On semantics of behaviour languages

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

May 8, 1996

Abstract

The paper focuses on semantics of behaviour languages. Three such languages are briefly introduced (the Behavior Language, REX, and PTNs) and their common semantics is discussed using the Statecharts formalism. The goal of this research is to address the need for augmenting the "behaviour-based design" paradigm with theoretical tools both adequate for expressing complex behaviours of autonomous reactive systems pursuing high-level goals, and applicable to both analysis and synthesis tasks for such systems. The analysis of the behaviour languages' semantics is a first step towards this goal.

1 Introduction

A great deal of research on autonomous systems design is done within the "behaviour-based" paradigm. This paradigm is the result of work initiated and pursued at the MIT AI Lab by Brooks and co-workers [Bro86, Bro90b, Bro91, Con89, Mat90], and augmented by research conducted in other places [Ark90, Pay90, KR90]. The main idea consists of building a controller for an autonomous (possibly intelligent) 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 results obtained so far, both at MIT and at other places, although interesting, raise some doubts about the applicability of this methodology to complex systems

*This research has been supported by the Center for Industrial Information Technology (CENIIT).

design. Only simple systems have been created so far and no sound design procedures have been proposed for more demanding cases.

This paper is an attempt to present some of the existing languages for specification of an autonomous agent's behaviour (Section 2). It also proposes a method for comparing their expressive power by formally studying their semantics using statecharts as a reference formalism (Section 3).

2 Overview of behaviour languages

In this section we have chosen to briefly present three specification methods: the Behavior Language, the situated automata approach (REX), and the Process Transition Networks.

2.1 The Behavior Language

The Behavior Language (BL) is a programming language with LISP-like syntax for specifying behaviour-based controllers [Bro90a]. Using the constructs of the language one can define a set of behaviours comprising the control system of a robot. Such a set of behaviours is usually intended (by the designer) to make the robot achieve some predefined set of goals. However, those goals are only *implicit* in the specification (thus the name "emergent functionality"). We will not present the full syntax of BL here, rather we will limit ourselves to some crucial elements of the language.

BL is based on the assumption that every behaviour is expressible as a set of so-called "real-time" rules. A real-time rule is an expression of the form:

(`whenever` *condition* `&rest` *body-forms*)

or

(`exclusive &rest` *whenever-forms*)

and is executed independently of other rules (at least in the ideal case), in real-time, possibly by an Augmented Finite State Machine (AFSM). An AFSM is a finite state machine equipped with registers and timing elements.

The *condition* field in a real-time rule can be occupied by one of the following:

`t` stating that the body of the rule will be unconditionally evaluated every certain amount of time (called the *characteristic time* of the system);

monostable the condition is true for the duration of the triggering of the named monostable;

`(delay` τ`)` like condition `t` above, but overriding the characteristic time by τ;

`(received?` *register*`)` the condition is true if a message has been deposited in the named register since the start of waiting in this `whenever` clause;

`(not|and|or` *&rest forms*`)` a restricted set of logical combinations of the conditions described above is also possible.

Observe that all those conditions refer either explicitly or implicitly to some timing element. On the other hand, there is an explicit assumption that any of those timers is not synchronized with any other, i.e., they are totally independent.

Behaviours are defined either as stand-alone real-time rules or as groups of such rules. A stand-alone rule is named by:

(`defmachine` *name declarations rule*)

and complex behaviours are constructed using the `defbehavior` facility:

(`defbehavior` *name inputs outputs declarations rules*).

Then input and output registers of behaviours can be connected by "wires" according to the designer's intentions:

(`connect` *source dest1* `&rest` *more-dests*)

`connect` statement is also used to implement suppression and inhibition mechanisms.

The core idea of the behaviour-based approach is an asynchronous, independent activation of real-time rules and behaviours. The following mechanisms are used for activating behaviours [Bro90a]:

- direct activation by some condition,
- thresholding,
- hormone system (conditions, releasers),
- spreading of activation.

We end this short presentation at this point, asking the reader interested in details to refer to the original document [Bro90a].

2.2 Situated automata

A somewhat more constrained approach to the problem has been proposed by Rosenschein and Kaelbling [Ros89, KR90]. Their work is based on the concept of situated automata (Figure 1, [Ros85]), where both an autonomous agent (or, more generally, a system) and its environment are modeled as finite state automata. This allows to formally express dependencies between the state of the environment and the state of the system. Kaelbling and Rosenschein consider the information about the environment carried by the states of the system, and use it to formally prove the correctness of the system's design.

On the basis of the idea of situated automata Kaelbling and Rosenschein have created a hierarchy of languages: REX, which provides a link from a set of reactive behaviour rules (as in the Behaviour Language) to hardware implementation; GAPPS, which is used for defining goal-oriented behaviours in the action part of the system (cf. Fig. refsitaut); and RULER, which is used for describing the perception component of the system. REX [RK86] is defined in LISP and serves for specifying a "low-level description" of a machine. Using an appropriately defined logic which accepts specifications computed by REX, one

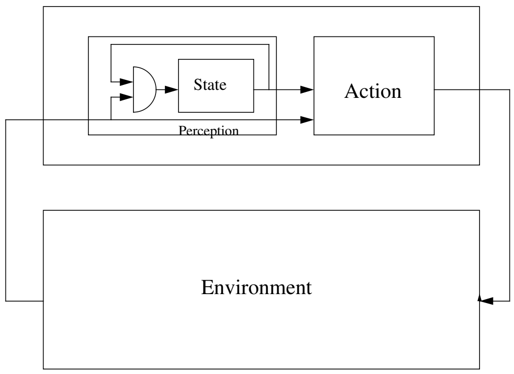

Figure 1: An automaton situated in its environment.

can prove properties of the behaviour of the designed machine. GAPPS [KR90] and RULER [Ros89] expressions compile down to REX statements, so all the theoretical machinery can be applied to GAPPS and RULER programs as well. Using these tools, one has a warranty that having specified the set of desired system's interactions with the world, *and having completely specified the possible set of situations occurring in the world, as perceived by the sensory input of the system*, the system will actually behave in the desired way, and, moreover, it will be compilable down to the gate level. However, the requirement of complete specification of all possible situations the system might find itself in (or might recognize), together with the limitations imposed on the specification languages in order to make them compilable to hardware, causes this approach to be somewhat restrictive.

2.3 Process Transition Networks

Yet another language created for the purpose of specifying behaviours of autonomous agents are Process Transition Networks (PTN, [Mal91b]). They have been assigned a formal semantics [Mal91a], and tools for implementing the control strategy represented by using PTNs have been created. PTNs enable designers to simultaneously specify behaviours of several independent agents *and* their environment. They are compilable (after making some necessary assumptions) to automata that describe behaviours of separate agents [SM91], and they can also be used to explicitly couple the goals of an agent to changes of its behaviour. They have the same expressive power as statecharts, what has been show in [Mal92b].

In the PTN language we introduce the primitive notion of a *process*, (corresponding to state in FSA). A process will be depicted by a box. We are interested in those moments in which discontinuities of a property occur, because discon-

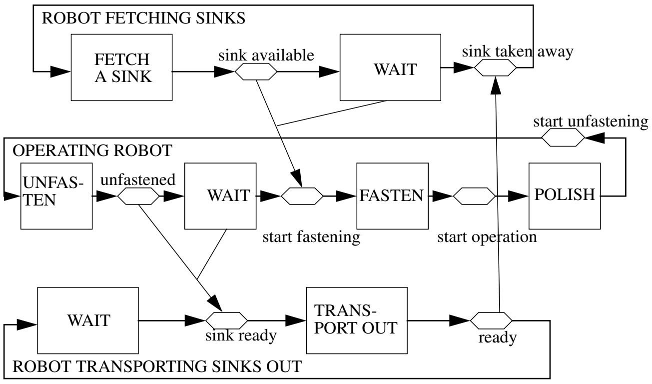

Figure 2: Three cooperating robots.

tinuities convey important information about changes of the current situation. We will call such discontinuities *events.* Events are represented in the PTN language by hexagonal boxes. Since discontinuities (by definition) may only appear between periods of continuous change of some property, events only appear between two processes. Such a *transition* (from process to event to another process) is denoted by two thick arrows, the first pointing from the "source" process to the event, and the second pointing from the event to another process.

The most important notion introduced in PTNs, Causal dependencies between discontinuities, are represented by *Forcing* and *enabling arrows*, denoted by thin lines and pointing either from an event alone or from an event and its subsequent process, respectively, to some other event. A simple PTN constructed from these primitives is shown in Figure 2. This PTN models the behaviour of a system consisting of three robots cooperating during the finishing phase of sink production.

In the PTN language, one may combine causality arrows using both **and** and **or** connectives. The **and** of, say, two enablings is expressed by two consecutive events on one transition between two processes. The **or** may be represented with two different paths leading from one process to another, but is usually depicted by merging the enabling arrows themselves.

The formal definitions of syntax and semantics of PTNs have been presented in detail in [Mal92a].

3 Semantics

In order to analyze and compare behaviour languages we had to choose some common formalism which would be general enough to entail all of those lan-

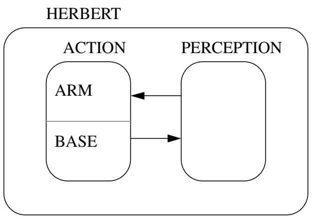

Figure 3: A higraph representing a hypothetical control system for a robot called HERBERT. The blob ACTION is a cartesian product of the blobs ARM and BASE.

guages and, on the other hand, sophisticated enough to express interesting properties. For this purpose we have chosen Statecharts.

3.1 Statecharts

An interesting language for specifying behaviours of reactive systems is the statecharts formalism [Har87] or its more generic incarnation, higraphs [Har88]. Statecharts are used to define complex, hierarchical finite state automata. Therefore, they seem to be a very plausible candidate for specifying autonomous agent's behaviour.

Due to their formal grounding in automata theory, statecharts have strong connections to the situated-automata approach of Kaelbling and Rosenschein; however, no tools are provided in the formalism for expressing goals of a system acting in some environment.

We begin the presentation of statecharts by quoting the formal definition of the more general construct of a higraph [Har88].

Definition. *A* higraph *is a quadruple* $H = (B, \sigma, \pi, E)$ *where* B *is a finite set of* blobs *and* E *is a set of* edges *(a binary relation on blobs). The* subblob *function* $\sigma : B \to 2^B$ *assigns to each blob* $x \in B$ *the set* $\sigma(x)$ *of its subblobs; it is assumed that this function is cycle free. The function* $\pi : B \to 2^{B \times B}$ *introduces an equivalence relation on blobs; these classes of equivalence for a given blob* x *will be denoted as* $\pi_i(x)$*. The* atomic blobs*, denoted by* A*, are those that do not have any subblobs (see Figure 3).*

Statecharts are a particular subset of higraphs, namely the ones in which the E relation corresponds to the transition function of an automaton. Some of the more advanced constructs of statecharts, such as e.g. *history-dependent* transitions, require an additional relation on blobs to be defined; however, we will not concentrate here on the formal aspects of the language.

The basic property distinguishing statecharts from other tools for describing automata is their ability to easily capture hierarchies and abstractions. An important related notion which we would like to present here is *orthogonality*.

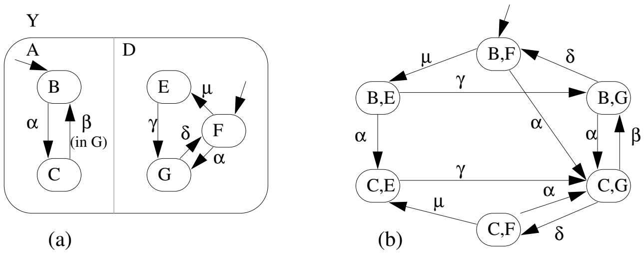

Figure 4: Orthogonality in statecharts.

It has already appeared in the formal definition of higraph in the form of the function π. Its meaning is the following: in the set of the states of a system, we can distinguish several subsets describing subsystems working in parallel. In terms of the state space this means that each global state is a tuple of local states of all subsystems, as illustrated in Figure 4. Subfigure (a) presents a blob (Y) divided into two orthogonal subblobs (A and D). The corresponding "flat" version of the state space is presented in subfigure (b). Note that the number of states and transitions in a non-orthogonal description is exponential with respect to the number of orthogonal components of the system.

Figure 4 (a) illustrates one more important feature introduced in statecharts: a *conditional transition.* The label β(in G) means that the transition (in this case from C to B) occurs only in case of input letter β appearing in the input **and** the subsystem D being in state G.

The last concept which needs to be mentioned here is the delay element. It is the only part of the statechart formalism actually extending it beyond the limitations imposed by classical finite-state automata. On the other hand, its power is exemplified by the fact that it is extensively used in real-time-related applications, e.g., VLSI specification [DH89].

Due to the introduction of timers the semantics of the full statechart formalism has to be defined using temporal constructs (such as e.g., timed traces or timed automata).

3.2 Behaviour languages vs. statecharts

We have chosen statecharts as the target formalism for semantical analysis of behaviour languages because they are powerful enough to represent the constructs of all the introduced languages. Some analysis of the embeddability of behaviour languages into statecharts has already been done [Mal92b, Mal94]. Its result show that for each of the original languages, a program (specification) translated into statecharts will look as depicted on Figure 5. This Figure illustrates some specific constructs of BL, where a behaviour corresponds to a state

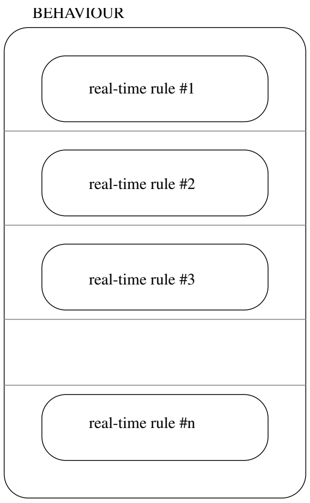

Figure 5: A BL (REX, PTN) behaviour.

with its rt-rules as parallel sub-states. The input and output registers of behaviours can be connected by *wires*. Such connection can be represented in the statechart formalism by an internal event generated by the state change of the output register and affecting the state change of the input register. To include the suppression and inhibition mechanisms, this construct should be modified accordingly.

The above observation is valid generally, i.e. for programs in either of the presented behaviour languages we can obtain an equivalent statechart (i.e. a timed FSA) with the same semantics.

4 Conclusion

We have shortly presented three languages used for specifying behaviours of reactive autonomous systems. The common factor is that all those languages are based in some way on finite state automata. So, an appropriate FSA-based formalism (statecharts) has been chosen to analyze the semantics of the introduced languages. Our research (reported elsewhere) shows that all the languages' constructs can be translated into appropriate statecharts.

The goal of this short presentation was twofold: to sketch the state of research in the area of specifying behaviours for autonomous agents, and to identify the existing constraints within which we are investigating the following problems:

1. the expressive power and limitations of specification tools used so far;
2. compositional specification of behaviours, so that from a set of specifications of simpler behaviours one could formally derive a specification of their composition;
3. decompositional specification of behaviours, so that specification of a complex behaviour would allow one to infer a set of specifications of hardware-realizable primitive behaviours, or at least would yield predictable software fitting some layered architecture;
4. theoretical tools for modeling dynamics of a single behaviour;
5. theoretical tools for modeling dynamics of a system of coupled behaviours.

References

[Ark90] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2:14–23, 1986.

[Bro90a] Rodney A. Brooks. The behavior language; user's guide. Memo 1227, MIT AILab, April 1990.

[Bro90b] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[Bro91] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence, Sydney.* Morgan Kaufman, 1991.

[Con89] Jonathan H. Connell. *A colony architecture for an artificial creature.* PhD thesis, MIT, 1989. AI Lab Tech Report 1151.

[DH89] Doron Drusinsky and David Harel. Using Statecharts for Hardware Description and Synthesis. *IEEE Transactions on Computer-Aided Design*, 8(7):798–807, July 1989.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.

[KR90] Leslie Pack Kaelbling and Stanley J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6:35–48, 1990.

[Mal91a] Jacek Malec. Process Transition Networks: A formal graphical knowledge representation tool. In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6th International Symposium*, pages 193–202. Springer-Verlag, October 1991. Lecture Notes in Artificial Intelligence 542.

[Mal91b] Jacek Malec. Process Transition Networks: What Are They For? In *Proceedings of the IEEE Systems, Man and Cybernetics Conference, Charlottesville, VA*, pages 1177–1182, October 1991.

[Mal92a] 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.

[Mal92b] Jacek Malec. Complex behavior specification for autonomous systems. In *Proc. IEEE International Symposium on Intelligent Control'92*, pages 178–183, Glasgow, Scotland, August 1992.

[Mal94] 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.

[Mat90] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, MIT AI Lab, May 1990.

[Pay90] David W. Payton. Internalized plans: A representation for action resources. *Robotics and Autonomous Systems*, 6:89–103, 1990.

[RK86] Stanley J. Rosenschein and Leslie Pack Kaelbling. The synthesis of digital machines with provable epistemic properties. In Joseph Halpern, editor, *Theoretical Aspects of Reasoning about Actions*, pages 83–98, Monterey, CA, 1986.

[Ros85] Stanley J. Rosenschein. Formal theories of knowledge in AI and robotics. *New Generation Computing*, 3:345–357, 1985.

[Ros89] Stanley J. Rosenschein. Synthesizing information-tracking automata from environment description. In *Proceedings of the First International Conference on Principles on Knowledge Representation and Reasoning*, pages 386–393. Morgan Kaufman, 1989.

[SM91] Hua Shu and Jacek Malec. From Process Transition Networks to Behavior Automata. In *Proceedings of the 1991 IEEE International Symposium on Intelligent Control, Arlington, VA*, pages 257–262. IEEE Control Systems Society, 1991.