PROCEEDINGS OF THE WORKSHOP ON APPLICATIONS OF SOFTWARE AGENTS
ISBN 978-86-7031-188-6, pp. 59 - 63, 2011

Integration of agents and planning systems

Saša Tošić[1], Miloš Radovanović[1], and Mirjana Ivanović [1]

[1] University of Novi Sad, Faculty of Science, Department of Mathematics and Informatics
Trg D. Obradovića 4, 21000 Novi Sad, Serbia
{sashat, radacha, mira}@dmi.uns.ac.rs

1 Introduction

Agents and planning systems represent two modern intelligent techniques. Although these two techniques can be very successfully combined, not much work has been done on their integration. One of the main reasons for this is the fact that these two kinds of systems evolved in an independent manner.

Planning systems started their evolution in the early 1960's and 70's when systems GPS (General Problem Solver) and STRIPS (STanford Research Institute Problem Solver) [Fikes, Nilsson 1971] were created. In 1989, ADL (Action Description Language) [Pednault 1989] was defined, and this language was the base for PDDL (Planning Domain Definition language) [McDermott et al. 1998], [Fox, Long 2003] the most commonly used planning language today. Another planning standard that is still in use is HTN (Hierarchical Task Network) [Nau et al. 2003], which was created in 1994.

The agent paradigm was created in the early 1990's and quickly became one of the most important fields of investigation in artificial intelligence. Very often, it is combined with other techniques to successfully solve a given problem. This paper will present how this paradigm can be efficiently combined with planning systems to solve complex tasks.

2 Related Work

Agents and planning systems are combined mostly in two ways. Some simple planners are used to create deliberative agent architecture, while complex ones are used only as external planners with a low level of integration/communication with agents.

Planners used to create deliberative agent architecture are usually simple planners that use a symbolic representation to describe the environment. Since symbolic representations generally lack adequate expressive power, abilities of agents with this architecture are limited. Another problem is that such planners are mostly domain-dependent and cannot be used in a large number of domains. Also, they are usually designed for a specific architecture by people that are not experts in planning systems and can be slow and not scalable to large data sets and complex environments.

Many of the mentioned disadvantages can be avoided by use of modern domain-independent planning systems. Most of them can be used in large number of domains, can be very fast and have scalable abilities. The largest disadvantage of such planers is that most of them can be used only to create plans and not to execute them. In adition, communication between agents and planners is very poor and is mainly realized by files exchange. In order to overcome all previously mentioned disadvantages, we created Agent Planning Package (APP) [Tošić, et al. 2008], a domain independent planning system whose main purpose is to extend agents with planning abilities.

3 Agent Planning Package

APP is a planning system whose main purpose is to extend agent abilities with planning. It is implemented as a Java package, and can be used in any agent system written in this language.

Some of the tasks that we wanted to fulfill are:

- to create a domain independent planning system that can be used in large number of domains,
- to create a system that can be easy integrated into different agent systems,
- to create a system with different planning abilities and different types of plans that can be used depending on agents' needs,
- to enable plan execution and adjustment of this process to a current planning domain.

First of all, APP is implemented as a domain-independent planning system and as such can be used in a variety of domains. It supports PDDL as the language used to describe the planning domain and problem and requires a description of the planning domain to be defined in a separate file before it can be used. The domain definition contains the definitions of all types of objects in the system, the predicates that can be used to define relations between objects, and the basic actions that can be executed by the system. In contrast to this, the planning problem can be defined during system execution, because it usually contains data about the environment which can be changed. One of the advantages of APP compared to other planning systems is that different problems for the same domain can be defined so an agent can create plans for several similar situations. Also, an agent can create more than one instance of APP's planner that enables the use of more than one planning domain, each defined in one instance. This allows the agent to create plans in different domains, for example to create a path for a mobile agent and to create a plan how to solve a given task.

Since APP is implemented as a Java package, it contains over 100 classes in its API that can be used in order to create, edit or modify the planning domain and problem. Using the classes implemented in APP's data manipulation module, an agent programmer can change the data stored in this module and make it consistent with the agent's environment. There are two advantages of using APP's data manipulation module: there is no need to duplicate data inside an agent and the planning system because all the data can be stored in this module, and the data stored in this module can be used directly during the plan creation phase.

The plan creation phase can be started whenever an agent has need to create a plan, and it can be initiated by calling one of the several methods implemented in the plan creation module. Method createPlan() is the basic method for plan creation, and it returns the first plan that can be found. In contrast to this, method createAllPlans() can return all the plans that can be found within a given time frame. More than one plan can be created also by calling createMultiPlan() method, but this method returns only the plans of the same serial length as the plan created by the createPlan() method. The reason an agent may have the need to create more than one plan is that the first created plan may not always be the optimal one, and the time needed to create more than one plan is often negligible compared to the time needed to create the first plan. This way, the agent has the ability to choose a plan from the given set of plans which can be more appropriate to the current situation.

Execution of a plan can be initiated by calling the execute() method from the plan execution module. The plan is decomposed depending on its structure into simpler subplans and every subplan is executed separately. In the case of a serial plan, the subplans are executed one after another, while in the case of a parallel plans they are executed in parallel. If a plan is created using the createAlterPlan() method, the first subplan in an alternative plan is executed, and if execution fails, the next subpan is executed. These alternative plans can be very useful in dynamic environments where the environment can be changed during plan execution and can decrease the possibility of execution failure. This way, the system can run faster because there is no need to spend time to create a new plan.

The plan can be decomposed into simpler subplans until single-action plans are reached. The execution of a single-action plan can be adjusted to the domain by redefining the execute() method of the class actionCode connected to the appropriate action in the planning domain. Inside the execute() method, the agent programmer can use the GUI to communicate with users, can access the database or do any other actions needed to execute that kind of action. It can use the action parameters obtained during the plan creation phase or access some agent’s data and change them. In case the execute() method finishes successfully, the data manipulation module is updated automatically according to the action effects defined in the planning domain. If not, the plan execution stops, except in the case of the alternative plan where the execution of the next subplan is automatically initiated.

3.1 Resource Management System

In order to illustrate the integration of agents and APP, we created the Resource Management System (RMS), a simple system for distributed resource management. In the current version of the software only two kinds of resources are supported: printers and main servers, which can be distributed over the computer network.

The main purpose of the system is to collect user requests for file printing or sending by mail. Once the system gathers enough requests, it creates a plan for their execution. The system has the autonomy to choose printers and mail servers to be used in order to optimize its performance. This way, the system can avoid bottlenecks in overloading one printer and can reduce network load in the case that the same file needs to be both printed and sent by mail.

The system is implemented as a multi-agent system employing four types of agents, using the JADE framework [Bellifemine et al. 2001] for their creation. The central part of the system is implemented as an intelligent RMSAgent whose job is to collect requests for file printing and mail sending, create the plan for their realization and initialize its execution. To make this easier, we created one instance of the APP planner inside the RMSAgent and loaded the planning domain written in PDDL which was previously constructed.

In RMS, there is one host agent residing on every computer in the network. Its job is to collect information about resources related to that computer and send that information via ACL messages to the RMSAgent, who stores that information inside its instance of APP's planner.

The requests are collected by user agents and sent to the RMSAgent via ACL messages. User agents can be hosted on any computer in the network and are used as a GUI for communication with users. Collected requests are gathered and used to create the goal inside the planning problem.

After an adequate number of requests is collected, the RMSAgent starts its planning behavior that contains several activities. First, it calls the createPlan() method from the plan creation module to create the plan for the created goal. After the plan is created, it calls the execute() method to initiate the plan execution process. The code for this has the following form:

```
plan p =
  RMSPlanner.getInstance(currentInstance).createPlan();
if (p!=null) {
  p.execute();
}
```

The main role in plan execution is assigned to working agents, whose job is to execute single actions defined in the plan. These actions can be one of the following: load file, move to another computer, reserve printer on host computer, print file using reserved printer, and send file using mail server installed on host computer. After plan execution is initiated, RMSAgent allocates these single-action tasks to appropriate working agents defined in the plan. After a working agent successfully finish his tasks, he informs the RMSAgent, who in turn can update the planner's data manipulation module and remove the requests from the planning goal.

4 Conclusion

As it is shown it this paper, integration between agents and planning systems can be improved and both technologies can gain benefits from that integration. Agents can use a planning system to create their intelligent behavior and more easily solve given tasks. On the other hand, agents offer a suitable environment for plan execution, especially in distributed environments.

By implementing APP as a Java package, agents can use APP's data manipulation module to store information about their environment and without the need to duplicate that information both in the agents and the planning system. The plan creation process can directly use that information with no need to transform it to a more suitable form in order to create a plan or plans, depending on the agents' needs.

Different methods can be used to create plan(s), while all of them can be limited to a maximum time spent for this process.

One of the most important benefits our system is that it allows plan execution and adjustment of that process to a planning domain by overwriting the execute() method of class actionCode for every action in the domain. This way, a plan can be executed directly, removing the need to create a plan parser and execute the plan manually, in the way it is done when using an external planning system.

References

[Bellifemine et al. 2001] Bellifemine, F., Poggi, A., Rimmasa, G., 2001, "Developing Milti-agent Systems with JADE", Intelligent agents VII, LNAI 1986. pp. 89-103, 2001, Springer-Verlag, Berlin Heidelberg, 2001.

[Fikes, Nilsson 1971] Fikes, R., Nilsson, N., "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence, 2, pp. 189-208, 1971.

[Fox, Long 2003] Fox, M., Long, D., 2003, "An extension to PDDL for expressing temporal planning domains", Journal of Artificial Intelligence Research, Special issue on the 3rd International Planning Competition, 2003

[McDermott et al. 1998] McDermott, D., & the AIPS'98 Planning Competition Committee (1998). "PDDL–the planning domain definition language". Tech. rep., Available at: www.cs.yale.edu/homes/dvm.

[Nau et al. 2003] Nau, D., Au, T., Ilghami, O., Murdock, J. W., Wu, D., Yaman, F., 2003, "SHOP2: A HTN Planning System", Journal of Artificial Intelligence Research 20 (2003), pp. 370-404.

[Pednault 1989] Pednault, E., P., D., "ADL: Exploring the middle ground between STRIPS and the situation calculus", In Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, 1989

[Tošić, et al. 2008] Tošić, S., Radovanović, M., Ivanović, M., "APP: Agent Planning Package", Scalable Computing: Practice and Experience, Volume 9, no1, pp 39-49