Prof. dr hab. Adam Borkowski
Mgr Dorota Daniecka
Mgr Michał Gnatowski
IPPT PAN

MODELLING CO-OPERATION OF MOBILE ROBOTS BY MEANS OF OBJECT-ORIENTED LANGUAGE[1]

Paper presents preliminary results on modelling co-operation of multiple mobile robots. Firstly, several tasks that require such a co-operation and at the same time are of practical importance are described. Then general data structure is proposed for dealing with situations characterised by distributed intelligence and need to co-ordinate actions both in space and in time. The proposed scheme is implemented in heterogeneous computer network environment using C/C++ in the real time part and Java in the simulation part.

MODELOWANIE WSPÓŁPRACY ROBOTÓW MOBILNYCH ZA POMOCĄ JĘZYKA OBJEKTOWO ZORIENTOWANEGO

Praca przedstawia wstępne wyniki badań nad modelowaniem współpracy wielu robotów mobilnych. Omówiono w niej szereg zagadnień, które wymagają tego typu współpracy a jednocześnie są ważne z praktycznego punktu widzenia. Zaproponowano ogólna strukturę danych odpowiednia dla sytuacji charakteryzujących się rozproszeniem inteligencji i koniecznością koordynowania działań zarówno w przestrzeni, jak i czasie. Schemat ten zaimplementowano w niejednorodnym środowisku sieciowym korzystając z języka C/C++ w części dotyczącej sterowania w czasie rzeczywistym i języka Java w części symulacyjnej.

1. INTRODUCTION

Recently much attention has been paid in Computer Science to the agent-based approach [6], [8], [11], [13], [14], [18]. Software agents are defined as self-contained programs capable of controlling their decision making and actions based on perception of environment in pursuit of one or more goals [18]. Thus, the main features of agents are:

- autonomy – the ability to act independently of human supervision;
- reactivness – the ability to respond to changing environment;
- pro-activeness – the ability to perceive goals and to take initiative for achieving them.
- social ability – the ability to co-operate with other agents and/or humans;

Some authors [9], [11] supplement this list by the ability to learn which seems to be quite reasonable. It is easy to observe that control systems of mobile robots, as described, e.g., in [1], meet three first requirements. Hence, multi-agent approach in mobile robotics refers to systems that are additionally capable of social behaviour and possibly of learning.

[1] Supported by the Committe of Scientific Research (KBN) under project No. 8T11A01312.

Given a group of agents their co-ordinated action can be achieved by a certain mechanism that lies in-between two extreme schemes:

- strict hierarchy – where subordinate agents obey the orders given by supervisors;
- full autonomy – where agents act on equal terms and come to common decisions through negotiations.

Each agent has its own knowledge, reasoning mechanism and possibly learning ability. In order to communicate with each other, agents should either possess a common high level language or interact synergistically in the common environment. The second possibility leads to interesting modelling of psychology of a crowd or insect-like behaviours [3]. It requires large amount of simple agents to interact and falls out of scope of the presented project.

2. POSSIBLE TASKS

Full autonomy is interesting from the cognitive point of view but very impractical: negotiations slow down the decision making process and make the resulting behaviour of the group unpredictable [17]. On the other hand, strict hierarchy leads to systems that do not learn and do not adapt itself to changes in environment. Hence, we adopted mixed approach: robots obey the hierarchical order shown in Fig. 1 but posses also a certain degree of autonomy.

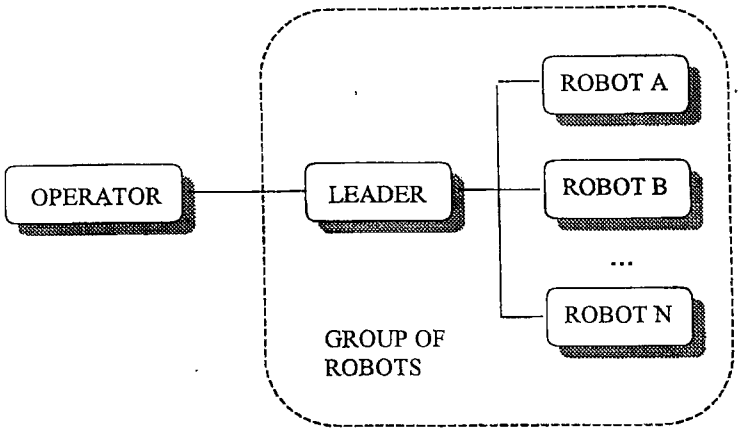

Fig. 1. Group of co-operating robots.

According to this scheme, human operator decides what *task* is to be accomplished, selects *group* of robots that should perform the task and assigns *leader* of that group. The leader obtains a *task order* from the operator, notifies members of the group about their *roles* in the *scenario* of the given task, assembles the group in an *initial configuration* and asks the operator for permission to begin execution of the task

After permission has been granted, robots perform autonomously their roles notifying on fly the upper level of the hierarchy about the current status. Thus, members of the group report to the leader and the leader reports to the operator. Supervising agent can interrupt the activity of subordinate unit if for some reasons the execution of current task is not carried out properly.

So far the following tasks were considered in the present project:

go(S, G, F) – go from starting position S to goal position G keeping formation F;

patrol(T, E) – patrol given terrain T notifying supervisor about events listed in E;

treat(T, P) – treat given terrain T according to prescriptions P;

move(O, S, G) – move given object O from starting position S to goal position G;

The first task has practical applicability both in civil and military area. The simplest formation is a *column*: robots follow the leader reproducing his trajectory and speed and keeping prescribed distance between each other. The leader plans a collision-free path between S and G, each robot avoids in reactive manner unexpected obstacles and tries to recover his position in the formation after such a manoeuvre. Note that leading manufacturers like Daimler-Benz AG already offer automatic driving in the column formation. This mode of traffic is also adopted in the project on automated highway currently under development in the USA [15]. A *row* formation is useful when robots should sweep certain area. It is more difficult to preserve under the presence of obstacles.

Patrolling given area has obvious military applications. It is also the main task for mobile robots that serve as inspectors of large industrial objects. The list of monitored events E can include then smoke, chemical contamination, increased radiation, etc. Very interesting problems occur when programming sentries that should guard the object against intruder. A group of such automatic guards should plan their path in such a way that no moving object could escape detection.

Treatment of large surfaces like cleaning, painting, polishing, etc. finds many applications in shipbuilding, construction engineering and maintenance of buildings. The simplest prescription P is for cleaning: several automatic mobile vacuum cleaners are available already on the market. Situation gets worse in painting where robot should not enter freshly painted area. Automatic planning of such a treatment has been subject of the paper [2].

Many authors have already considered co-operation of multiple robots that try to move an object exceeding their individual load carrying capacity. The lack of grippers on our robots precludes us from exploiting this task properly. We restrict ourselves to considering a somewhat simplified problem: two Pioneers trying to carry a stick laid on them. This problem will be discussed in the other paper presented by our group at this conference [4].

3. DATA STRUCTURE

Well-known advantages of the Object-Oriented Programming (OOP) motivated us to use this methodology. The top level of data representation consists of abstract classes representing generic objects under consideration: *Task, Group, Robot, Sensor* and *Map*. The diagram of classes shown in Fig. 2 follows the Unified Modelling Language (UML) notation [16]. Abstract classes contain templates of objects that can be further specified at lower levels.

For example, the abstract class *Robot* contains instance variables defining current state of robot:

x, y, fi – pose (cartesian co-ordinates and orientation);

v, vfi – linear and angular velocity;

a, afi – linear and angular acceleration;

etc.

and abstract methods defining abilities of each robot:

givePose ( ) – returns current pose;

moveAhead (a, v) – drives distance *a* with average velocity *v*

turnLeft (theta) – turns *theta* degrees counter-clockwise

turnRight (theta) – turns *theta* degrees clockwise

etc.

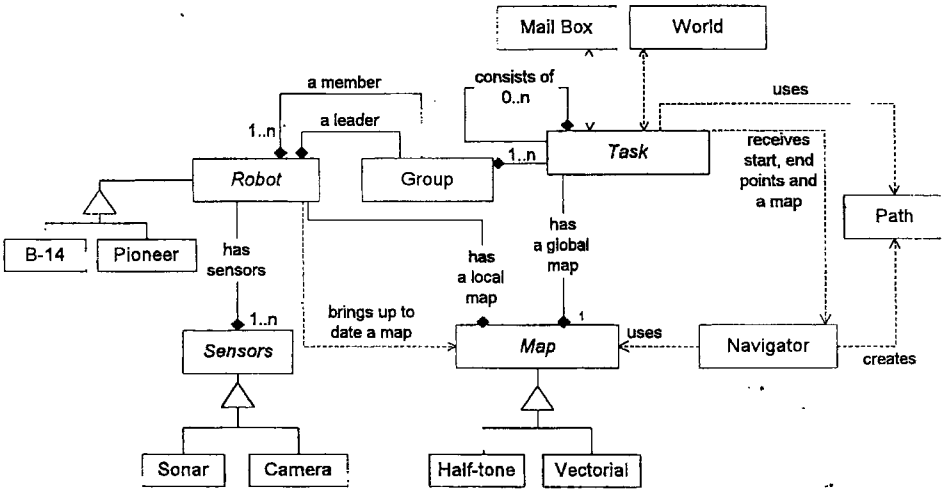

Fig. 2. Diagram of classes and their relations.

The abstract class *Group* contains *Leader* and *Member* as instance variables of type *Robot*. It also has integer variable *Size* and method *giveSize ( )* that refer to the number of robots building the group. The class *Sensor* has Boolean variable *isActivated* and the methods *turnOn ( )* and *turnOff ( )* that controll its activation. Common features of maps are defined in the abstract class *Map*. The lower level of representation contains classes that describe particular types of objects. Thus, *B14* and *Pioneer* are subclasses of *Robot* that allow instantiation: three Pioneer 1 units that are available at our laboratory constitute instances of the class *Pioneer*. They are distinguished by *name* – the instance variable of that class.

Similarly, specific sensors like contact panels, infrared beams, ultrasonic range detectors, laser scanners or CCD-cameras form subclasses of the generic *Sensor* class. Due to inheritance principle only specific features need to be defined at the lower level of representation. Modularity and data encapsulation increase the resistance of the system against errors and allow us to introduce new elements (tasks, robots or sensorial devices) in the future without changes in the general representation scheme.

4. COMMUNICATION AND CO-ORDINATION

A crucial element of any multi-agent system is the communication layer. Too scarce messages preclude intelligent behaviour whereas too big amount of exchanged information slows down the system and can make it even unsuitable for controlling multiple robots in real time. Several attempts to develop general Agent Communication Language (ACL) have been reported in the literature. On of them is the Knowledge Query and Manipulation Language (KQML) [5]. It introduces *ask, tell* and *reply* as basic communication primitives without restricting the internal contents of the message. Hence, a content language like, e.g., the Knowledge Interchange Format (KIF) [6] should supplement KQML.

Looking for a compromise we decided to allow for unrestricted asynchronous two-way communication between all agents (robots) but to simplify parsing of messages as much as possible. Thus, communication is network based and uses sockets. Each robot inherits *send( )*and *receive( )* methods from the class *Robot*. Their implementations for B14 and Pioneer units are obviously different. The mail server *MailBox* running as parallel process in the background supervises the flow of messages. A message can be either urgent, or regular and either persistent or accidental. Urgent messages like "stop" trigger immediate response of the agent. Persistent messages are send by an agent continuously in prescribed time intervals. At present only one message of this type has been implemented: each mobile robot broadcasts persistently his current pose. Accidental messages carry information about certain events that occur during the mission.

An accidental message has the following format: *(receiver; sender; code; [parameters])*. The receiver can be a particular agent or all agents involved in the current task. The numerical code indicates the contents of the message and optional parameters usually refer to the state variables of the robot. For example, message "go to" has the code 05 and 3 parameters defining the goal pose *x, y, fi*.

It assumed that initially all robots remain idle at certain poses and monitor the mail. The operator sends to the robot that will lead the group a message describing the task and the composition of the group. The leader plans the execution of the task and sends to the members of the group the initialisation messages. After all members responded with "ready", the leader asks operator for permission to start the group action. During the operation individual agents follow the scripts of their roles stored in the task database. Local disturbances are resolved in reactive manner.

5. IMPLEMENTATION ISSUES

Hardware configuration used in the present project is shown in Fig. 3. It consists of 2 Sun SPARC-driven units and 3 Pentium-driven PC's connected by a local Ethernet network. Robot B14 has Pentium-driven board computer but is connected to the network by wired serial link. Pioneer robots A, B and C have slower Motorola processors on board but communicate with their host PC's via radio modems.

Real World Interface, Inc. the manufacturer of B14 and Pioneer 1 mobile robots supplies software controlling these robots in 2 versions: for SPARC-driven Unix workstations and for Intel-driven PC's under Windows 95/98 or Windows NT. The user can customise it by including own C-functions into the system's library.

The distributed control system MULTI under development in the present project consists of 2 subsidiary parts. The part controlling real robots MULTI-R is implemented in C and dis-

cussed in detail in the parallel paper [4] presented at this conference. The part MULTI-S is a simulator of the same multi-robot environment. It is implemented in Java in order to ensure direct portability and lower cost of the development.

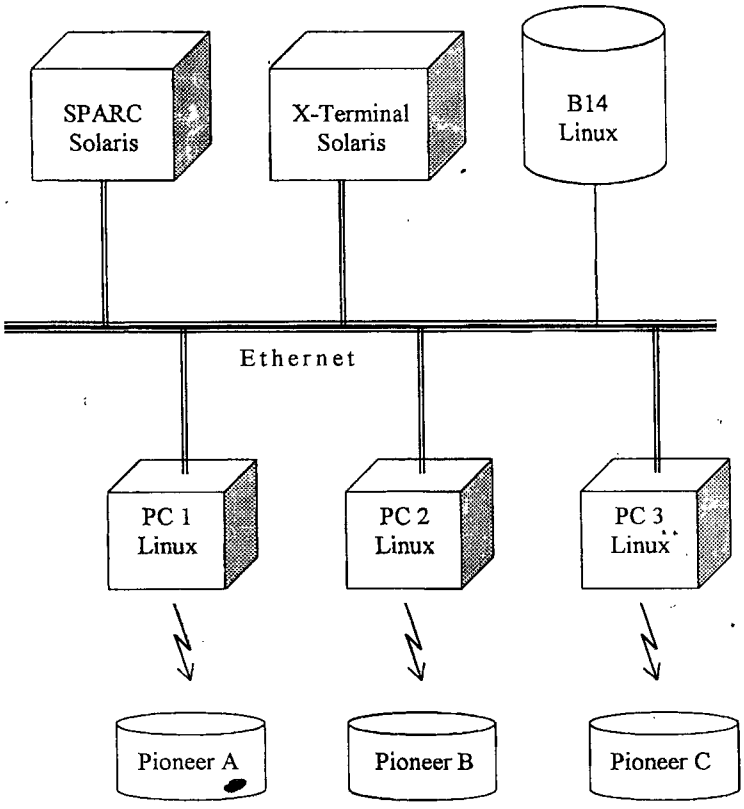

Fig. 3. Experimental setup at Laboratory of Adaptive Systems.

Java has been designed for rapid prototyping of software that should work in heterogeneous distributed environment. Retaining the elegant metaphor of OOP as introduced by Smalltalk it avoids difficulties and hazards that frustrate programmer in C++. Automatic memory allocation with efficient garbage collection, easy management of events, multiple threads and exceptions, simple but sufficiently rich toolkit for building Graphical User Interface (GUI) and finally off-the-shelf tools for communication in the network make programming in Java pleasant and efficient.

Full portability of Java aplets and applications is achieved by introducing the Virtual Java Machine — an interpreter that translates an intermediate language (byte code) into the machine code of particular processor. This makes Java programs slower than binary codes ob-

tained by compiling and linking programs written in C/C++. As the side effect of our project, we want to check experimentally how significant is this handicap and to what extent is Java superior to C/C++ at the development and debugging phase.

6. CONCLUSION

The adopted scheme of object-oriented data representation and rapid prototyping turned out to be very efficient. The portability of Java allowed us to work on any platform without recompiling and adjusting the code. The development of the real-time part of the system is much more laborious Many improvements and tests are still pending at present.

Experiments conducted so far confirm that the proposed control structure allows the group of 2–3 robots to perform co-ordinate manoeuvres. The main issue remains poor directional resolution of sonars that serve as main sensors for Pioneer robots. Further efforts will be directed towards incorporating computer vision based sensing that is already possible for B14 robot and will be available for next generation Pioneers.

The second bottleneck is the slow serial RS-link that is used at present for transmitting messages. Replacing it by Ethernet radio modem would radically improve the system. Probably much can be gained also by tuning the co-operation between the group leader and the subordinate members of the group. Results of such experiments will be reported during the Workshop on Mobile Robots (WMR'99) that will take place near Zakopane in September 1999.

REFERENCES

[1] Borkowski A., Dubrawski A., Siemiątkowska B.: *Navigation problems for mobile robots* (in Polish); Prace IPPT PAN, in print.

[2] Borkowski A., Zawidzki J.: *Region filling problem – knowledge representation and solution in PROLOG*; Archives of Machine Building, 39, 1992, 135–146.

[3] Brooks R. A.: *Intelligence without reason*; A. I. Memo 1293, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts, 1991.

[4] Chmielniak A., Dubrawski A., Siemiątkowska B.: A distributed system for control and management of teams of mobile robots, Proc. Automation'99, March 1999, Warsaw, to appear.

[5] Finin T., Labrou Y.: *KQML as an agent communication language*; in *Software Agents*, Bradshaw J. M. (ed.), MIT Press, Cambridge, Massachusetts, 1997, 291–316.

[6] Genesereth M. R., Ketchpel S.: *Software agents*; Communications of the ACM, 37(7), 1994, 48–53.

[7] Hunhns M. N., Singh M. P., Ksiezyk T.: *Global information management via local autonomous agents*; in *Readings in Agents*, Huhns M. N., Singh M. P., Morgan Kaufmann, California, 1998, 36–45.

[8] Jennings N. R., Wooldrige M. J.: *Software agents*; IEE Review, January 1996, 17–20.

[9] Ndumu D. T., Nwana H. S.: *Research and development challenges for agent-based systems*; IEE Proceedings – Software Engineering, 144(1), 1997, 2–10.

[10] Nwana H. S., Lee L. C., Jennings N. R.: *Coordination in software agent systems*, BT Technology Journal, 14(4), 1996, 79–88.

[11] Nwana H. S.: *Software Agents: An Overview*; Knowledge Engineering Review, 11(3), 1996, 205–244.

[12] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.: *Object-Oriented Modeling and Design*; Prentice–Hall, Engelwood Cliffs, New Jersey, 1991.

[13] Russel S., Norvig P.: *Artificial Intelligence: A Modern Approach*; Prentice–Hall, Engelwood Cliffs, New Jersey, 1995.

[14] Shoham Y.: *Agent-oriented programming*; Artificial Intelligence, 60 (1), 1993, 51–92.

[15] Thorpe Ch.: *Mixed traffic and automated highways*; Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'97), Grenoble, France, June 1997, 1011–1017.

[16] *Unified Modeling Language, Notation Guide Ver. 1.0*; on line at http://www.rational.com, January 1997.

[17] Wederhold G.: Mediators in the architecture of future information systems; IEEE Computer, 25(3), 1992, 38–49.

[18] Wooldrige M. J., Jennings N. R.: *Intelligent agents: theory and practice*; Knowledge Engineering Review, 10(2), 1995.