Multi-agent approach to control

Andrej Lúčny
MicroStep-MIS & DAI FMFI UK Bratislava
andy@microstep-mis.com
http://www.microstep-mis.com/~andy
• Escaping from hardware layout
• Multi-agent modularity
• Agent-space architecture
• Indirect communication
• Data flow many:many
• Time validity
• Implicit sampling
• Data packages
• Subsumption and priorities
• Comparison with iConnect
• Integration of traditional AI
• An example
Future of control

• Structure of control is exclusively implemented by software

However:

• Layout of the software modules still mimics hardware layout

Question:

• Can we imagine more sophisticated and profitable layout?
Traditional control

- modules have fixed number of inputs, outputs and parameters
- one output is linked to several inputs
- transformation of inputs to outputs is performed by a scheduler and it is often uninterruptible
Multi-agent modularity

• An alternative solution is application of multi-agent modularity which turns modules to agents and links to communication among them

• In this way we give to control system the same modularity as typical for distributed and decentralized systems
Multi-agent system example

• Typical example: robot-soccer
Multi-agent system

• It would be too difficult to write a program which controls all the players.
• It is much easier to code programs for individual players and let the team control to emerge from their interaction
• Such interacting programs are called agents
Decentralization

Such solution is decentralized:

• for example, if we remove midfielder from the team, striker does not stop (endlessly waiting for a pass from the removed midfielder), just probably scores a goal less frequently. Even if the striker never got a pass, he is still moving and ready to shoot ball whenever it is available to him.
Nature of agents

- agents can be implemented as objects equipped with an own thread of control and a mechanism of a mutual data exchange including sensation and action of the system environment
Nature of agents

- each agent is endlessly running a sense-select-act cycle. Any course through this cycle calculates some actions upon information sensed from environment or provided by other agents.
Communication among agents

The communication mechanism can be based on

• direct message passing
• indirect communication through a more or less sophisticated blackboard (called also space)
Back to architecture of control

• Can we use the same modularity for implementation of one player? (i.e. on lower level)
• Can we organize internal modules of one player in similar way as cooperating players in the team?
• Can we use multi-agent modularity for building of control?

Yes, we can
Agent – Space architecture

We transform:

• modules to agents
• links among modules to indirect communication via blocks in space (on blackboard)
Indirect communication

Agents:
• can read, write or delete particular blocks in space
• know nothing about other agents, just know names and structure of the blocks they manipulates with
• perform their code on timer and/or trigger (a change of selected blocks in space)
Indirect communication

Details of read, write and delete operations are:

• no method for block creation
• reading of non-existing blocks is handled by returning a default value specified by reader
• value stored in block can have a limited time validity specified by writer; after its expiration the block becomes automatically empty
• value stored in block can have a priority specified by writer; such value overwritten only by value with same or higher priority
• space has no knowledge about value meaning; the reader is responsible for correct interpretation
package com.microstepmis.agentspace.demo;
import com.microstepmis.agentspace.*;

public class Agent1 extends Agent {
    int i = 0;

    public void init(String[] args) {
        attachTimer(1000);
    }

    public void senseSelectAct() {
        System.out.println("write:" + i);
        write("a", i++);
    }
}

public class Agent2 extends Agent {
    int i;

    public void init(String args[]) {
        attachTrigger("a");
    }

    public void senseSelectAct() {
        i = (Integer) read("a", -1);
        System.out.println("read:" + i);
    }
}

public class Starter {
    public static void main(String[] args) {
        new SchdProcess("space","com.microstepmis.agentspace.SpaceFactory", new String[]{"DATA");
        new SchdProcess("agent1","com.microstepmis.agentspace.demo.Agent1", new String[]{});
        new SchdProcess("agent2","com.microstepmis.agentspace.demo.Agent2", new String[]{});
    }
}
Data flow many:many

- each block can be written by many producers and read by many consumers
- consumers do not know how much producers generates the value or from whom the read value is coming
Time validity

• When we have more producers, neither of them can write “bad values”
• Rather such producer does not write a value at all
• But then it can happen that an old value persists in space and it is taken by consumers as valid
• Ideal solution is to define time validity for any written value. After its expiration, the value disappears from space (without agent intervention)
• Thus it can happen that block is empty, so consumer have to handle this state. Ideal solution is to use a default value specified when consumer calls read operation
Implicit sampling

- Since write operation overwrites data stored in a block regardless their consumers have undertaken them or not, any data flow is inherently (potentially) sampled.
Why no packages

• What to do if data are too frequent to be communicated one by one? What to do if no data can be lost by implicit sampling?

• One solution: turn blocks to queues, process data in packages

• Problem 1: not clear semantics of more producers

• Problem 2: more complicated processing (additional loops)

• Solution: Rather special triggers than packages, or multiply blocks
**Soft crash landing**

- each agent can be restarted without impact on system operation, mainly if they have no inner state (rather they can have analogical information in space)
- thus we can easily to add subsystem which starts crashed agents again and thus provide recovery from errors
- (each application specific code is concentrated in agents, space is independent from application domain)
Subsumption

- a design principle of control which mimics simplified biological evolution
- any complex control has an origin in a simpler ancestor
- descendant mechanism subsumes the mechanism of its ancestor
- higher levels rather inhibit and regulate than active the lower levels
Subsumption

**Question:** How could the newer levels influence the older ones? The older levels have been designed for particular use and have no interfaces for future development!

**Answer:** they have to have modular structure which enables it!

**Solution:** concept of indirect communication is suitable to provide that
Priorities

- However, blocks need to be associated also with priorities

```
write  read
    ↓
write(prio)  read
        ↓
write(prio+1)  read
              ↓
delete(prio+1)  read
```

- monitoring
- suppression
- inhibition
Integration of traditional AI

Agent modularity
- perfectly separates codes of modules
- enables to integrate slow modules into the system

Thus it is suitable to integrate slow cognitive structures on higher levels which subsumes fast but just reactive processing on lower levels
- reinforcement learning
- neural networks
- rule-based systems
<table>
<thead>
<tr>
<th>Comparison to iConnect</th>
<th>MultiConnect</th>
<th>SignoGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent (or several agents)</td>
<td>Block</td>
<td>Real-time, combining slower with faster - OK</td>
</tr>
<tr>
<td>No data packages</td>
<td>Data packages only</td>
<td>Data packages only</td>
</tr>
<tr>
<td>Implicit sampling</td>
<td>Explicit sampling</td>
<td>MultiCom</td>
</tr>
<tr>
<td>Time validity of data</td>
<td>Timestamp only</td>
<td>MultiCom</td>
</tr>
<tr>
<td>More producers of block, priority</td>
<td>Multiplexer, ?</td>
<td>MultiCom</td>
</tr>
<tr>
<td>Recovery from errors</td>
<td>?</td>
<td>MultiCom</td>
</tr>
<tr>
<td>Any data structures</td>
<td>Arrays of basic types</td>
<td>MultiCom</td>
</tr>
</tbody>
</table>
An example

A mobile robot following a ping-pong ball
Traditional pipeline

Camera → BW → Sobel → Threshold → Filter -izol → Filter -thin → Filter -prune → Edges → Hough -circle → Logic → Motor
Competition among more thresholds
pseudo-optimal threshold calculation
more balls in scene

Detector a

circle-a1
circle-a2
circle-a3
circle-a4

Detector b

circle-b1
circle-b2
circle-b3
circle-b4

Logic

Detector c

circle-c1
circle-c2
circle-c3
circle-c4

follower
concentration on particular ball protected to occlusion
combining ball following with obstacle avoidance
(looking for a ball when no ball is present)
Thank you for attention!

Multi-agent approach to control

Andrej Lúčny
MicroStep-MIS & DAI FMFI UK Bratislava
andy@microstep-mis.com
http://www.microstep-mis.com/~andy