Autonomous robot control using a three-layered architecture

Jacek Malec*
Department of Computer and Information Science
Linköping University
S–581 83 Linköping, Sweden
jam@ida.liu.se

Abstract

In this paper we present an approach to implementing behaviour-based control using a three-layered architecture. In this way we intend to combine the advantages of both approaches: flexibility of behaviour-based control specification with formal analysis and software generation tools developped for the layered architecture.

1 Introduction

The problem of designing reactive autonomous systems has been the subject of much attention in recent years. The research has focused on designing systems which would possess the following attributes:

- *reactivity*, in order to allow the system to cope with unpredictable changes in the dynamic environment while performing its mission;
- *robustness*, meaning the ability to function in a variety of situations, including failure of some of its subsystems;
- *selectivity of attention*, in order to effectively use existing resources of the system, such as computing power, or sensory equipment;
- *ability to pursue goals* either defined by the designer, or by the system itself.

*This research has been supported by the Center for Industrial Information Technology (CENIIT) and by the Volvo Research Foundation and the Volvo Educational Foundation.

One can distinguish several approaches to this problem. One of them is called "behaviour-based", because behaviour is the basic structure used for specifying control for an autonomous system. Another one is based on layered control architecture, where usually each layer has its own, well-defined task(s) to perform, often some software tools supporting implementation of those tasks, and some appropriate mechanism for passing control between the layers (or distributing it among them).

Those two approaches are usually opposed to each other: "Behaviourists" claim that a rigid architecture requires excessive computational power and enforce too many constraints on the actual design and implementation. The other group claims that behaviour-based approach is unsuitable for applications requiring either sophisticated control algorithms, or symbolic reasoning (as e.g. planning) procedures, or both. Both criticisms are to some extent justified.

In this paper we present an approach to implementing behaviour-based control using a three-layered architecture. In this way we intend to combine the advantages of both approaches. The paper is organized as follows: Section 2 presents the behaviour-based approach to autonomous robot control. Section 3 presents layered architectures and, in particular, a three-layered architecture developed at our university. In section 4 we discuss an algorithm for translating a behaviour-based specification of an autonomous system's control into a specification for the layered architecture. The last section contains the conclusions.

2 Behaviour-based control

The "behaviour-based design" paradigm is the result of work initiated and pursued at the MIT AI Lab by Brooks and his co-workers [5, 7, 8, 9, 24], and augmented by research conducted in other places [3, 15, 28]. The main idea consists of building a controller for an autonomous system out of a set of primitive reactive behaviours that couple sensory inputs of the system to its actuating outputs. Ultimately a system built this way should reveal the ability to cope effectively with dynamic and unpredictable changes in the real world while performing its task.

The major open problems of the behaviour-based approach, as stated by Brooks [8], are the following:

- "Understanding the dynamics of how an individual behaviour couples with the environment via the robot's sensors and actuators."
- "Understanding how many behaviours can be integrated into a single robot."
- "Understanding how multiple robots can interact as they go about their business."

These questions have been already addressed by the research done within the competing paradigm, which we might call symbolic-AI-based approach to the problem. The existing answers are, however, formulated with a totally different framework in mind, with the notable exception of Rosenschein and Kaelbling's research. In the behaviour-based approach, behaviours are the basic building blocks, and the problem of constructing a complex autonomous system amounts to the synthesis problem. In the symbolic-AI-based approach, a final result (i.e., a complex behaviour as a whole) is the starting point, and the design of a system can be seen as the analysis problem. Of course, both approaches use different tools, and, to some extent, even different languages. This raises the question of what common ideas they share, and what could be the starting point towards integration of both approaches.

Assuming that the (set of) system's behaviour(s) has been designed, one then has to solve the problem of choosing an appropriate architecture for the implementation. The bottom-up approach guarantees that effective realization is always possible — the system is physically grounded from the very beginning. The top-down design, on the other hand, provides systems proven to pursue top-level goals requiring non-trivial amounts of intelligence. However, the question whether systems possessing both these properties can be designed remains open.

It is worth noting that quite a lot of publications (see e.g. [11, 15]) stress very strongly the need to *constrain* the design so that it is realizable by simple, and thus fast, hardware. This leads to the third, to some extent orthogonal to the previous two, major aspect of design: the problem of choosing the best medium of implementation. One can imagine solutions ranging from general purpose machines with appropriate software implementing the adopted control strategy, to collections of specialized processors designed for specific purposes (processing visual data, motor control, sensor fusion), to purely hardware realizations of control algorithms, as is the case of some of the MIT robots.

While *emerging intelligence* is still the unachieved goal of the behaviour-based systems researchers, the property of being *autonomous* has definitely to be acknowledged to their designs. Within limits of capacity of their not too big batteries, subsumption robots deal effectively with cluttered environments, avoiding collisions, following people or other robots, picking soda cans, and following their urge to hide from the light or sound source. Autonomy, understood as the ability of a system to survive in an unknown, maybe even hostile, environment while pursuing its (usually not too complex) mission is the major achievement of the behaviour-based approach, unchallenged by any of the more classical, symbolic-reasoning-based systems.

If the goal is to design a system with a complex behaviour emerging from some interaction between low-level, more primitive ones then a better understanding of this interaction is needed. One of the most intriguing questions in this context is: How to find, for a given set of goals to be achieved, a set of primitive behaviours and a topology of their interconnections that would *provably* lead the agent towards these goals. The areas of behaviour-based specification of control and of goal achievement by execution of a plan (i.e. a series of actions) are quite separated, therefore not much research has been devoted to this problem yet. Some interesting work has recently been done in this direction by Saffiotti, Konolige and Ruspini [29, 30]. We will return to this problem later in this paper.

3 Three-layered software architecture

Our approach to autonomous real-time systems builds on the three-layered software architecture (shown in Figure 1) developed in our group [14, 26, 27] since 1986. The three layers are separated on the basis of the type of computations they are intended to perform. The first layer (*process layer*) performs periodic numerical computations at predetermined frequencies and would typically host the control algorithms, both for identification and for actuator control. The middle layer is called the *discrete response layer*, and computes the response to asynchronous events that are generated in (or rather recognized by) the process layer. One type of response can be to change the mode of the process layer due to the change of mode in the environment. The last layer (*analysis layer*) handles symbolic reasoning such as prediction, planning, and replanning. This layer would also host eventual learning mechanisms which might adapt the discrete response layer while maintaining its critical response requirements.

Literally dozens of layered architectures have been proposed recently for autonomous system implementation (see e.g. [2, 4, 10, 12, 31] for just some of them). Although there are differences both in the way of assigning various tasks to different layers and in the way the overall control of the system is executed, the general conclusion is that such layering is beneficial, if not necessary, in designing autonomous *intelligent* real-time systems. We do not claim that the architecture presented here is beneficial per se, rather our claim is that its advantage for the design of an autonomous agent is twofold:

1. By explicit separation of tasks requiring different conceptual and computational frameworks;
2. By providing a potential designer with a set of tools simplifying and systematizing the design process itself.

We would especially like to stress the latter property.

Each layer in our software architecture is supported by an implementation tool, or *engine*, which can be thought of as a restricted kernel managing the particular type of computation accommodated in that layer. So far, we have implemented engines for the *reactive subsystem* consisting of the process and discrete response layers. The work on an implementation of the third engine, suitable for implementing a restricted subset of reasoning processes, is close to its final stage.

The process layer is supported by a multi-threaded time-triggered real-time engine (Process Layer Executive) which may operate in several modes. The engine's real-time properties have been thoroughly analyzed [25] which resulted in a parameterized description of the system overhead automatically derivable from the application description in the Process Layer Configuration Language.

The engine of the discrete response layer maintains a set of symbolic state variables, or *slots*, which have rules associated with them. A rule specifies how a change in the value of the slot conditionally should change the value of other slots, which, in turn, may have associated rules. Thus, the arrival of an event that changes the value of a slot may

Figure 1: A layered software architecture.

start a forward-chaining process which eventually results in a new state, representing the response to the event. Computations in the asynchronous event-driven layer also have time-constraints, typically stating that the response to a particular event has to be computed within some defined amount of time.

We found that in general the systematic generation of a sound rule base for the discrete response layer was non-trivial. To facilitate this task we developed a formal graphical language, Process Transition Networks (PTN), which we used for description of complex traffic scenarios [17, 18, 22]. During the last two years we have provided a formal semantics also for some of the better-known languages for expressing behaviour-based control [18, 19, 16], among them the Behavior Language (BL) [20]. The result of our investigations is that programs in any of these languages can be rewritten using statecharts [13], what makes them realizable in the discrete response layer of our architecture and, moreover, gives the possibility of obtaining predictable control software.

4 Implementing behaviours

Neither of the tracks of research presented above has answered the question of vital importance for autonomous system design, namely: what is the mechanism of emergence of complex, goal-oriented behaviours out of a set of interconnected low-level ones. Usually, the behavioural design is made without any thorough analysis of the low-level computational processes underlying this attributed emergence. This is due to the fact that the complexity of the interaction among those processes might be enormous, given the real world environment and a large number of parameters influencing the interacting behaviours (cf. [20]) In other words, although one can create behaviour-based control for toy robots managing well in a laboratory setting, more complex behaviours are still outside the range of possibilities of this approach. Control systems for relatively simple robots in relatively simple environments are currently designed using the only existing methodology: "trial and error" [23]. However, when the size of the problem increases, the programmer has no reasonable methodology to support him in this design task. Trial-and-error would be too expensive in terms of programming man/time, cost of equipment damaged during failed trials and, last but not least, in terms of uncertainty. Even if, after a series of debugging sessions and tests, the robot appears to behave correctly, there is no guarantee that it will continue doing so.

In order to change this situation we have focused on analyzing the possibility of merging the behaviour-based specification with the set of analysis tools available for our architecture. The result of this investigations is an algorithm for translating a behaviour program (written in the Behavior Language [6]) into a program for the two layers (reactive subsystem) of our architecture. The translation consists of two steps, with statecharts [13] as the intermediate representation. With such translation we expect to get the possibility of implementing larger systems directly within the layered architecture paradigm, but with behaviours as the basic conceptual blocks for specification of control.

The two-step translation procedure described in [20, 21] yields, for any Behavior Language program, the corresponding RLCL program which can be directly processed by the tools associated with the discrete response layer of our architecture (with the reservation that all the timing associated with BL constructs such like `delay` or `monostable`, has to be implemented in the processes residing in the process layer. However, automatic generation of such code does not pose any particular problems.)

5 Conclusions

In this paper we have presented the possibility of translating Behavior Language programs into a rule base controlling the middle layer of our layered architecture. This procedure opens several interesting possibilities:

- It allows to implement large systems without worrying about the scalability: the tools associated with the layered architecture help the designer to analyze and maintain even large rule bases;

- It suggests the possibility of using formal specification and verification techniques for designing control for autonomous agents;

- It gives some hope that we will be able to get some insight into interaction among primitive behaviours: the resulting set of rules can be subjected to thorough formal analysis both in terms of consistency, completeness, deadlock-freeness, reachability, etc., but also in terms of timing properties;

- Some system-theoretic properties, such as controllability, observability, or stability, can be defined and analyzed for particular robots and chosen environments: this activity could not have been done within the framework of behaviour-based control alone.

The implementation of the algorithm is expected to be ready later this year. We intend to look then more carefully at the structure and complexity of the resulting rule base. This can provide some insight into the nature of behaviour emergence.

Acknowledgment

The research described in this paper is a collaborative work done by the former and current memebers of the RKL Laboratory, Department of the Computer and Information Science, University of Linkping. The author bears the responsibility for all omissions, misinterpretations and errors in the text.

References

[1] *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence.* Morgan Kaufman, 1991.

[2] J. S. Albus, H. G. McCain, and R. Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NIST Tecnical Note 1235, National Institute of Standards and Technology, Robot Systems Division, Center for Manufacturing Engineering, Gaithersburg, MD 20899, 1989.

[3] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[4] R. Peter Bonasso. Integrating reaction plans and layered competences through synchronous control. In *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence, Sydney* [1], pages 1225–1231.

[5] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2:14–23, 1986.

[6] Rodney A. Brooks. The behavior language; user's guide. Memo 1227, MIT AILab, April 1990.

[7] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[8] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence, Sydney* [1].

[9] Jonathan H. Connell. *A colony architecture for an artificial creature.* PhD thesis, MIT, 1989. AI Lab Tech Report 1151.

[10] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2719–2724, Nice, France, May 1992.

[11] Anita M. Flynn, Rodney A. Brooks, and Lee S. Tavrow. Twilight zones and cornerstones: A gnat robot double feature. AI Memo 1126, MIT, 1989.

[12] Erann Gat. Integrating reaction and planning in a heterogenous asynchronous architecture for mobile robot navigation. *SIGART Bulletin*, 2(4):70–74, 1991.

[13] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[14] Johan Hultman, Anders Nyberg, and Mikael Svensson. A software architecture for autonomous systems. In *Proc. 6th International Symposium on Unmanned Untethered Submersible Technology*, pages 279–292, 1989.

[15] Leslie Pack Kaelbling and Stanley J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6:35–48, 1990.

[16] Jacek Malec. On semantics of behaviour languages. Available also as Technical Report IDA-RKL-93-TR17, Department of Computer and Information Sciences, Linköping University.

[17] Jacek Malec. Passing an intersection, or automata theory is still useful. In Brian Mayoh, editor, *Proc. of the Scandinavian Conference on Artificial Intelligence – 91*, pages 258–265. IOS Press, 1991.

[18] Jacek Malec. Applied knowledge representation. *CC-AI: The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, 9(1):9–41, 1992.

[19] Jacek Malec. Complex behavior specification for autonomous systems. In *Proc. IEEE International Symposium on Intelligent Control'92*, pages 178–183, Glasgow, Scotland, August 1992.

[20] Jacek Malec. On formal analysis of emergent properties. In Christer Bäckström and Erik Sandewall, editors, *Current trends in AI planning*, pages 213–225. IOS Press, 1994.

[21] Jacek Malec. On implementing behaviours using a three-layered architecture. In Adam Borkowski and James L. Crowley, editors, *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*, pages 62–69, Grenoble, France, July 1994.

[22] Jacek Malec and Per Österling. Driver support system for traffic manoeuvers. In M. Bielli, G. Ambrosino, and M. Boero, editors, *Artificial Intelligence Applications to Traffic Engineering*, pages 283–318. VSP International Science Publishers, Zeist, The Netherlands, 1994.

[23] Maja J. Mataric. Basic tips for programming in the behavior language. Unpublished memo.

[24] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, MIT AI Lab, May 1990.

[25] Magnus Morin. Predictable cyclic computations in autonomous systems: A computational model and implementation. Licenciate thesis 352, Department of Computer and Information Sciences, Linköping University, 1993.

[26] Magnus Morin, Simin Nadjm-Tehrani, Per Österling, and Erik Sandewall. Real-time hierarchical control. *IEEE Software*, 9(5):51–57, September 1992.

[27] Magnus Morin and Erik Sandewall. A Software Architecture Supporting Integration of Sub-Systems. In Lars Asplund, editor, *National Swedish Symposium on Real-Time Systems*. Technical Report No. 30, Dept. of Computer Systems, Uppsala University, 1991. Also available as Technical Report LAIC-IDA-90-TR21, Dept. of Computer and Information Science, Linköping University.

[28] David W. Payton. Internalized plans: A representation for action resources. *Robotics and Autonomous Systems*, 6:89–103, 1990.

[29] Alessandro Saffiotti, Kurt Konolige, and Enrique H. Ruspini. A multivalued logic approach to integrating planning and control. Technical Note 533, Artificial Intelligence Center, SRI International, Menlo Park, CA, U.S.A., 1993.

[30] Alessandro Saffiotti, Enrique H. Ruspini, and Kurt Konolige. Blending reactivity and goal-directedness in a fuzzy controller. In *Proc. 2nd IEEE Conf. on Fuzzy Systems*, pages 134–139, 1993.

[31] Lee Spector and James Hendler. Knowledge strata: Reactive planning with a multi-level architecture. Technical Report CS-TR-2564, Department of Computer Science, University of Maryland, College Park, MD, November 1990.