et al. 2006)</td>
<td>used in complex robots like “Domo”, but there the hard realtime parts run on DSP boards supports QNX and based on ACE -&gt; distributed hard realtime execution possible ?</td>
</tr>
<tr>
<td>MIRPA</td>
<td>hard realtime, distributed execution performance when many DOF ?</td>
</tr>
</tbody>
</table>
Pragmatic Approach

- None of the existing middleware concepts seemed to fit our main needs: hard realtime, scalable computing resources, easy to use.
- How far can we go, when doing it as simple as possible?

- Hardware constraints?
  - D/A- and A/D-conversion in the robots
  - fast buses up to 1GBit/s to the robots
  - CPU clocks > 3GHz, Multi-Core and Multi-CPU (e.g. Quad-DualCore Opteron >50GFLOPS)
  - fast communication with low delay (1GBit: 1.5KByte in 15us)
  - configurable infrastructure: eg. switched Ethernet or P2P links

No!
Pragmatic considerations

What is essential?
- deterministic, concurrent execution
- fast, deterministic communication
- standardized interfaces for communication
- mechanisms for system configuration, startup and shutdown

and what is not?
- dynamic reconfiguration at runtime
- higher levels of abstractions (e.g. abstract “range sensor”)

What can we get from a realtime OS?
- Processes
- Shm on one PC
- Fast, deterministic drivers to (multiple) network cards

What has to be done ourselves?
- standardized interfaces
- abstraction of distributed communication
- system handling
Implementation of aRD-concept (‘‘agile Robot Development’’)

1. aRDnet-suite: lightweight (~3000 lines)
   - aRDnet-library (C/C++): implementation of block’s ports
   - ardnet block: bridge between blocks on different PCs
Implementation of aRD-concept

1. aRDnet-suite: lightweight (~3000 lines)
   - aRDnet-library (C/C++): implementation of block’s ports
   - `ardnet` block: (UDP)-bridge between blocks on different PCs
   - Unix-like hierarchical startup and shutdown of distributed system, with detailed configuration of QoS for communication

2. Matlab/Simulink-toolchain
   - standard tool for control design
   - Automatic code generation with RTW
   - Parallelizing with RTLab from OpalRT ([www.opal-rt.com](http://www.opal-rt.com))
   - stubs for interfacing aRDnet-blocks
Implementation of aRD-concept

1. aRDnet-suite: lightweight (~3000 lines)
   - aRDnet-library (C/C++): implementation of block’s ports
   - ardnet block: (UDP)-bridge between blocks on different PCs
   - Unix-like hierarchical startup and shutdown of distributed system, with detailed configuration of QoS of communication links

2. Matlab/Simulink-toolchain
   - standard tool for control design
   - Automatic code generation with RTW
   - stubs for interfacing aRDnet-blocks

Key decision: QNX for realtime and Linux for non-realtime
(limited support for VxWorks and Windows)
Performance:
apRDnet performance

QNX-PC with Core2Duo, 2.8GHz

roundtrip of packet of 1kByte size between two standalone blocks

- on one QNX-PC: 20us in worst case
- on two QNX-PCs with ardnet-bridge (UDP) over P2P 1GB-Ethernet link: 200us in worst case (160us on average)
Performance:
High Rate

Simulink-HIL-setup: reading analog values and writing them to harddisk

- Rates > 50 KHz without drop-outs possible
- At 30 KHz system overhead is less than 10 % cpu-time
Applications: (July 2005)
Preliminary study for Two-Hand-Arm-System
Applications: (May 2006)
Two-Hand-Arm-Torso Justin
vision

world model

path planning

GUI

input device

VxWorks

QNX

Linux/Windows

REAL TIME

hand
dev

hand
dev

arm
dev
torso
dev

arm
dev

3D
viewer

camera
dev

state
machine

EC

dev

GUI
Two-Hand-Arm-Torso Justin
Applications

- Two-Hand-Arm-Torso Justin
- Virtual reality with haptic feedback
- Telepresence/Telemanipulation
- Medical robotics
- Brain/Muscle-Interface-Group
- Teststands
  - Finger
  - Arm
  - Micro-surgical devices
  - …
- Dynamics simulators
  - Ball catching
  - …
- …

→ ~ 15 “aRD-seats” at our institute
Conclusions

- **hard realtime** is a must, when mechanics is in the loop
- increasing complexity of mechatronic systems demands for modern software concepts also there, namely: modularity, concurrent execution, distributed execution
- an “operating system for robotics” has to provide hard realtime
- our pragmatic and lightweight aRD concept proved the feasibility even on “cheap” PC based computing hardware by
  - introducing only a thin layer of abstraction above the realtime OS
  - allowing to configure the QoS of all communication

Hope, that a perfect operating systems for robotics will emerge soon, so that we can again work on that, what roboticists love to do …
Justin Serving Tea ...