Agent – Space Architecture

Andrej Lúčny

Department of Applied Informatics, Faculty of Mathematics, Physics and informatics, Comenius University, Bratislava

andy@microstep-mis.com, www.agentspace.org, www.microstep-mis.com/~andy

Introduction

Agent-Space architecture is a software tool for building control systems of robots or virtual models which modularity is based on decentralization and massive parallelism. It follows Minsky's Society of Mind and Brooks' subsumption architecture, mainly idea how higher-level modules can regulate activity in the system by influence of mutual communication among lower-level modules. Unlike its points of origin, the architecture overcomes limitations of hardware-layout fashion due to concept of indirect communication among agents similar to Gelernter's LINDA space. Moreover it concerns real-time operation as crucial and supports it by several original mechanisms. In this way it properly puts together fast and slow processes and enables very complicated data flows among them.

Control as a set of agents

Agent-Space architecture approaches any control as a set of agents where the proper behavior of the whole system is achieved by proper activities of individual agents at the proper instants of time (Fig. 1). The overall behavior of the system is dedicated to emerge from mutual interaction among the agents.

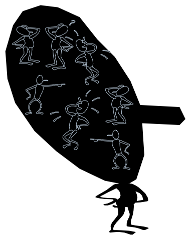

Figure 1.

How the agent works

Each agent has own control and performs own code, running in endless cycle (Fig. 2). Each course through the loop is dedicated for computing of appropriate actions upon perception of the situation and internal state. Unlike neural networks, here it is possible to specify how each agent contributes to the overall behavior.

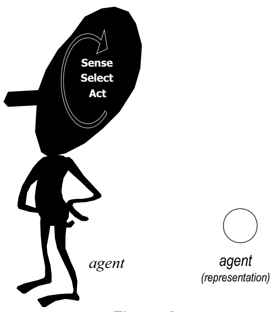

Figure 2.

The major part of agent perception and action resides in mutual data exchange among agents. The rest is tied with sensors and actuators.

How the agents communicate

Agents are put together into a system by mutual indirect communication which is provided by an advanced blackboard called *space* (Fig. 3). *Space* can contain named data of various forms (called blocks) and agents are able to read, write or delete them. Agents have to know name and form of blocks they manipulate. In special case, it is enough to know a mask to manipulate more blocks which names are matching the mask.

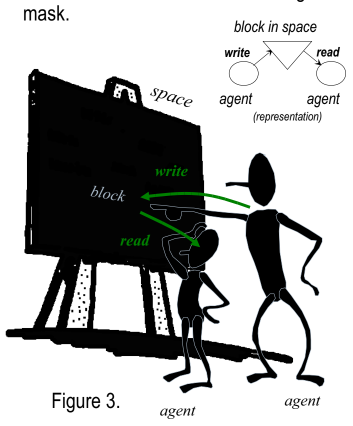

Figure 3.

Implicit sampling

Several details are important to enable modeling with the architecture as easy as possible. Mainly there is no special operation for creation of a block in *space* - blocks are physically created by the first write operation. However, they can be read even before this moment, throwing no exception in the system: for such case agents ought to define a default value for each read operation which is returned as a result if the read block has contained no value yet. Blocks can contain only one value; thus the value written by one write operation is overwritten by the next write operation. The value is overwritten regardless an agent has read the value or not. Thus when a producer writes a next value, before the former value is read by a consumer, the former value is simply lost. Thus the values are implicitly sampled (Fig. 4). Due to the sampling it is not possible to overwhelm the system. It is also easy to combine slow and fast modules.

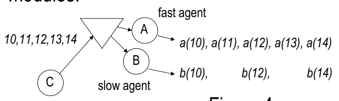

Figure 4.

Besides the manipulation by agents, *space* provides to blocks two additional features: their time validity and priority.

Time validity

Time validity can be defined when agent calls write operation. At the moment, it can specify the period after which the written value automatically expires. Otherwise the value is valid until it is rewritten. Time validity is a good mechanism for expression of tentative character of information.

Data flows many:many

Unlike traditional wiring of one output to several inputs (1:many), we enable to establish data flows from many producers to many consumers (Fig. 5).

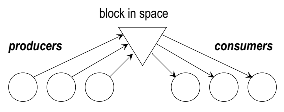

Figure 5.

Making this relation clear, let us imagine that you have several producers of the same value. However, each of them produces the value under certain (and different) conditions only. In this situation we recommend to let all the producers to write their proposals for the value to the same block. Thus consumers do not need to reflect to the fact that we have more producers, they simply undertake the value stored in the common block. Just when conditions disable a producer to compute the value, producer must not write a value "unknown" to the block as it could overwrite useful value computed by another producer. As a result, the block value is not changing when conditions are so unfavorable that no producer can propose the value. After a period, consumer could undertake too old value. This problem can be solved easily just by use of time validity.

Priorities

Priorities enable us to prefer value provided by a particular producer. E.g. such producer could produce more accurate value than others. Priority can be defined when agent writes the value to *space*; default priority is concerned otherwise. When other agents try to overwrite the value by another one, the overwritting is physically performed only if the priority of the new value is not lower then priority of the former value.

Hierarchy

It is necessary to underline that agent whose write operation is ignored gets no information about the fact. This is important to enable higher levels of the system to influence operation of lower levels (Fig. 6) without invocation of an exception which could stop operation of the whole system.

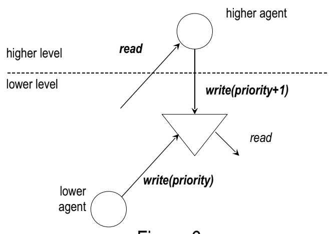

Figure 6.

Simultaneous write operations and priorities enables us to implement idea that higher levels of control rather regulate than activate lower levels: an agent in the higher level can read some blocks in lower level and - at a proper moment – it can overwrite them.

Real-time operation

Real time operation is partially reflected by the fact that in real implementation the course through agent loop consumes certain time which differs from agent to agent. However the major real-time feature is blocking. It means that agent usually sleeps and performs one course through its loop just when a notification is received. The notification is based on two different mechanisms.
The primary mechanism is notification by timer (Fig. 7, on the left), which can be set up during agent initialization to provide regular notification with a given time period. As a result, propagation of events can be delayed little bit by any participating agent. Such system can tentatively exhibit inconsistent relations among propagating values, but mostly tends to achieve a consistent state.

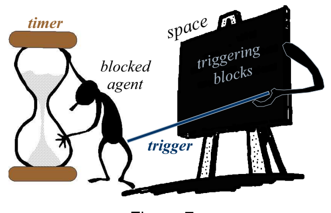

Figure 7.

The secondary mechanism is notification by trigger (Fig. 7, on the right). Trigger is a notification provided by *space* when a particular block (or one block from a given group of blocks) is changed. Thus immediate reaction on a stimulus is also possible.

Conclusion

Due to its character Agent-Space architecture enables to establish decentralized control structures with simple layout of modules and complicated data exchange among them without danger of livelock or deadlock. We tested the architecture on several robots and virtual models.

An Example

Robot following ping-pong ball under various conditions (various lighting, more balls, occlusion of the ball) (Fig. 8)

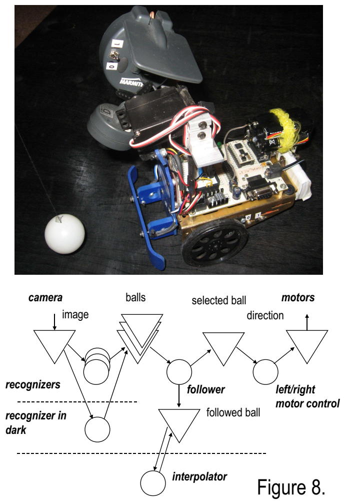

Figure 8.