Parallel Programming for Mobile Robot Control: Agent-Based Approach

Stevo Bozinovski
Electrical Engineering Faculty
Karpos II, 91000 Skopje, Macedonia

Abstract

Some issues concerning agent-based systems and parallel programming are discussed. A practical problem of designing a mobile robot for educational purposes given IBM Series/1 EDX is considered, and as a solution, a multiagent control system is designed and implemented using the EDX multitasking system. A discussion concerning the notion of agents and an agent society is also provided.

1. Introduction

1.1 Presentation logic

There are several research streams around the issue of distributed systems. These include parallel programming [1], communicating modules [23], actors [12], software primitives for synchronization [10], graphical representations including Petri nets [7], distributed software engineering [19], multicomputing systems [21], neural networks and parallel distributed processing [18], encapsulated software objects [22], distributed artificial intelligence [11], negotiation [9], contract nets [20], cooperation and conflict resolution [24], project and enterprise integration [8, 16], open systems [14], and computational ecosystems [15]. This list is far from exhaustive.

From the practical point of view, however, our main concerns in solving real-world problems involve such questions as how to model the problem that exhibits concurrent activities, how to deal with synchronization and communication among distributed parts, and which computer and which programming language to use for solution implementation.

Based on these observations, the logic that will drive the presentation in this paper has two parts:

1) Reasoning about the system: We will consider and propose a *general metaphorical framework* that could cover the issues surrounding the programming of distributed systems.

2) Reasoning within the system: We consider an *implementation:* design a mobile robot using a parallel programming system. We will deal with this problem as we view the problem using the proposed framework. In other words, we are using the inheritance principle: "big picture" notions are inherent in the practical, real-world problem.

1.2 A metaphorical framework: Society of software beings

The *agent society framework* [2, 3] is used in this work. In this view, agents can be considered as a form of *software being.* The set of agents activated in a system forms an *agent society,* one form of which is an *agent network.* In such a society, the agents have certain roles, some of them being managers. An important feature of agents is that they act according to goals. Agents *communicate* among themselves according to some *order* in their society.

This view is consonant with that of other researchers in the area who have undertaken a global view. Zlotkin and Rosenschein [25] consider the agent to be an individually motivated system that is usually acting rationally only when it maximizes its own utility. Pan and Tenenbaum [16] point out that the essence of being an intelligent agent is being "alive," that is, responding autonomously and proactively to perceptions. They also point out that there is a great temptation to anthropomorphize intelligent agents and think of them as "electronic humans." Zhao and Natarjan [24] use agents as elements of cooperative systems for achieving an overall goal. Hogg and Huberman [15] stress that agents are designed not only to cooperate with each other, but also to adapt appropriately to global consequences of their actions; they also introduce the notion of the computational ecosystem as a framework for studying global behavior of distributed systems. Durfee [11] notes that the "society of systems" is a more powerful strategy for distributed artificial intelligence (DAI) than injecting more knowledge into a system. Hewit and Inman [14] point out that DAI must embrace social and anthropological concepts if it is to bring AI ideas into human organizations.

1.3 An implementation: Software system for mobile robot control

From 1982 to 1986 we carried out the ADRIEL (ADaptive Robot of the Institute of ELectronics) project: to design and construct mobile robots for

educational purposes. The computer and the software system *were given:* the only computer available to us and capable of coping with such a task at that time was the IBM Series/1 computer with its Event Driven Executive (EDX) operating system. The principal programming language was a multitasking programming language, the Event Driven Language (EDL).

Here is the formulation of the problem: design a mobile robot control system given the Series/1 computer as a control computer, and explain the design process to students.

The work on this problem has led us to solutions that are described below. After giving an illustration of an EDL code and the graphical representation technique we developed and used, we describe the multiagent architecture we developed and implemented.

2. Event Driven Language: A parallel programming system

2.1 The language

As has been pointed out by Shatz and Wang [19], many distributed programming languages have been proposed, including Ada, Argus, Concurrent C, Communication Sequential Processes (CSP), Distributed Processes (DP), Linda, Modula, Nil, Occam, Programming Language in the Sy (PLITS), and Synchronizing Resources (SR). Only a very few have become commercially available (including Ada, Occam, and Concurrent C); the others are primarily "paper languages" or research prototypes. The theoretical bases for these programming languages are such concepts as semaphores, monitors, remote procedure calls, message passing, and guarded commands [1].

It is interesting that, in distributed programming literature, EDL is not usually mentioned as a parallel programming language, though its synchronization primitives, WAIT/POST and ENQ/DEQ, were proposed in sixties [23] and implemented in the IBM OS/360 operating system. In this paper we would like to illustrate the EDL language and its synchronization primitives to the broader distributed computing audience.

The EDL system distinguishes two types of entities that can run in parallel: programs and tasks. A program is an executable module; it is compiled separately and stored as a file. It is called with the load command. A task is a module that is compiled along with the main program and is called with the attach command. Once called, both programs and tasks run in parallel with the main (calling) program. The main program thus plays the role of an application operating system for the application it represents.

```
MANAGER   program
          A
          load      MAN1
          load      MAN2, event=m2_ok,
                          error=abort
          attach    AGENT1
          B
          wait      agent1_message
          C
          wait      m2_ok
          attach    AGENT2
          D
          wait      agent2_done
          wait      agent1_done

          E
abort     progstop

AGENT1    task event=agent1_done
          F
          post      agent1_message
          G
          enq       RESOURCE
          H         {critical section}
          deq       RESOURCE
          I
          endtask

AGENT2    task event=agent2_done
          J
          enq       RESOURCE
          K         {critical section}
          deq       RESOURCE
          L
          endtask

RESOURCE  qcb
database  data      dim=5000
                    {other data definitions}
          endprog
          end
```

Figure1. An EDL main program that manages parallel execution of two programs and tasks.

Figure 1 is an illustrative EDL main program. Parallel execution of programs and tasks, synchronization

between them, and mutual exclusion in accessing a critical resource are shown. Irrelevant segments of the code are denoted by A, B, C, and so on. The program MANAGER manages the multiprogramming of two programs, MAN1 and MAN2, and the multitasking of two tasks, AGENT1 and AGENT2. The tasks are in competition for a resource database controlled by a queue control block, RESOURCE. Two types of synchronization coding primitives are used: **event** for successful completion of a task or a program and **post** for signaling any other relevant event for which a segment of the main program or task is in a **wait** state. In this example the program MAN1 runs asynchronously, and the program MAN2 monitors whether MAN1 is successfully loaded (if not, the whole system aborts) and whether it successfully terminates. The critical sections of the two competing tasks are controlled by the **enq/deq** program primitives.

The program given in Figure 1 uses the labels MANAGER, AGENT1, and AGENT2 to implement an agent-based system using multitasking software.

2.2 Graphical primitives for multitasking programming

Let us now consider the problem of representing parallel programming. When representing sequential programming, one usually mentions structural graphical primitives such as SEQUENCE, IF-THEN-ELSE, DO-WHILE, and others. One problem in parallel programming is the lack of an appropriate graphic symbolism to represent the parallel programming and synchronization primitives. Although Petri nets are used to model parallel programming [17], they do not correspond directly to the programming primitives used in intertask communication. Here we describe the graphic programming technique [3,4,6] that we used for design purposes and for teaching parallel programming to electrical engineering students. A similar effort was made by Gomma [13].

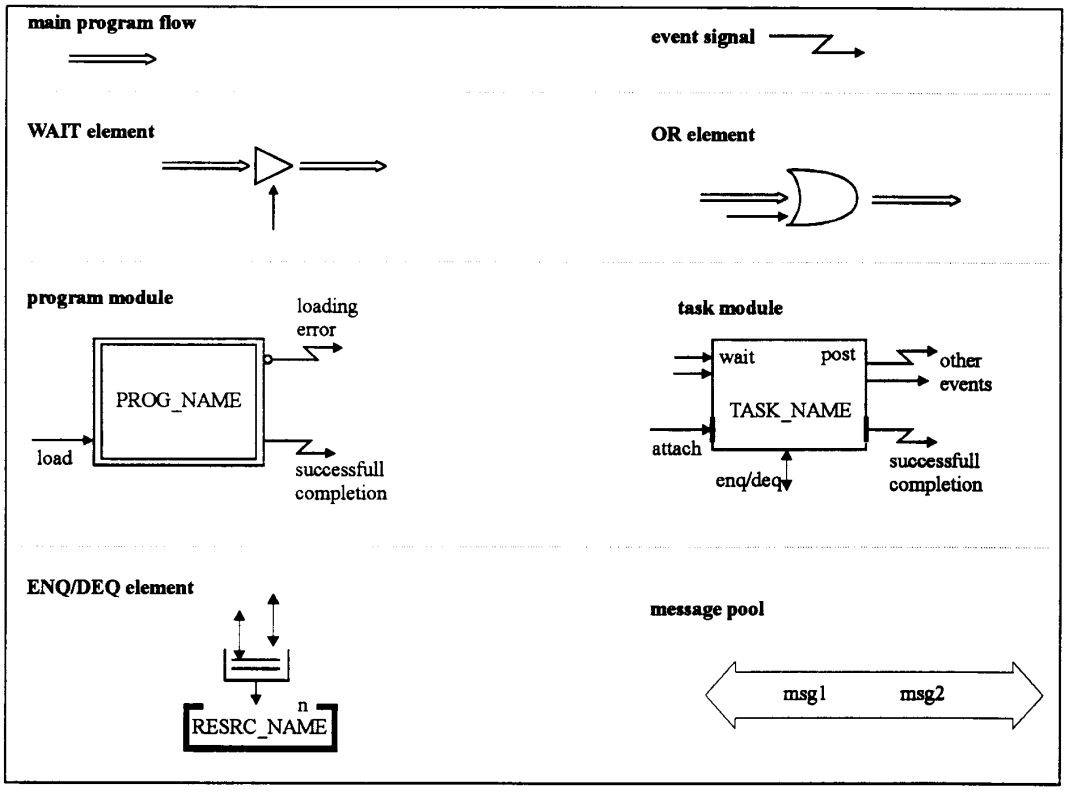

Figure 2. The basic LDPP elements

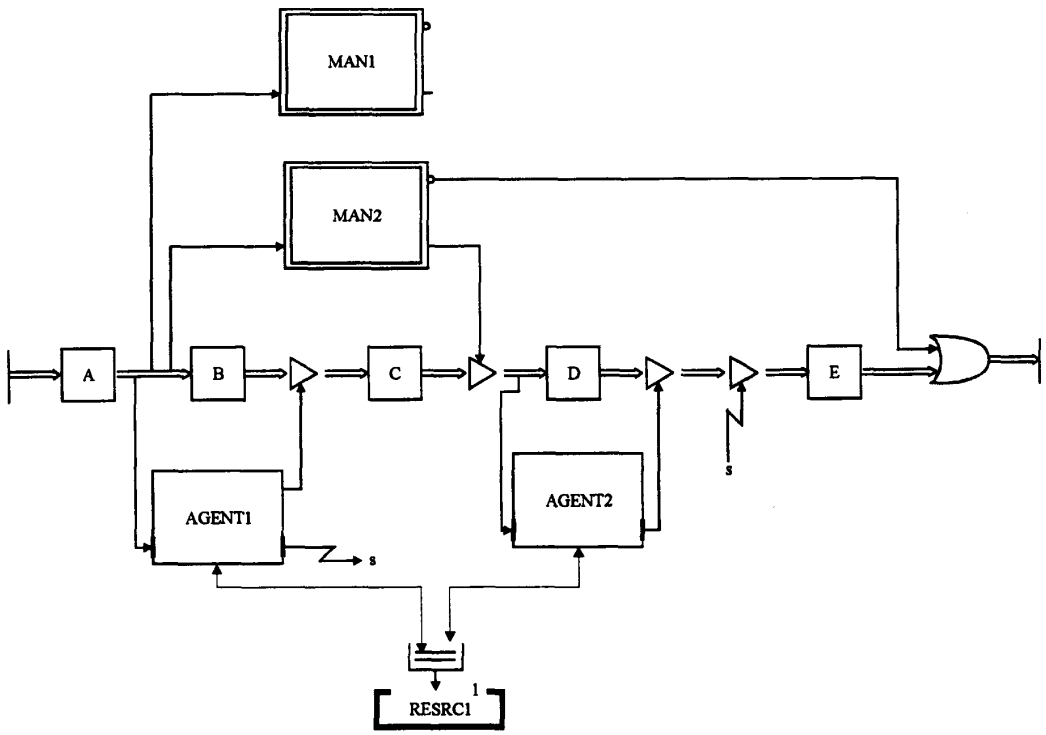

Figure 3. The LDPP representation of the multitasking programming structure shown in Figure 1

Figure 2 shows the graphic elements we use in a technique we call *logic design parallel programming* (LDPP) representation. As shown in Figure 2, in LDPP we take advantage of the graphic primitives used in logic circuit design, such as OR gates, three state buffers (here WAIT elements), and sequential circuits. This helps the understanding of software parallel processing in general, and shows the resemblance to parallel processing in hardware logic circuits, where parallelism is an inherent feature. For educational purposes, this is an advantage of this technique over other techniques.

Figure 3 shows the LDPP representation of the example given in Figure 1. The main program segments, together with WAIT and OR elements, build the synchronizing backbone to which parallel modules are attached at the moments of their creation. The signal s is distinguished for better visualization; the other signals are drawn with continuous lines. The resource module has only one kind of element. For animation purposes the lines can be drawn dynamically to show the status of the processes and of the whole system.

3. A multiagent system for mobile robot control

Given a multitasking parallel programming language, it was natural to build a modular design for mobile robot control in which modules can be represented as EDL tasks [6]. A similar approach taken by Brooks [7] had modules programmed in LISP.

The hardware we used consisted of (1) the Series/1 computer, (2) an analog/digital converter that was part of the Series/1 sensory unit, (3) an interface consisting of current amplifiers, and (4) an electrically controlled toy car for which we built an appropriate "outfit" to make it look like a factory vehicle.

The experimental verification was provided using the design shown in Figure 4: The mobile robot, searching for a goal (a place at which a voltage is applied), makes its way along a path and eventually stops when the goal is achieved. The robot has only tactile sensors—in front, in back, and at its left side.

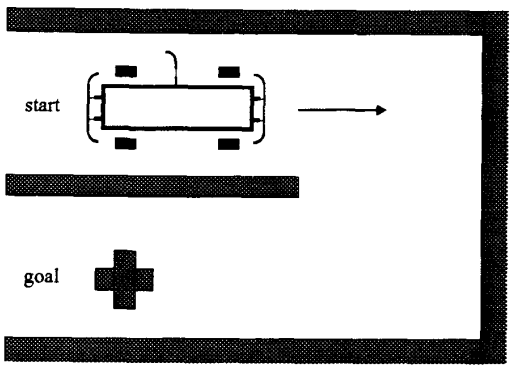

Figure 4. Experimental design for mobile robot control

For this system we designed a multiagent network and implemented it as a multitasking structure. Figure 5 shows the solution.

The agent network in Figure 5 consists of four agents: agent SENSE (goal: send information about the SITUATION), agent TIME (goal: signal that the time, TIM, is elapsed), agent TURN (goal: achieve and keep the direction DIR), and agent DRIVE (goal: drive the robot with a velocity VEL). The agent MANAGER synchronizes the agent network.

When activated (by the **post** signals go3 and go4), the agents SENSE and TIME have the local goal of supplying information about the SITUATION. To avoid a writing collision, the SITUATION is considered as a resource so that mutual exclusion is ensured. The agent TURN receives the parameter DIR (direction) via the logical intertask bus; when activated (**post** go1), its goal is to keep DIR implementing the servo mechanism. The agent DRIVE receives the parameter VEL (velocity), and its goal is to keep VEL for time TIM.

The system has proved to be successful in solving the described task. It was used for demonstration purposes for students in courses on systems software (including parallel programming) and robotics.

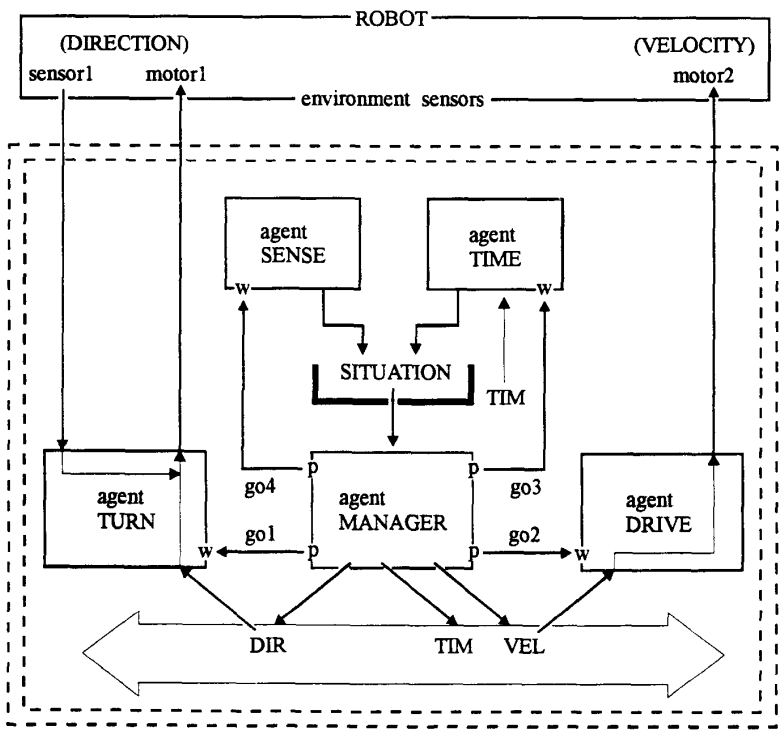

Figure 5. A multiagent system for mobile robot control

```
MS          program
            attach      S1          { create
            attach      S2            agents
            attach      MANIP         as
            attach      MOBIL         tasks                    }
            post        MOBIL_standby { agent MOBIL
            post        S1_on           goes toward S1
            post        MOBIL_go        using dead reckoning
            wait        S1_arrived      and after arriving
            post        S1_off          S1 is turned off       }
unloading   wait        MOBIL_adjust  { after adjusting
            post        MANIP_go        the agent MANIP
            wait        MANIP_done      unloads agent MOBIL    }
nextloc     post        MOBIL_turn    { leaving the
            post        MOBIL_standby   unloading place,
            post        S2_on           using dead reckoning,
            post        MOBIL_go        the agent MOBIL
            wait        S2_arrived      navigates toward S2
            post        S2_off          and eventually
            progstop                    arrives                }
            endprog
            end
```

Figure 6. The agent manager routine for an FMS scenario

4. Extension to multirobot system: An FMS scenario

In this section we will discuss the application of our agent-based architecture to a multirobot system. Consider the following flexible manufacturing system scenario, which was used in the ADRIEL project [5].

Four players are in this scenario: two light sources S1 and S2, a 2-degree-of-freedom (DOF) mobile transporter T capable of dead reckoning according to a light source, and a 3-DOF manipulative robot M. The scenario consists of three basic parts. First, the loaded transporter T goes toward the illuminated S1. The environment setting enables the proper positioning of T. Next, the manipulative robot M (placed at S1) unloads T. Finally, the unloaded T goes toward a new place, the illuminated S2.

For this problem we need an FMS agent manager and four agents: MANIP, MOBIL, S1, and S2. The agent manager routine is given in Figure 6.

From the program code in Figure 6 it is clear that the same program code can be used if we consider not only agents but also, for example, objects, monitors, or remote procedure calls as software concepts. However, we believe that all those notions are forms of the notion of "software being." It seems to us that the notion of agent as an adaptive, goal-seeking, communicating entity is a form closer to the notion of "being" at this point of our understanding, and that is why we are using it.

5. Conclusion

The notion of an agent-based system is important in software engineering, parallel distributed processing, and AI. It leads toward the notion of software beings.

Implementing the notion of an agent society in the contemporary state of the software technology is manageable, using concepts already implemented in the programming languages such as tasks and objects. This paper describes multiagent systems being implemented as multitasking systems for robot control.

It is our belief that as neural network paradigms established a link between parallel computation and biology, the agent society paradigm is establishing a link between parallel computation and sociology.

Acknowledgments

The author expresses gratitude to Dr. Wei Zhao and

anonymous referees for valuable comments and suggestions, as well as to Sasko Stefanovski for technical assistance in the preparation of the paper.

References

[1] Andrews, G., Schneider, F. "Concepts and notations for concurrent programming," *ACM Computing Surveys,* Mar. 1983.

[2] Bozinovski, S. "Guest editor's introduction," *Automatika: Special Issue on Biological and Nonbiological Beings,* Zagreb, 1985.

[3] Bozinovski, S. "Implementation of multiagent cooperation in programming for robot control," in P. Gabko, P. Kopacek, and M. Voicu (eds.), *Proc. Workshop on Computer Science Topics for Control Engineering Education,* Vienna, 1993.

[4] Bozinovski, S. "A representation and visualization of parallel and structured programming," *Proc. Symp. Inform. Technol.,* Sarajevo, 1988.

[5] Bozinovski, S., Koco, I., Hristofi, A. "A model of multirobot system supervising control in a flexible manufacturing system" (in Macedonian), Proc. Conf. JUROB, Opatija, 1985.

[6] Bozinovski, S., Sestakov, M. "Multitasking operating systems and mobile robot control" (in Macedonian), *Proc. Workshop on Information Science in Macedonia,* Skopje, 1983.

[7] Brooks, R. "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation,* Mar. 1986.

[8] Cutkosky, M., Engelmore, R., Fikes, R., Genesereth, M., Gruber, T., Mark, W., Tenenebaum, J., Weber, J. "PACT: An experiment in integrating concurrent engineering systems," *IEEE Computer Magazine,* January 1993.

[9] Davis, R., Smith, R. "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence,* 1983.

[10] Dinning, A. "A survey of synchronization methods for parallel computers," *IEEE Computer Magazine,* July 1989.

[11] Durfee, E. "The distributed artificial intelligence melting pot," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.

[12] Ferber, J., Carle, P. "Actors and agents as reflective concurrent object: A MERING IV perspective," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.

[13] Gomma, H. *Software Design Methods for Concurrent and Real Time Systems,* Addison-Wesley, 1993.

[14] Hewitt, C., Inman, J. "DAI betwixt and between: From 'intelligent agents' to open systems science," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.

[15] Hogg, T., Huberman, B. "Controlling chaos in distributed systems," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.

[16] Pan, J., Tenenbaum, J. "An intelligent agent framework for enterprise integration," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.

[17] Raeder, G. "A survey of current graphical programming techniques," *IEEE Computer Magazine,* Aug. 1985.

[18] Rumelhart, D., McClelland, J. *Parallel Distributed Processing,* MIT Press, 1986.

[19] Shatz, S., Wang, J-P. "Introduction to distributed software engineering," *IEEE Computer Magazine,* Oct. 1987.

[20] Smith, R. "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Computers,* Dec. 1980.

[21] Tanenbaum, A., Kaashoek, F., Bal, H. "Parallel programming using shared objects and broadcasting," *IEEE Computer Magazine,* Aug. 1992.

[22] Wegner, P. "Dimensions of object-oriented modeling," *IEEE Computer Magazine,* Oct. 1992.

[23] Witt, B. "Communicating modules: A software design model for concurrent distributed systems," *IEEE Computer Magazine,* Jan. 1985.

[24] Zhao, W., Natarajan, S. "Cooperative resource management in R-shell," *Proc. Complex Systems Engineering Synthesis and Assessment Technology,* Silver Spring, 1992.

[25] Zlotkin, G., Rosenchein, J. "Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains," *IEEE Trans. Systems, Man, and Cybernetics,* Nov./Dec. 1991.