UNICORE NXT

Controlling a Lego NXT robot through UNICORE

May 29, 2012 | Sandra Bergmann & Matthias Richerzhagen
Overview

- What is a NXT robot
- How to control the NXT robot
- Integration in UNICORE
- Current implementation
- Future
UNICORE NXT
Part I: What is a NXT robot

May 29, 2012 | Sandra Bergmann & Matthias Richerzhagen
What is a NXT robot

Hardware
What is a NXT robot

Hardware specifications

<table>
<thead>
<tr>
<th>processor</th>
<th>Flash memory</th>
<th>Ram</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ARM-processor(^1)</td>
<td>256 kB</td>
<td>64 KB</td>
<td>48</td>
</tr>
<tr>
<td>coprocessor(^2)</td>
<td>4 KB</td>
<td>512 Byte</td>
<td>8</td>
</tr>
</tbody>
</table>

- Bluetooth
- USB-2.0
- 3 Motor-(Output)-Ports
- 4 Sensor-(Input)-Ports
- 100x64px LC-Display
- Speaker(8-bit, 2-16 kHz)

\(^1\) AT91SAM7S256
\(^2\) Atmel 8-Bit AVR, ATmega48

Source: http://de.wikipedia.org/wiki/NXT
What is a NXT robot

Software

- Access with USB or Bluetooth
- Programming:
  - Standard Firmware: Drag&Drop
  - Custom firmwares offer other programming languages for example: Java (lejos_nxj) (Much more available)
UNICORE NXT
Part II: How to control the NXT robot

May 29, 2012  |  Sandra Bergmann & Matthias Richerzhagen
How to control the NXT robot

Overview

Local
A NXT program is compiled and uploaded to the NXT and then executed on the NXT-Brick.

Remote
A program running on a host device connects to the NXT and controls motors/sensors remotely.
How to control the NXT robot

Differences

**Local**
- Good interfaces
- Unknown when or if a program has been finished
- Job is finished, when the program has been uploaded

**NO SCHEDULING!**

**Remote**
- Very tricky access to sensors
- Job finishes, when job is done

**Scheduling!**
UNICORE NXT
Part III: Integration in UNICORE

May 29, 2012 | Sandra Bergmann & Matthias Richerzhagen
Integration in UNICORE

Overview

**Script Job**
- Uploading and running a compiled .class file on the NXT
- Running a remote-program

**simpleidb**
Creating predefined jobs by using the simpleidb file that can be either local or remote-programs
Uploading a robot program

- Bluetooth and Lejos-NXJ commands must be available on target machine!
- Job finishes when the program has been uploaded
- Source or class files must be uploaded with the job
- Robot is not available when program is running
- Results need to be fetched by a remote-program

Example Code

```
nxjc LocalTest.java
nxj -r LocalTest
```
Running a remote program

- Bluetooth and Lejos-NXJ libraries must be available on target machine!
- Job finishes when program finishes
- Source or class files must be uploaded with the job
- Remote API is tricky
- Only one remote program at a time

Example Code

nxjpcc RemoteTest.java
nxjpc RemotetTest
UNICORE NXT
Part IV: Our Use Case

May 29, 2012  |  Sandra Bergmann & Matthias Richerzhagen
Our Use Case
Follow the path

Idea:
Find and follow a path from A to B on a given map avoiding obstacles.

Prerequisites:
- Map
- Position and rotation of the robot

Tasks:
- Robot programs such as "Travel x" or "Rotate $\alpha$"
Our Use Case
Follow the path - instruction queue

Process:
- Find a path from A to B → waypoints
Our Use Case
Follow the path - instruction queue

Process:
- Find a path from A to B ⇒ waypoints
- Convert this waypoints to travel and rotate commands
Our Use Case
Follow the path - instruction queue

Process:
- Find a path from A to B ⇒ waypoints
- Convert this waypoints to travel and rotate commands
- Send these commands to the NXT robot
Current implementation

What we got

Components:

- Server that holds the bluetooth connection
Current implementation

What we got

Components:

- Server that holds the bluetooth connection
- Client, that submits travel and rotate jobs to the Server
Current implementation
What we got

Components:
- Server that holds the bluetooth connection
- Client, that submits travel and rotate jobs to the Server
- Tool for map ⇒ waypoints
- Tool for waypoints ⇒ travel/rotate commands
Current implementation
What we got

Components:
- Server that holds the bluetooth connection
- Client, that submits travel and rotate jobs to the Server
- Tool for map $\Rightarrow$ waypoints
- Tool for waypoints $\Rightarrow$ travel/rotate commands
- simpleidb entries for the application specification
Current Implementation
Architecture

User sends workflow to UNICORE, which starts the NXTJobClient. The NXTJobClient waits for FINI or FAIL. The UNICORE then sends workflow result. The Bluetooth connection is used to move/turn with arguments.

Job information is sent via the Socket connection to the NXTJobServer, which executes each job. The job status is WAIT, WORK, FINI or FAIL.
Demo
Current implementation

Problems

- Workflows interfere with each other
- Cannot associate job with owner
- Jobs are not independent from each other
- Long idle times, between 2 job executions
Future

- reduce the time between job executions $\rightarrow$ extension of the workflow system
- Association between job and owner $\rightarrow$ more security
- GridBeans
Thanks
Any Questions?