Plan Representation and Plan Execution in Multi-agent Systems for Robot control

Alexander Osherenko

University of Applied Sciences, Berliner Tor 3, 20099 Hamburg
osherenko@gmx.de

Abstract. Hardware agents as a part of cooperative multi-agent systems act in dynamically changing environments and accomplish tasks jointly. Since the pure hybrid plan representation provides no explicit means to represent particularly communication, task coordination and distribution in the multi-agent system, the proposed plan representation was adjusted to describe these aspects and combines the hybrid approach and an agent communication language. This paper focuses on the plan execution using this plan representation in a developed multi-agent system for robot control.

1 Introduction

Hardware agents as a part of cooperative multi-agent systems act in dynamically changing environments and accomplish tasks jointly. This paper focuses on the plan execution using a plan representation in a developed multi-agent system for robot control.

The two conventional approaches to the plan representation, the pure reactive and the pure deliberative approaches, have their disadvantages ([7]): the reactive approach ([6]) has no world model and thus lacks the learning capability, the deliberative approach ([8]) is slow and hence unable to solve tasks in a dynamically changing world. The hybrid approach to the planning and the plan execution combines the both conventional approaches making use of their advantages and eliminating disadvantages ([2], [4], [5], [17]).

Cooperative multi-agent systems facilitating tasks in dynamically changing environments execute hybrid plans. These plans describe reactive behaviors to commit short-term tasks, allow deliberative planning to execute long-term strategies and show cooperation between agents. Cooperation during plan execution is based on messages in an agent communication language. Our approach is to combine a hybrid plan representation and an agent communication language to execute plans in cooperative multi-agent systems.

2 Plan Representation

Recent work explores possibilities of a hybrid plan representation in multi-agent systems ([1], [3], [9], [10]). Our work is a contribution in this direction.

The proposed approach provides a plan representation for multi-agent systems accomplishing cooperative tasks in the real world (Fig. 1).

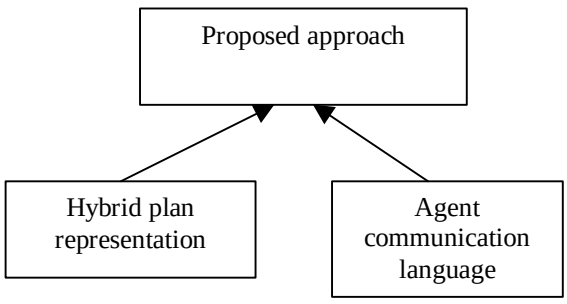


Fig. 1. Proposed plan representation.

The proposed plan representation combines:

- *The ACT Formalism*

The ACT formalism is an approach to the hybrid plan representation, a language for representing the knowledge required to support both the generation of complex plans and reactive execution of those plans in dynamic environments ([17]). Two features distinguish the ACT approach: (1) the development of a heuristically adequate system that will be useful in practical applications, and (2) the need to generate and execute complex plans with parallel actions of multiple agents. ACT has been used as the interlingua in several applications, including robot control and military operations, thus attesting its expressive and computational adequacy ([15], [16]). Since the proposed plan representation is based on the ACT formalism, it is necessary to provide a brief description of its features.

The purpose and applicability criteria for an act are formulated using a fixed set of environment conditions. Action specifications are called the plot, and consist of a partially ordered set of actions and subgoals ([13]).

The basic unit in the ACT formalism is an *act* which can be used both to encode plan fragments and standard operating procedures. An act describes a set of actions to achieve a specific purpose under certain conditions. The purpose is either satisfying a goal, or responding to an event thus introducing deliberative and reactive properties of the plan representation.

Actions are described by means of metapredicates. The most important of them in context of plan execution are the following:

- The ACHIEVE Metapredicate specifies a set of goals an act would like to achieve at that point in the partial plan.
- The WAIT-UNTIL metapredicate specifies that execution of an act is to be suspended until the indicated event occurs.
- The REQUIRE-UNTIL metapredicate specifies a protection interval, namely a condition to be maintained until another indicated condition for terminating this requirement occurs.
- The ACHIEVE-BY metapredicate specifies the goals to be achieved as well as a set of Acts, one of which must be used to achieve the goals.

Since the ACT Formalism provides no explicit means to represent particularly communication, task coordination and distribution in a multi-agent system, it was adjusted to meet these requirements. Thus, the ACT formalism was extended with the ICL language;

- *ICL* (an Interagent Communication Language)

ICL is the interface, communication, and task coordination language providing platform and language independence, heterogeneity in a multi-agent system ([11]). ICL serves in the proposed plan representation as a foundation for communication and cooperation.

ICL is based on events and has means describing how to:

- Perform queries;
- Execute actions;
- Exchange information;
- Set triggers;
- Manipulate data in the agent community.

Some event parameters are as follows:

- `block(T_F)` specifies whether the system waits for an event response or not;
- `address(Addr)` specifies to which address to send the event notification;
- `solution_limit(N)` specifies the number of event responses.

3 Scenario

The proposed plan representation was tested using a scenario for transferring an object (Fig. 2). The robots making the transfer aren't allowed to overstep the specified responsibility areas thus implying cooperation in the multi-agent system.

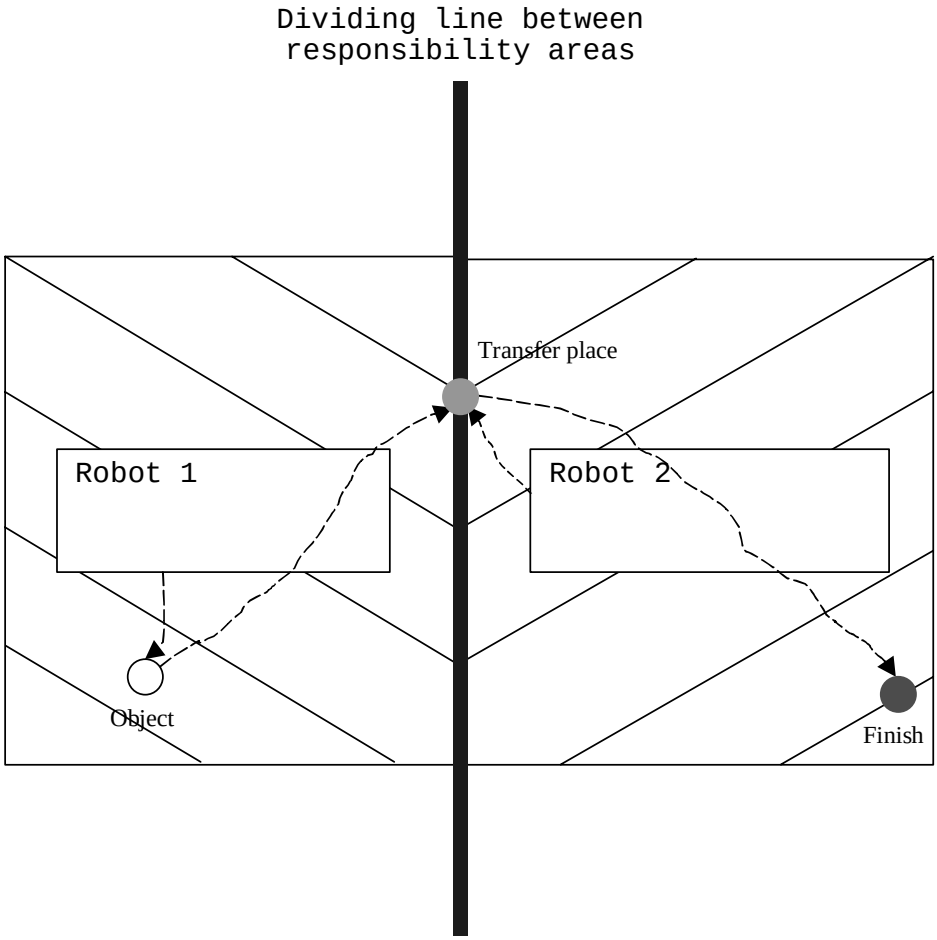
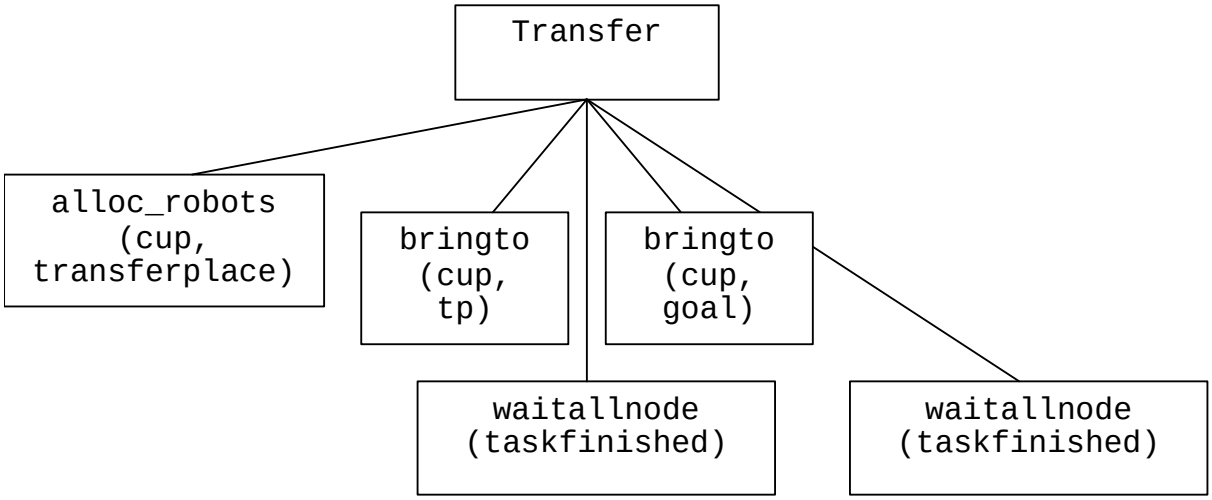


Fig. 2. Scenario for the object transfer.

The sequence for transferring an object is defined as:

1. Search and allocate two free robots that can make the transfer.
2. Instruct the first robot to bring the object to the transfer place.
3. Wait until the operation is finished.
4. Instruct the second robot to bring the object to the finish.
5. Wait until the operation is finished.

Since testing of the plan representation included only checking its adequacy for plan execution and not for planning, the plan was predefined on the design step. Fig 3 shows a plan structure for the scenario.



tp – transfer place

Fig. 3. Plan representation.

A text version of the proposed plan representation shows Fig. 4.

```
plot
((plotnode
     (alloc_robots(args(Object, Transferplace), options(block(true)))),

(plotnode
     (bringto
         (args(cup, transferplace))),
         options(block(true)),
             solution_limit(1)), address(data(free_robots1(Robot1)))))))))))),
(waitallnode
     (plotnode
         (taskfinished
               (args(dobringtoab),
               options(block(true),
               solution_limit(1),address(data(free_robots1(Robot1)))))))),
(plotnode
     (bringto(args(cup, finish)),
     options(block(true)),solution_limit(1),address(data(free_robots2(Robot2)))),
(waitallnode
     (plotnode
         (taskfinished
             (args(dobringtoac),
         options(block(true),solution_limit(1),address(data(free_robots2(Robot2)))))))))
```

Fig. 4. Plan representation for the transfer scenario.

This textual representation shows some analogues of ACT metapredicates (e.g. `plotnode` as the ACHIEVE metapredicate) and ICL predicates, e.g. `block(true)`.

System design and plan execution are described in the next section.

4 System Design

For means of testing the proposed plan representation for its adequacy a multi-agent system was developed. This system is based on the OAA architecture ([11]) and includes the following components:

- a plan unit with an executor providing plan execution and a server supplying the system with plans;
- a resource manager allocating resources during plan execution;
- robot agents making the transfer that serve as wrapper modules for real robots. In our case we used Pioneer 2 CE robots ([14]);

- some OAA service agents – a Facilitator that manages the whole system and a Debug agent providing a GUI for the multi-agent system.

Fig. 5 shows the components of the developed multi-agent system.

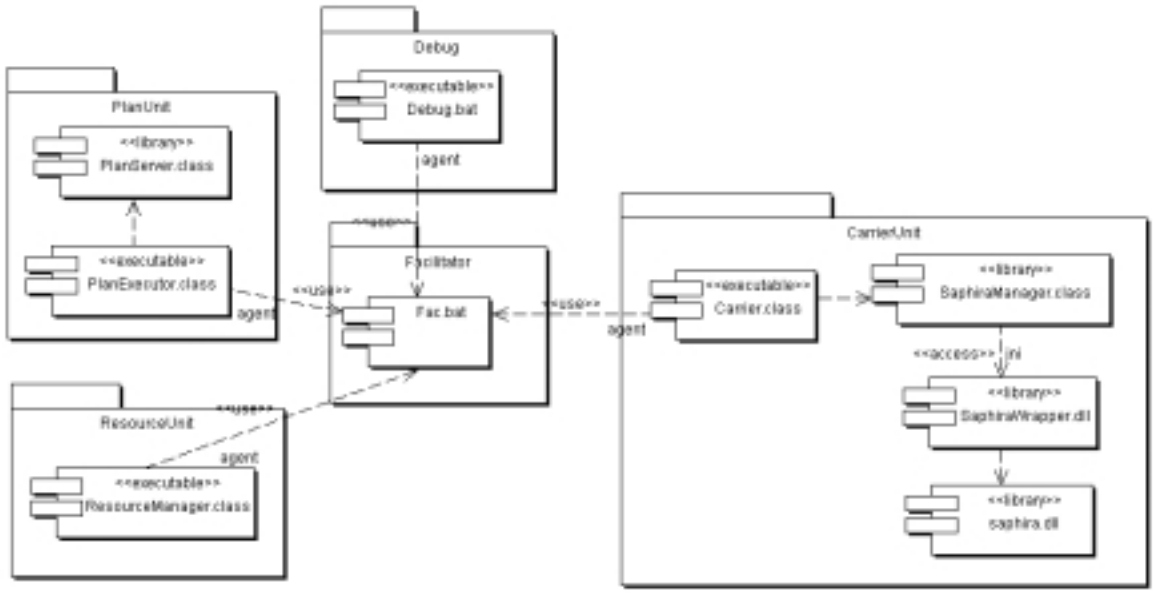


Fig. 5. System components.

During the plan execution the executor traverses through the plan representation and starts corresponding operations. Fig. 6 shows the sequence diagram for the transfer scenario (cf. Fig. 4).

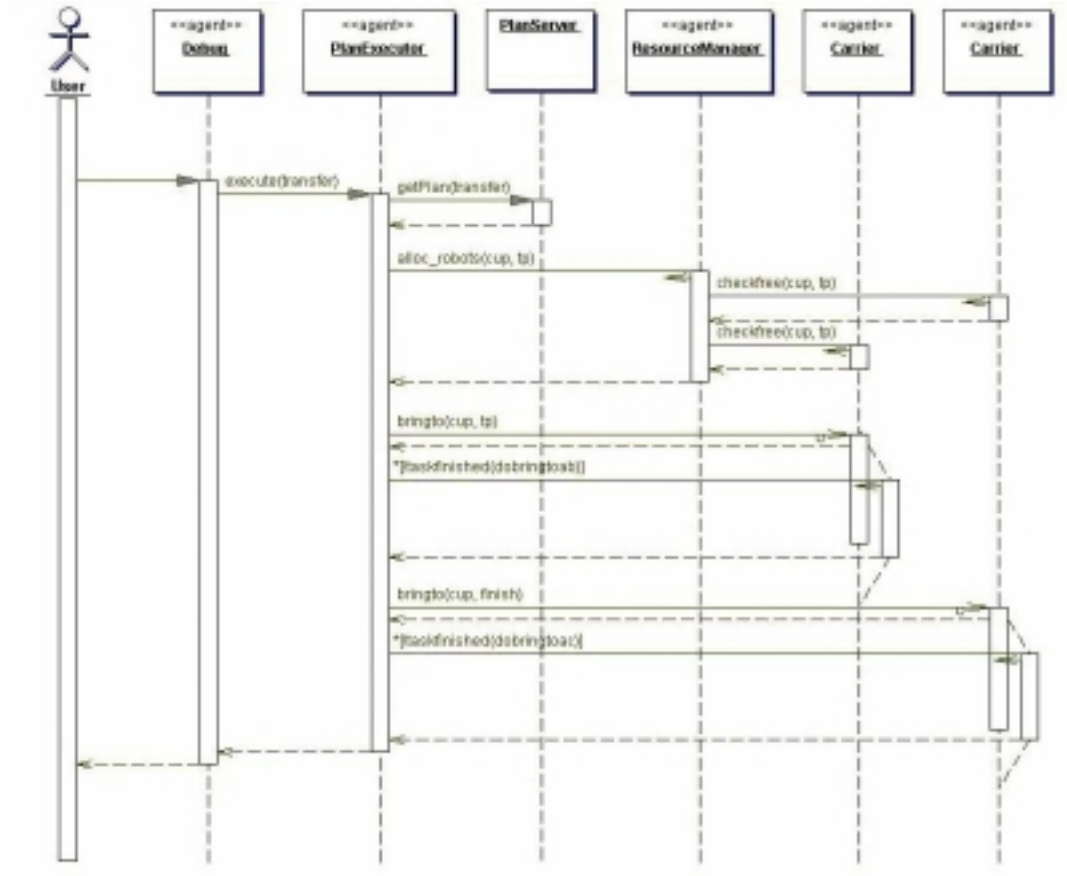


Fig. 6. Sequence diagram for the transfer scenario

The sequence diagram shows the following steps:

1. The user starts the transfer by issuing an `execute(transfer)` command at the Debug console. The Debug agent gets a plan for the transfer operation using the `getPlan` operation from the plan server. At the moment there is only one plan in the system. However the planning capability including flexible plan generation will be implemented in the next versions of the system.
2. The plan executor executes searching for two free robots that can make the transfer by starting the `alloc_robots` capability of the resource manager.
3. The next operation in the plan calls the `bringto` capability of the first robot to bring the object to the transfer place.
4. The `taskfinished` capability checks if the `bringto` operation is finished.
5. The next operation in the plan calls the `bringto` capability of the second robot to bring the object to the transfer place.
6. The `taskfinished` capability tells if the `bringto` operation is finished.

Each term type in the plan representation is interpreted individually because of its strongly different meaning from the other term types. Correspondingly, execution of the term `options(block(true))` results in text `block(true)`, whereby execution of the term `bringto(cup, tp)` runs the `bringto` agent capability.

The multi-agent system works under Windows 9x, Windows NT and Windows 2000 and was developed in Java 2.

5 Conclusion

This paper described plan execution using plan representation based on the combination of a hybrid plan representation and an agent communication language. Plan execution was shown using the transfer scenario. Our work demonstrates the possibility of plan execution on the base of those plans and their adequacy.

The future system enhancements include implementation of further predicates representing possible reactive and deliberative behaviors thus allowing the multi-agent system for planning and replanning. As a solution to the frame problem ([12]) in multi-agent systems and integrating existing approaches in this respect, we will make corresponding changes to the plan representation. We will develop an IDE to debug and to monitor the functioning of multi-agent systems based on the proposed plan representation.

6 References

1. Atkins E. M. Knowledge Representation for Real-time Plan Development. Department of Aerospace Engineering. University of Maryland. 2000.
2. Au S., Parameswaran N. Progressive Plan Execution In A Dynamic World. Department of Information Engineering, School of Computer Science and Engineering, The University of New South Wales. Australia.
3. Aylett R. Multi-Agent Planning: Modelling Execution Agents. IT Institute. University of Salford. 1995.
4. Bastié C., Régnier P. Planning and execution in a dynamic environment: the supervision of execution in SPEEDY. Department of Computer Science, University of Essex. URL: http://cswww.essex.ac.uk/conferences/ukpssig/essex-14/regnier-uue-2.ps.gz. 1995.
5. Blythe J., Reilly W. S. Integrating Reactive and Deliberative Planning for Agents. URL: http://citeseer.nj.nec.com/blythe93integrating.html. 1993.
6. Brooks R. A. "A Robust Layered Control System for a Mobile Robot". IEEE Journal of Robotics and Automation. URL: http://www.ai.mit.edu/people/brooks/papers/AIM-864.pdf. Vol. 2, No. 1, March 1986, pp. 14-23; also MIT AI Memo 864, September 1985.
7. Davidsson P. Autonomous Agents and the Concept of Concepts. Doctoral dissertation submitted in partial fulfillment of the requirements for the degree of PhD in Computer Science. URL: http://www.dna.lth.se/Research/AI/Papers/PhD.ps. 1996.
8. Fikes R. E., Hart P. E., Nilsson N. J. Learning and executing generalized robot. Artificial Intelligence, 3 : 251-288. 1972.
9. Hertzberg J., Jaeger H., Morignot P, Zimmer U. R. A Framework for Plan Execution in Behaviour-Based Robots. German National Research Institute for Information Technology (GMD), ILOG. 1998.
10. Horn G. S., Baxter J.W. Issues affecting the Control and Co-ordination of Agent Teams in the Simulated Battlefield. Defence Evaluation and Research Agency.
11. Martin D. L., Cheyer A. J., Moran D. B. The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence, vol. 13, pp. 91-128, January-March 1999.
12. McCarthy J., Hayes P. J. Some Philosophical Problems From The Standpoint Of Artificial Intelligence. Computer Science Department. Stanford University. URL: http://www-formal.stanford.edu/jmc/mcchay69.pdf. 1969.
13. Myers K. L., Wilkins D. E. The Act Formalism. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~act/act-spec.ps. 1997.
14. Pioneer Documentation. URL: http://robots.activmedia.com/docs. ActiveMedia. 2001.
15. PRS-CL: A Procedural Reasoning System. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~prs. 2001.

16. SIPE-2: System for Interactive Planning and Execution. SRI International Artificial Intelligence Center. URL: http://www.ai.sri.com/~sipe. 2001.
17. Wilkins D. E., Myers K. L. A Common Knowledge Representation for Plan Generation and Reactive Execution. SRI International. Artificial Intelligence Center. 1994.