A Layered Architecture for Implementing Autonomous Planning Agents

Giuliano Armano, [1] Vanna Galaffu, [1] Carlo Muntoni, [2] and Eloisa Vargiu [2]

(1) DIEE, Dipartimento di Ingegneria Elettrica ed Elettronica, Università di Cagliari, Piazza d'Armi, I-09123, Cagliari, Italy, email: armano@diee.unica.it; galaffu@diee.unica.it

(2) CRS4, Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna, VI Strada OVEST, Z.I. Macchiareddu I-09010 Uta (CA), Italy, email: muntoni@crs4.it; eloisa@crs4.it

Abstract

This paper briefly describes an architecture for implementing autonomous agents that embody sophisticated planning capabilities. In particular, we are currently working on a two-pass vertically layered architecture, designed to deal with a complex environment. Such an architecture is currently based on three levels of abstraction (i.e., situated, strategic and deliberative), but has been designed for being easily generalized to a N-levels architecture, depending on the given environment and task complexity. Each level controls the underlying one, so that an agent behavior is supported by a clean hierarchical organization. Our autonomous agents act in a virtual world created for a computer game, and must interact with it by suitably planning and executing complex actions.

1. Introduction

In the last few years, AI researchers have concentrated their efforts in the field of intelligent autonomous agents, i.e., on systems capable of autonomous sensing, reasoning and acting in a complex environment. As in this paper we are mainly concerned about architectures, let us briefly recall that an agent architecture is essentially a map of the internals of an agent (i.e., its data structures, the operations that may be performed on them, and the corresponding control flows [1]). In particular, as far as two-pass vertically layered architectures are concerned, decision making is realized via various software layers; each layer been devoted to deal with the environment at different levels of abstractions. In such architectures information flows up until a layer able to deal with the received stimulus is reached, and then the control flows back down to the actuators (e.g., INTERRAP [2]).

In this paper, we briefly illustrate a layered architecture that exhibits planning capabilities in an environment whose complexity is comparable to the one that characterizes the real world. After a short introduction on planning issues, we describe in more detail the proposed architecture, together with some experimental results.

2. Planning Agents

It is well known that the AI planning community has greatly promoted innovations in agents' design, and that planning capabilities are one of the most important features to be implemented in an autonomous agent. In particular, the introduction of intelligent autonomous agents has promoted investigations on planning algorithms with the aim of using them in real-world domains. To this end, it is worth pointing out that a relaxation of all the simplifying assumptions made by classical planners about their working environment has to be made. In particular, atomic actions, deterministic effects, accessibility, and static environment are simplifying hypotheses that no longer hold for real-world domains (see, for example, [3] and [4] for a discussion on this topic). In fact, a real-world domain is usually (i) not-completely controllable, i.e., an agent does not have complete control over its actions, (ii) not-completely accessible, i.e., it is impossible for an agent to have complete knowledge about the underlying environment, and (iii) dynamic, i.e., the presence of other agents, or exogenous events can modify the underlying environment. In the following, for the sake of simplicity, we will characterize as "complex" an environment that exhibits such characteristics.

To deal with a complex environment, several extensions of classical planning algorithms have been proposed. Such extensions can be classified according to several features, thus giving rise to different classification hierarchies. In our opinion, complex domains need to point out how a planning agent adapts a plan to the changes occurred in its operating environment while trying to attain a goal. To this end, one may try to deal with any change from an "open-loop" perspective, or else to accept the possibility of adopting a "closed-loop" approach based on re-planning.

Contingency planners represent a typical example of the former approach, whose underlying strategy consists of handling off-line any source of uncertainty (see, for example, WARPLAN-C [5], CNLP [6]). As far as the latter approach is concerned, let us recall that the integration of planning and execution can lead to powerful control strategies (see, for example, [7]).

Of course, a plan is updated only whenever a change in the environment makes it obsolete. The underlying hypothesis is that such a kind of planners should be highly reactive against unexpected environmental changes.

The planning capabilities of the proposed architecture allow an agent to operate in a complex environment while exploiting a "closed-loop" approach based on local re-planning. The search complexity is controlled by allowing an agent to make plans at different levels of abstraction. In this way, only part of the original plan must be modified according to a change on the environment.

3. A Three-Layers Architecture

We are currently implementing a two-pass vertically layered architecture equipped with three layers, i.e., situated, strategic and deliberative (see figure 1). Each layer is numbered according to its level of abstraction (1, 2, and 3, respectively). Thus, level 3 is more abstract than level 2, and so

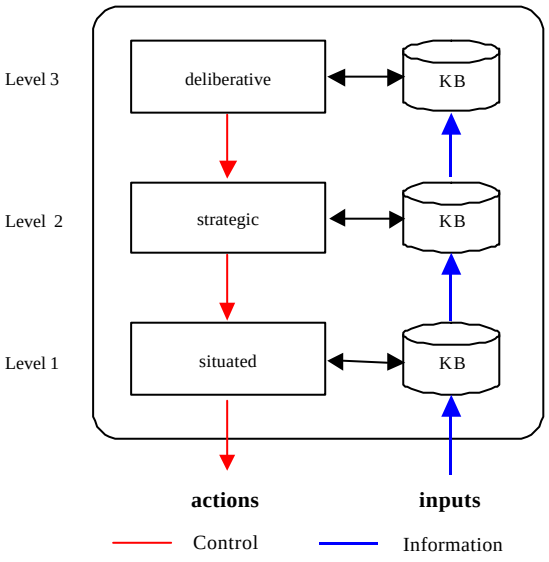

*Fig. **1** – A three-layers, two-pass, vertical architecture.*

on. It is worth pointing out in advance that the proposed architecture is completely scalable; in fact, it might be easily equipped with N layers, depending on the environment complexity. Furthermore, whereas in classical layered architectures each level is usually devoted to cope with different behavioral characteristics, in the proposed architecture each layer is –at least conceptually– identical to any other one, each of them being able to embody reactive, deliberative and proactive functionality. Only the "responsibilities" of a layer change radically, depending on the level of abstraction being considered. According to the features that characterize a two-pass vertically layered architecture, the information flows from level 1 up to level 3, whereas the control flows from level 3 down to level 1. Thus, an agent that receives an input from its environment lets the information flow until a suitable layer able to deal with it is found. Then, the reactive module is devoted to decide whether or not the current activity has to be continued or aborted. In the latter case, the deliberative module is devoted to set a new goal, depending on the nature of the input.

Now, let us focus our attention on the hierarchical planning process, with the aim of explaining how agents are able to solve difficult tasks in a complex environment. As expected, plan generation is performed on different layers, each one equipped with a suitable planner. Each planner is allowed to use only the operators available at the corresponding level. It is worth pointing out that, whereas typical hierarchical planners use a fixed (a priori) decomposition strategy (e.g., HTN [8], ABSTRIPS [9]), in the proposed architecture, the decomposition is performed at runtime and distributed on several layers. Each layer is devoted to perform a (local) planning on any goal imposed by its overlying planner (if any), or to perform a re-planning activity, if needed. Thus, a goal to be attained at level K enforces other goals (i.e., plans) on the underlying K-1 levels. That is why, an abstract operator defined for a given layer may hide a complex plan on the underlying layer. As agents generate hierarchical plans, a complex plan can be easily adapted in response to any unpredictable event occurred while executing it; in this way any re-planning activity requires only a local search at the proper level of abstraction.

As far as predicates and operators are concerned, it is well known that hierarchical planning can be done by abstracting over predicates or over operators, both mechanisms being successfully used to reduce the computational complexity of the search ([10]). We adopted a three-levels hierarchy of predicates, according to the layers that make up the current release of the proposed architecture. Predicates can be ground or abstract, depending on their allocation. Only predicates defined at level 1 are ground, whereas every abstract predicate is defined on top of other predicates (the visibility of level K being strictly reduced to level K-1). Moreover, each layer has its own set of operators that can be managed by the corresponding planning module. Operators can be ground or abstract, and are defined according to a STRIPS-like syntax (we prefer to use the term "ground", instead of "atomic", to stress the fact that considering operators no further decomposable is only a convention). Note that, our abstraction

hierarchy is not generated by dropping literals from the original problem definition. In this sense, our abstract operators are more similar to HTN compound tasks than to abstraction hierarchies described in ABSTRIPS. On the other hand, operator expansion is similar to the one performed in ABSTRIPS.

It is worth pointing out that there is no need of generating a completely detailed plan before starting execution; in fact, execution starts when the first situated plan has been generated (as an expansion of a the abstract operator that starts the plan at the strategic level). In this way, a plan is progressively generated and executed in an interleaved way. The general form of a layered plan is shown in figure 2 (for the sake of simplicity, the deliberative layer has been disregarded). This strategy allows an agent to elaborate a plan in small "chunks", whose detailed representation can be deferred until actually needed. In this way, a complex task can be started

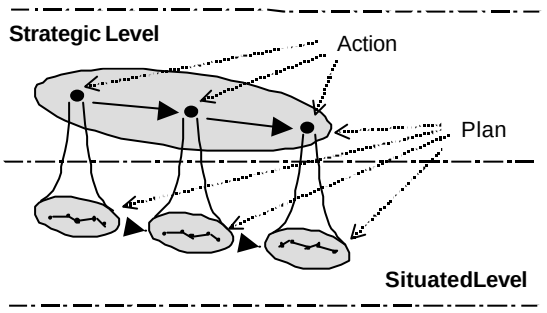

*Fig. **2** – A typical hierarchical plan.*

even though only a minimal part of the entire plan has been generated, thanks to the adopted "closed-loop" planning strategy. Note that, if a plan at the situated layer fails, local re-planning is attempted; if no solution is found, the failure is notified to the upper (strategic) level, where local re-planning is attempted at a higher level of abstraction, and so on.

The adopted strategy (which falls into the category of Interleaved Planning and Execution), exploits the architecture by moving any event that cannot be handled by the current layer up to the next (more abstract) one. In this way, local re-planning can be used as a general mechanism to deal with unexpected events, without being always compelled to start a global re-planning activity. Moreover, in such a process an agent does not need to have a complete knowledge of all the information required to successfully complete the whole plan: thanks to the interleaved strategy that has been implemented, during plan execution it can gather the information required to expand in the proper way abstract operators.

4. Experimental Results

The proposed architecture has been developed for an ongoing project aimed at developing a computer game. The game lets the user play within a virtual city populated by two different kinds of entities, i.e., physical and network "avatars". The former ones represent inhabitants of the virtual world, whereas the latter ones (e.g., computer viruses) live within a virtual computer network, very similar to Internet. A prototype of the system has already been implemented, using CLOS as a target language for the sake of rapid prototyping. The current release is aimed at giving avatars planning capabilities able to cope with the complex environment that characterizes the game. In particular, physical avatars must be able to move within the virtual world and to interact with its objects. Moreover, they must be able to exploit the capabilities of the underlying virtual computer network, to send and receive information. As already pointed out, agents are able to deal with a given problem at different levels of abstraction. For each layer, actions schemata are suitably represented according to a STRIPS-like syntax. Planning results from the interaction among the planners located at different layers of the architecture, and the behavior of each planner follows a UCPOP-like strategy ([11]).

The main difference among the domains defined on different layers is the abstraction adopted to represent an agent behavior and knowledge. For example, the *(goto ?l1 ?l2)* action –strategic level– requires, as a precondition, that the agent must be near to the starting location *(is-near ?l1)*. The post-condition (effect) is that the agent must be near to its destination *(is-near ?l2)*. The *is-near* predicate becomes more detailed at the situated level, where it is expanded as *(:or (next-to ?l1) (inside ?l1))*, the *or* being resolved at runtime.

Let us consider, now, the following sample problem: "an agent is located inside a building connected to the network. Its goal is to retrieve a file located in another building (not connected to the network), and to send it to the mailbox of its corresponding player". To solve the problem at the strategic level the agent must (i) go to the physical location where the file is located, (ii) get the file, (iii) return to the initial building and (iv) send the file by email. Of course, the goal that must be attained at the strategic level is set by the deliberative level. After that, a backward search is started that carries out the creation of a plan at the strategic level. The actions of the strategic level are abstract operators that need to be further expanded at the situated level. For example, the strategic operator *goto* originates a planning problem at the situated level, which is solved by

using the situated operators *open-door, go-outside*, and *move*. The latter operators are to be considered ground and are implemented by suitable actions that act on (and modify) the underlying virtual environment.

5. Conclusions and Future Work

In this paper, a two-pass layered architecture for implementing autonomous planning agents has been briefly outlined. Agents built upon such an architecture are able to deal with a complex environment. Currently, three levels of abstraction (i.e., situated, strategic and deliberative), have been implemented, although a N-levels architecture is also feasible, depending on the given environment and task complexity. Each level controls the underlying one, so that an agent behavior is supported by a clean hierarchical organization. Agents implement "avatars" that act in a virtual world created for a computer game. The main mechanism exploited to interact with such a world is an interleaved iteration of planning and execution. A virtual computer network is also available, to give physical avatars the capability of performing typical actions allowed on Internet (telnet, ftp, etc.). Network-based entities (e.g., computer viruses and hacking tools) are implemented using agents, too.

As far as future work is concerned, we are currently studying the problem of giving agents the capability of learning abstract operators by examining their own plans. Furthermore, complexity issues, aimed at characterizing re-planning activities according to the hypotheses made about the underlying dynamic environment, are currently under study.

Acknowledgements

This work has been carried out under the joint project "Side-EffectZ", which involves CRS4[1] and Mediola.[2] We wish to thank M. Agelli (CRS4) and G. Dettori (Mediola) for their useful suggestions and valuable help.

References

[1] Wooldridge, M., "Intelligent Agents", in G. Weiss (ed) *Multiagent Systems*, MIT Press, April 1999.

[2] Muller, J., "A cooperation model for autonomous agents", in J.P Muller, M. Wooldridge and N.R. Jenning (eds) *Intelligent Agents III, LNAI Vol. 1193*, pages 245-260. Springer-Verlag, Berlin, Germany, 1997.

[3] Weld, D., "An Introduction to Least Commitment Planning", *AI Magazine*, pp. 27-61, 1994.

[4] Nareyek, A., "A Planning Model for Agents in Dynamic and Uncertain Real-Time Environments", in *Proceedings of the 1998 AIPS Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, pp. 7-14, AAAI Press, Menlo Park, California, 1998.

[5] Warren, D.H.D., "Generating Conditional Plans and Programs", in *Proc. of the Summer Conference on AI and Simulation on Behavior*, pp. 344-354, 1976.

[6] Peot, M., and Smith, D., "Conditional Nonlinear Planning", in Proc. of the 1st International Conference on AI Planning Systems, pp. 189-197, 1992.

[7] Nourbakhsh, I., *Interleaving Planning and Execution for Autonomous Robots*, Kluwer, Academic Publishers, Boston, 1997.

[8] Erol, K., Hendler, J., and Nau, D., "UCMP: A Sound and Complete Procedure for Hierarchical Task-Network Planning", in *Proc. of the Second International Conference on AI Planning Systems (AIPS-94)*, pp. 249-254, June, 1994.

[9] Knoblock, C.A., "Learning Abstraction Hierarchies for Problem Solving", in *Proceedings of the Height National Conference on AI*, pp. 993-928, AAAI Press, Menlo Park, CA, 1991.

[11] Penberthy, J.S., and Weld, D., "UCPOP: A sound, complete, partial order planner for ADL", in *Proc. of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 103-114, Oct 1992.

[10] Knoblock, C. A., "An analysis of ABSTRIPS", in *Proc. of the 1st International. Conference AI Planning Systems*, June 1992.

[1] Centro Ricerche Sviluppo e Studi Superiori in Sardegna.

[2] Mediola is a company that works on multimedia, games, and Internet-based applications.