Design and Implementation of Cognitive Soccer Robots

C. Castelpietra, A. Guidotti, L. Iocchi, D. Nardi, and R. Rosati

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198, Roma, Italy
{castelp,guidotti,iocchi,nardi,rosati}@dis.uniroma1.it

1 Introduction

One of the major challenges in the design of robots that can act autonomously in unstructured, dynamic and unpredictable environments is the ability to achieve desired goals by executing complex high-level plans, while promptly reacting to unpredicted situations and adjusting the behavior based on new knowledge acquired through the sensors during task execution. Several years of research have focussed on the software architecture by exploiting the sense-plan-act (or deliberative) approach [9], the behavior-based (or reactive) one [1], as well as hybrid architectures [3] which combine advantages of both reactivity and deliberation. However, for an effective application of hybrid approaches a crucial role is played by the organization of information among the layers, which is heavily influenced both by the features of the robotic platform and by the problem at hand.

In this paper we present a hybrid approach that has been used in the implementation of both the players of our University that contributed to the ART team [8] (now competing in the SPQR-Wheeled team) and the players of the SPQR-Legged team [7]. The main features of the approach are: a hybrid architecture that is heterogeneous and asynchronous allowing for an effective integration of reasoning and reactive capabilities; a high level representation of the plans executed by the player; a formal account of the system for generating and verifying the plans. Specifically, we present a novel approach towards the realization of a system that is able to execute the plans that are generated from a formal specification. In this respect the approach is in line with the work on Cognitive Robotics [10,6,2], which aims at the realization of a system based on a specification given in a formal language, such as for example the Situation Calculus. The use of a high-level specification has provided significant advantages also from a practical viewpoint. A significant outcome of the approach is that we were able to implement the players of the SPQR-Legged team (that had a good performance in the Sony Legged League of RoboCup 2000) in a very short time.

A. Birk, S. Coradeschi, and S. Tadokoro (Eds.): RoboCup 2001, LNAI 2377, pp. 312–318, 2002.

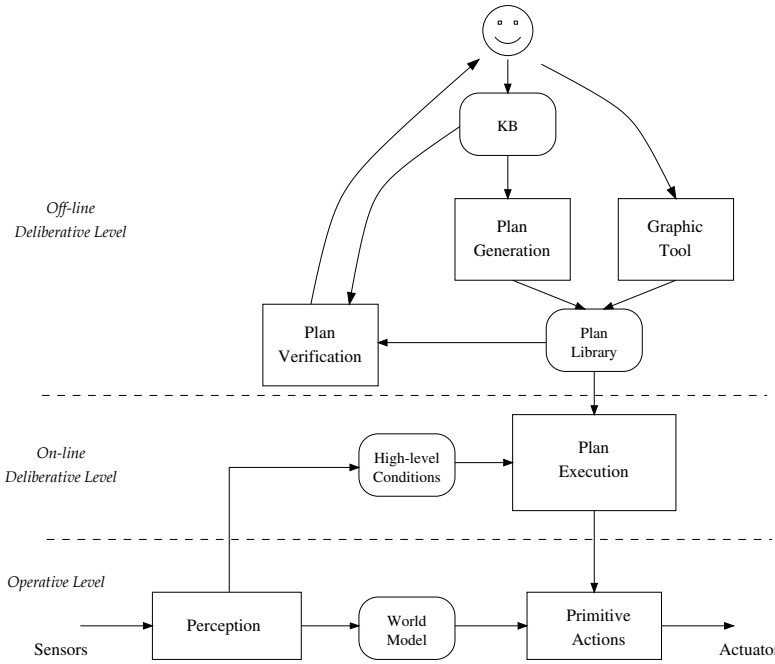

Fig. 1. Layered architecture for our robots

2 Software Architecture

In this section we describe a layered hybrid robot architecture (see also [5]) that has been implemented on different kinds of robotic platforms, namely Sony AIBOs, Pioneer, and home-made wheeled robots. The software architecture for all our robots follows a common schema (Fig. (1)) presenting two layers:

1. An *Operative Level* based on a numeric representation of the information acquired by the robot sensors and of the data concerning the current task;
2. A *Deliberative Level* based on a symbolic representation of the information about the state/situation and of the data concerning the task to be accomplished: the *On-Line Deliberative SubLevel* is in charge of evaluating data during the execution of the task, while the *Off-line Deliberative SubLevel* is executed off-line before the actual task execution.

This layered architecture is based on a different representation of the information. In fact knowing the exact position of the objects on the field is often not necessary in the high-level and therefore a symbolic representation is more appropriate, while the low-level numerical representation is based upon the exact coordinates of the various objects. For instance the high level representation of the position of the ball with respect to the robot could be given by a predicate such as *BallOnTheLeft*, while at the operative level we have the (x, y) position of

the ball with respect to the robot. The robot control is also different in the two layers: the deliberative level makes use of action plans representing high-level decisions about the execution of primitive actions; in the operative level instead we have the implementation of primitive actions, that are specific to each platform.

The main features of our architecture for integrating reasoning and reactivity are *heterogeneity* and *asynchronism*. Heterogeneity is important in order to make use of different techniques, for example a fuzzy reactive controller, a logic-based knowledge representation system, and low-level image processing routines. The use of a single representation system, as proposed in [4], for both the reasoning and the reactive system introduces a limitation in the technologies that can be used. With a heterogeneous architecture instead, we provide the designer with all the flexibility required by the complexity of the development process. Asynchronism is also very relevant for an effective integration of reasoning and reactivity. Indeed, asynchronous architectures, unlike synchronous ones [9,10], allow time-critical modules to have processing time available when needed. Moreover, the division of these layers has been designed in such a way that the deliberative level is the same for all our robotic platforms, while the operative level depends on the robotic platform. In fact, all our robots share the same deliberative level, while the implementation of primitive actions and numeric world representations are different for our different platforms (AIBO, Pioneer, home-made base).

The *operative level* is heavily dependent on the robot platform: Saphira controller is used for the Pioneer wheeled robots, and Sony OPEN-R for the AIBO robots. The perceptual module processes sensor data in order to reconstruct a geometrical view of the environment around the robot (e.g. ball, robots, and marker position). The main sensor for all our robots is a color camera, therefore image processing includes color segmentation, feature extraction and world reconstruction, that are very similar for all our robotic platforms. The control module in the operative level implements a set of primitive actions, that are considered atomic at the higher level, but that correspond to a sequence of control commands to send to the robot's actuators. This module is again dependent on the different robots.

The *deliberative level* is mainly concerned with an explicit symbolic representation of the robot's knowledge about the environment. This knowledge is formed by both a general description of the environment provided by the robot's designer and the information acquired during task execution. The world model in this level contains a set of symbolic information corresponding to the data in the geometric world model of the operative level. The deliberative level is formed by two main components: 1) a plan execution module that is executed on-line during the accomplishment of the robot's task and is responsible for coordinating the primitive actions of a single robot; 2) a reasoning module, that is executed off-line before the beginning of the robot's mission, and generates a set of plans to be used to deal with some specific situations. The reasoning system processes a knowledge base containing information on the world for making decisions about actions to be performed for goals achievement. In addition to an automatic plan generation module, the user can also manually design (or

modify) a plan by using a graphic tool or validate a plan with respect to the knowledge base by using a plan verification module based on model checking.

3 Representation and Execution of Plans

A *plan* is represented as a *transition graph*, where each node denotes a state, and is labeled with a set of properties, and each arc denotes a state transition and is labeled with the action that causes the transition.

Actions are represented using preconditions and effects. Preconditions are the conditions that are necessary for activating the action and indicate what must be true before and during the action execution. Effects are the conditions that must hold after the execution of the action and characterize how the state changes after the execution of the action: they specify direct effects of an action if executed under given circumstances. A description indicating the overall behavior of the action is associated with each action, and is only concerned with the execution of the action in that particular context. The actions in the graph can be classified into ordinary (i.e. movement) actions and sensing actions. The former cause changes in the environment, while the latter are used for acquisition of information, in order to let the robot take better decisions. In Section 4 we present a more general characterization of actions used for plan generation.

A *Plan Execution Monitor* is in charge of the correct execution of the actions composing the plans. In the monitor's implementation, a plan is stored as a graph data structure. The monitor's task is that of visiting the graph, calling and suspending the actions as necessary. In the formal framework that we have adopted, a number of assumptions are made both to limit the complexity of the language and to enable for automatic reasoning. Consequently, for an effective execution of the plans, additional information concerning the verification of preconditions and effects is needed. In particular, both the duration of actions and the failures of action execution must be taken into account. For example, some preconditions must be constantly verified during the entire execution of the action, while others need to be checked only for the action's activation. In the first case, the action is carried out as long as the condition is true. If the condition becomes false during the execution of the action, the action fails and a recovery action is needed. For instance, in a *PushBall* action (the robot pushes the ball towards the goal) the condition *NearBall* (the robot has the ball next to itself) has to be true during the entire execution of the action. In other cases, the condition is checked only once at the activation and, once the action has been activated, it is carried out, independently of the condition's value during the action. For example, *NearBall* can be considered as an activation condition of a *Kicking* action, that is thus checked only before the action starts.

Following the considerations above, the plan must be marked with additional information, that is external to the formalism, but necessary to the monitor for interpreting and executing the plan. Specifically, the monitor has to know which preconditions have to be verified during the entire execution of the action and which effects determine the state transition.

4 Plan Generation and Verification

We now briefly present the formalism used for the high-level specification of dynamic scenarios (we refer the interested reader to [2] for further details). We have adopted a logic-based formalism, and have exploited both its formal semantics and algorithms for automated reasoning in order to address both plan generation and plan verification in our system. More specifically, the formalism, which is called $\mathcal{ALCK}_{\mathcal{NF}}$, is a *description logic* that has been used for modeling dynamic systems. The representation is based on the use of *concepts*, i.e., unary relations (predicates). Each *state* of the world is labeled by a set of concepts, corresponding to the properties that hold in that state. Moreover, the execution of actions is modeled through *roles*, i.e., binary relations between states.

We represent the robot's *knowledge* about the environment by means of a logical theory ($\mathcal{ALCK}_{\mathcal{NF}}$ knowledge base) $\Sigma = \Gamma_S \cup \Gamma_I \cup \Gamma_P \cup \Gamma_E \cup \Gamma_{DFR}$, where each Γ_x is defined below. In the following, we assume to deal with a set of primitive actions, partitioned into a set of *ordinary* actions, which are assumed to be deterministic actions with (possibly) context-dependent effects, and *sensing* actions, which are assumed to be actions able to sense boolean properties of the environment. We represent primitive actions in $\mathcal{ALCK}_{\mathcal{NF}}$ through a set of atomic role symbols, which we call *action-roles*.

State constraints (Γ_S) State constraints are used for representing background knowledge, which is invariant with respect to the execution of actions. We formalize state constraints as general $\mathcal{ALCK}_{\mathcal{NF}}$ axioms, not involving action-roles.

Initial state (Γ_I) The specification of the initial state in our formalization is given in terms of a formula of the form $C(init)$, where C is a concept, and *init* is the constant denoting the initial state. This axiom can be read as: C holds in the state *init* in every possible interpretation.

Precondition axioms for primitive actions (Γ_P) Precondition axioms specify sufficient conditions for the execution of primitive actions in a state. Precondition axioms are expressed as $C \Rightarrow (\mathit{Exists}\ R)$, where C is a concept representing the precondition and R is a primitive (either an ordinary or a sensing) action. This axiom can be read as: if C holds in the current state s, then there exists a state s' which is the R-successor of s.

Effect axioms (Γ_E) Effect axioms specify the effects of executing an action in a given state. We distinguish between ordinary actions and sensing actions. In our framework, for ordinary actions we specify an axiom $C \Rightarrow \mathit{Effect}\ (R, D)$, meaning that if in the current state the property C holds, then, after the execution of the action R, the property D holds. Moreover, effect axioms for sensing axioms are of the form $\mathit{True} \Rightarrow (\mathit{Effect}\ (R_S, D) \vee (\mathit{Effect}\ (R_S, \neg D))$, and specify the fact that, after the execution of the sensing action R_S, the robot knows whether the property D is true or false.

Frame axioms (Γ_{DFR}) In $\mathcal{ALCK}_{\mathcal{NF}}$ it is possible to enforce different forms of inertia laws, including *default frame axioms* which state that, if in the current state the property C holds, then, after the execution of the action R, the property C holds, if it is consistent with the effects of R. Such a rule can be represented in our framework by the axiom $C \Rightarrow \mathit{Effect}\ (R, \neg(\mathit{Consistent}\ C) \vee C)$, and call the set of default frame axioms in our specification.

Given a dynamic system specification in $\mathcal{ALCK}_{\mathcal{NF}}$ and a goal expressed in terms of a set of concepts, we are able to express the plan generation problem in terms of a reasoning problem in $\mathcal{ALCK}_{\mathcal{NF}}$ [5,2]. and implemented a plan generation/plan verification module based on such an automated reasoner.

5 Discussion

The main goal of the present work is the attempt to provide a new, effective way to drive the behaviour of the system based on a high-level specification of the actions that the agent is able to perform. In fact, a weakness of systems implementing similar approaches [10,6], is that the plans that are derived according to the formal specification are very loosely connected to the underlying execution layer. In this respect we have enriched the plan representation with additional information that can partly be extracted from the plan generation process and partly needs to be given as an additional specification. We have thus been able to implement a plan execution mechanism that relies on this richer representation and can take into account some of the aspects that cannot be adequately treated at the formal level, but are necessary in order to make the implementation effective. One consequence of the approach taken is that, in the design process, it is possible to interleave both the use of automatic plan generation and the direct specification of plans through a graphical interface.

The approach has been successful in two respects. First of all the system has been successfully implemented in our RoboCup teams both in the Legged and in the Middle-size League of RoboCup 1999 and 2000 and it performs well in a task (soccer playing) where the environment is much more dynamic as compared with other experiments described in the literature of Cognitive Robotics Systems (see for example [10,6]). The design of a system based on a formal high level specification has lead us to speed up the software development, when we have been able to port the system developed on a wheeled mobile base to the Sony legged platform.

References

1. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.
2. G. De Giacomo, L. Iocchi, D. Nardi, and R. Rosati. A theory and implementation of cognitive mobile robots. *Journal of Logic and Computation*, 5(9):759–785, 1999.
3. Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. 1992.

4. S. Hanks and R. J. Firby. Issues and architectures for planning and execution. In *Proc. of Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.
5. Luca Iocchi. *Design and Development of Cognitive Robots*. PhD thesis, Univ. "La Sapienza", Roma, Italy, 1999.
6. Y. Lesperance, K. Tam, M. Jenkin. Reactivity in a logic-based robot programming framework. In *Proc. of AAAI98 Fall Symposium on Cognitive Robotics*, 1998.
7. D. Nardi, C. Castelpietra, A. Guidotti, M. Salerno, and C. Sanitati. S.P.Q.R. In *RoboCup-2000: Robot Soccer World Cup IV*. Springer-Verlag, 2000.
8. D. Nardi et al. ART-99: Azzurra Robot Team. In *RoboCup-99: Robot Soccer World Cup III*, pages 695–698. Springer-Verlag, 1999.
9. N. J. Nilsson. Shakey the robot. Technical Report 323, SRI AI Center, 1984.
10. M. Shanahan. Reinventing Shakey. In *Proc. of AAAI98 Fall Symposium on Cognitive Robotics*, 1998.