Rigorous Design and Analysis of Behaviour Based Controllers

Martin D. Westhead
martinwe@aifh.ed.ac.uk
Department of Artificial Intelligence,
University of Edinburgh,
5 Forrest Hill, Edinburgh, EH1 2QL, UK

Abstract

One of the questions that arises with with the use of Behaviour Based Systems for robot control is the scalability of the approach. Networks of concurrent, interacting finite state machines will inevitably become harder to understand and construct as their complexity increases. This poster proposes that the use of formally defined control languages could ease such problems by removing language ambiguities and providing a basis for modularity and the formal proof of properties.

The Problem

Design Complexity

Behaviour based control systems, such as Brooks' Ghengis [Bro89], are composed of a large number of highly parallel, interacting units. Ghengis was composed of some 56 finite state machines, of up to six states each, which were connected by means of a complex network of wires. The hexapod's behaviour allowed it to walk over rough terrain, and follow a slow moving heat source. Whilst the operation of Ghengis is quite impressive for its robustness, the task it achieves is not really very complicated. If we are interested in building a truly autonomous agent capable of interacting with human beings in a real environment, carrying out complex tasks and adapting to a changing world then we have to face the possibility that this may require a controller several orders of magnitude greater in complexity than Brooks' six legged walker. Such control systems could eventually rival complexity of modern VLSI chip designs. In order to make the construction of such systems feasible a number of design issues will have to be addressed and it is not clear that current methodologies and tools will be up to the task. There are at least two ways in which the complexity could be considered, in the design of systems, and then in their debugging.

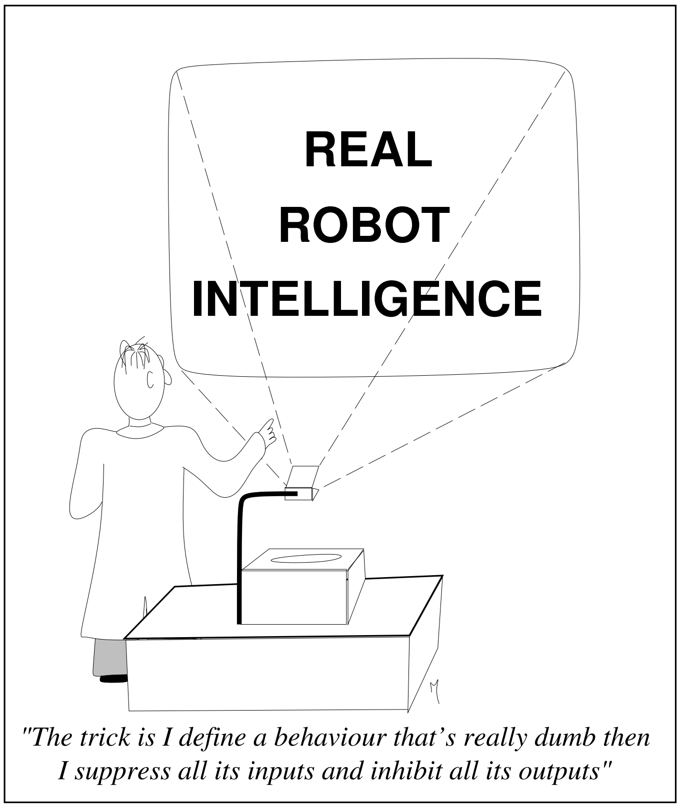

Figure 1: Its not always clear how someone else's system works

System complexity

Assuming that scaling up the sort of parallel networks of finite state machines we are currently using is the way to build robot controllers, then the complexity of the controllers that it will be feasible to design and construct will become limited by our ability to understand them. Formal systems can provide unambiguous languages with clearly defined module structures and automated tools for manipulating the controllers and proving properties about them. This could provide a language for communicating details of systems between programmers or researchers, as well as a potential basis for the development of theories and design principles about behaviour based systems.

Bugs

A bug in a robot control system manifests itself as an *error* : some sort of unexpected (and usually undesirable) behaviour on the part of the robot. Such bugs can be divided into two distinct categories:

1. *Faults* i.e. bugs that occur due to mistakes in the coding e.g deadlock, unexpected behaviour resulting from errors in the controller.

2. Bugs that are the result of unexpected aspects of the interaction of robot and environment.

"C'mon Al, what do you mean the security robot's malfunctioning?"

Figure 2: It's difficult to ensure that you have tested for all eventualities

The second of these is a result of the unpredictable nature of the real world. The iterative methodology [Bro89] advocated for the design of situated agents is one way of approaching this problem[1].

Faults however are not effectively addressed by this methodology. A complex system built from a large number of interacting subsystems is particularly prone to unpredictable and intermittent errors present in the controller. For example without exhaustive testing there is no guarantee that Ghengis won't try to lift all six legs off the ground at the same time.

Whilst in principle it may be possible to uncover such errors by testing alone, the greater the complexity of the controller, the less practical this becomes. Exhaustive testing can quickly become impractical as the number of potentially different inputs can grow exponentially in the number of sensory states stored by the robot. Without the possibility of exhaustive testing it becomes very difficult to be sure if all the possible singularities and limiting cases in the controller's implementation have been properly investigated. It is for this reason that safety standards are increasingly requiring the use of formal methods and mathematical proofs in the verification of safety critical control systems.

In the case of behaviour based robots, as with VLSI chip design, it is the complexity, or at least the potential complexity of control systems that makes the use of formal methods attractive. The tools and techniques that are available could significantly increase our understanding of the control systems we are building, and the way they should be built.

[1]Though it has been pointed out [Mal93, NT93] that existing mathematical control models could be applied in some cases and this could reduce the number of design iterations required.

A Solution?

A Formally Defined Compiler and tool suite

This work is in the early stages of defining and building robot control systems based on formally defined systems such as CCS [Mil89], Lustre [HCRP91], Esterel [BG88] and Argos [Mar91]. These all have associated automatic tools for analysis and verification, allowing a different perspectives on the flow of signals and data through the system, and the advance of time. The latter three have compilers available so controllers built from these languages can be translated directly into C code and run on a real robot.

CCS : This is a highly abstract language for representing asynchronous, concurrent processes. Actions within a CCS agent are communications, either internal or with the environment. The notion of state is implied by these transactions. CCS has been implemented as an extension to ML called LCS.

Lustre : Like the remainder of these languages, Lustre is a synchronous language, designed to form the operating kernel of a reactive control system. Unlike the others it is a dataflow language; it is declarative in nature, so a programs output is expressed as a function of its input.

Esterel : Esterel compiles down to the same intermediate finite state representation as Lustre and Argos. It is an imperative language, with explicit representations of watchdogs, timeouts and triggers.

Argos : Argos is another declarative language, but this time represented in a graphical form much like statecharts [Har87]. Work has been done to allow Argos to integrate with Kronos a tool for reasoning about real time operation [JMO93].

Interacting systems

It is crucially important in robot design to be aware of the many systems and subsystems that are being dealt with. A point which the Behaviour Based philosophy underlines is that "robot + environment" is a system. Equally "robot controller + robot hardware" is a system, as is a controller made up of interacting behaviours. It is important that controllers are constructed in a language that reflect this interaction. There are at least three different levels, at which analysis could be carried out:

- Analysis of an isolated subsystem. To determine certain properties of an individual behaviour, or an isolated group of behaviours.
- Analysis of the integration of a new subsystem into an existing supersystem, e.g. to check for the introduction of specific, undesirable behaviours.
- Using control theory techniques to analyze certain aspects of a robot controllers interaction with a mathematical plant model. For example integration with existing work on hybrid control systems such as the work of Nadjm-Tehrani [NT93].

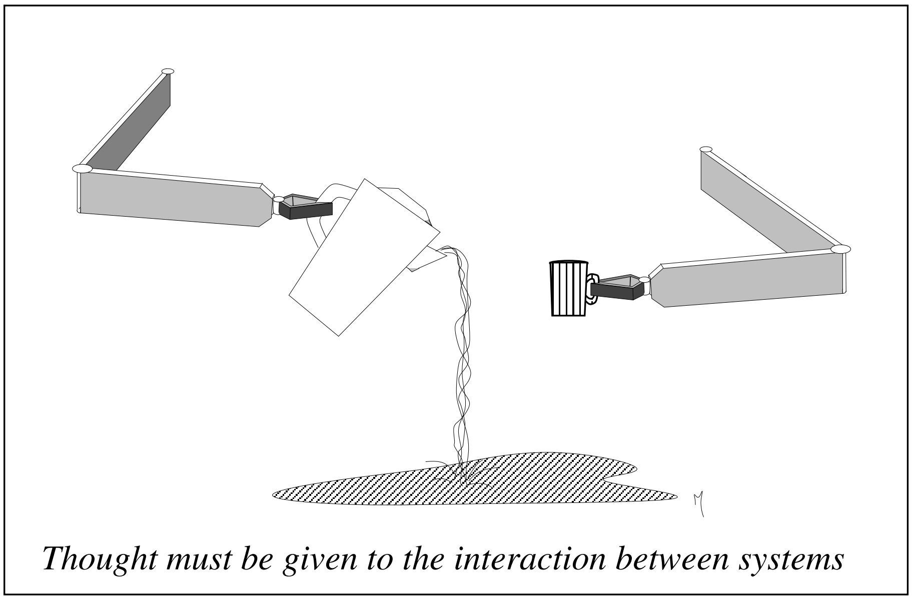

Figure 3: The complete behaviour of a system cannot be determined simply by understanding the parts

At each level, particularly the last, a complete analysis will be impractical in most non-trivial cases. However it will be practical to check for the presence and absence of particular desirable (and undesirable properties) and thus to aid the programmer's understanding of a program's operation.

Formal analysis and environmental assumptions

It is argued by some workers in Behaviour Based Systems that it is misguided to consider a robot controller in isolation. The controller is only part of a much larger system which includes the environment, and an isolated analysis allows us to conclude very little about the robot-environment interaction. However it is legitimate, and useful, to consider sub-systems in isolation so long as they are recognised as such. A controller in isolation from a robot can be usefully analyzed without the need for making any assumptions about the environment.

For example, in a robot such as Ghengis, if it were implemented in a formally defined language there are a number of properties that could be established. We could ensure that the controller only ever asked the robot to raise at most (say) two legs at any one time, and further that legs that had been signaled to lift, would then be asked to move forward, and legs that were signaled to remain on the ground would be asked to moved backwards. It is important to point out that this approach could not produce a proof that the robot can walk, only that the necessary control signals would be there for it to do so. Properties such as these could then be verified at each stage of the iterative construction. When extra behaviours are added on top, the programmer could ensure that these properties still held, before the code was even compiled.

Conclusion

This poster represents a statement of work in progress, there is a great deal of work to be done to investigate the ideas outlined here. However, it seems certain that as behaviour based controllers increase in complexity, more rigorous design principles and methodologies will have to be introduced. The formal systems presented here could provide a theoretical basis for such developments.

References

[BG88] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. Technical report, INRIA, 1988.

[Bro89] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. A.I. Memo 1091, Massachusettes Institute of Technology Artificial Intelligence Laboratory, February 1989.

[Har87] David Harel. On the formal semantics of statecharts. In *Proceedings of the second IEEE Symposium on Login in Computer Science*, pages 54–64, 1987.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[JMO93] M. Jourdan, F. Maraninchi, and A. Olivero. Verifying quantitative real-time properties of synchronous programs. In LNCS 697 Springer verlag, editor, *Fifth Int. Workshop on Computer Aided Verification (CAV), Elounda (Crete)*, 1993.

[Mal93] Jacek Malec. On formal analysis of emergent properties. Research Report LiTH-IDA-R-93-45, Department of Computer Science and Information Science, Linköping University, Sweeden, 1993.

[Mar91] F. Maraninchi. Argos : a graphical synchronous language for the description of reactive systems. Spectre report c29, LGI-IMAG, Grenoble, mar 1991.

[Mil89] Robin Milner. *Communications and Concurrency.* Prentice-Hall, 1989.

[NT93] Simin Nadjm-Tehrani. *Reactive Systems in Physical Environments.* PhD thesis, Department of Computer Science and Information Science, Linköping University, Sweeden, 1993.