Use of Cognitive Robotics Logic in a Double Helix Architecture for Autonomous Systems

Erik Sandewall
Department of Computer and Information Science
Linköping University
S-58183 Linköping, Sweden
erisa@ida.liu.se

Abstract. This paper addresses the two-way relation between the architecture for cognitive robots on one hand, and a logic of action and change that is adapted to the needs of such robots on the other hand. The relation goes both ways: the logic is used within the architecture, but we also propose that an abstract model of the cognitive robot architecture shall be used for defining the semantics of the logic.
For this purpose, we describe a novel architecture called the *Double Helix Architecture* which, unlike earlier proposals, emphasizes a precise account of the metric discrete timeline and the computational processes that take place along that timeline. The computational model of the Double Helix Architecture corresponds to the semantics of the logic being used, namely the author's Cognitive Robotics Logic which is based on the 'Features and Fluents' theory.

1 Introduction

This paper addresses the two-way relation between the architecture for cognitive robots on one hand, and a logic of action and change that is adapted to the needs of such robots on the other hand. The relation goes both ways. There is widespread agreement that a logic-based deliberative system is one necessary part of the robot architecture, but in addition we propose that an abstract model of the cognitive robot architecture should be applied to defining the semantics of the logic being used. Essential notions in the logic of the cognitive robot, such as the concepts of 'state' and 'action', the success or failure of actions, and even the notion of time within which observations are made and actions are performed - all of these notions are pertinent for both the logic and the system architecture.

1.1 Logic and Architecture

The logic being used here is the author's Cognitive Robotics Logic (CRL) [20] which is based on his earlier work on 'Features and Fluents' [17]. CRL is closely related to Doherty's Time and Action Logic (TAL) [4] and, somewhat more remotely, to the modern event calculus [23]. It is characterized by the use of explicit, metric time that allows for concurrent actions, actions of extended, overlapping duration, combinations of continuous and discrete change, characterizing the range of precision in sensors and actuators, and more. The approach presented here can probably be easily transferred to other logics in the same group. It appears that it can not easily be transferred to logics without metric time, such as the situation calculus [8], since the modelling of low-level, real-time processes is an important part of our enterprise.

We also present a novel architecture, called the Double Helix Architecture (DHA) that differs from previously proposed robotic architectures by being much more specific with respect to the time axis and its associated computational processes. It is common to define robot architectures in terms of graphs consisting of boxes and arrows, where the boxes denote computational processes and data stores, and the arrows denote data flow or control flow. Such diagrams abstract away the passage of time, which means that they are not sufficient for characterizing the semantics of a logic of time, actions, and change. The DHA will therefore be presented using two complementary perspectives, including both the traditional data-flow diagrams and the new time-axis diagrams.

The actual Double Helix Architecture contains more details than can be described in an article of the present size, so our account must be limited to the most salient aspects.

1.2 Implementation

The work on the Double Helix Architecture is a separate and rather small part of the WITAS project which aims at the construction of an intelligent helicopter UAV (Unmanned Aerial Vehicle)[1] [3] as well as research on a number of related technologies. The on-going and very large implementation effort for the on-board system in the WITAS helicopter is described in [5]. The DHA is a concept study and an experimental implementation; we wish to make it clear that the implementation of DHA is not integrated in the main implementation effort in the project. Also, similarities in design between the main WITAS system and the DHA are due to common background, so the design described here should not be interpreted as a description of the main WITAS system.

The priority for the DHA study is to obtain a design and an implementation that are as simple as possible, even at the expense of considerable

[1] `http://www.ida.liu.se/ext/witas/`

idealization, in order to make it possible to establish a strong relation between the design and the corresponding logic. In addition, the DHA implementation provides a simulated environment of UAV and ground phenomena that is used for the continued development of a user dialogue system for WITAS[7]. It will be further described in section 9.

2 Assumptions

Since DHA is presently tried out in a simulated environment, it is very important to make precise what are the assumptions on the forthcoming situations where the system will serve an actual robot. Every simulation must be a simulation *of* something concrete.

2.1 Assumptions on the robotic system

We focus on cognitive robotic systems with the following characteristics:

- They control mechanical robots (rather than e.g. 'web robots') and in particular, vehicles using computer vision as one important sensor and source of information about its environment.
- They have strict real-time requirements on them, but different aspects of the behavior operate on different time-scales, which makes it appropriate to use a layered architecture. In the case of our UAV application, it takes after all a while to fly from point A to point B, so there is room for computations that may take many seconds or even several minutes, if need be, besides other processes that operate on fractions of seconds as you would expect in an aircraft.
- Besides the robotic vehicle itself, which typically has a fixed set of components, the robot must also be able to represent a set of observed objects that changes over time, including for example roads, automobiles, and buildings on the ground (in the case of the UAV). Each such observed object is assumed to be within the robot's field of vision for a limited period of time, during which its properties can be identified although with limited precision and reliability. Dealing with such observed objects, their characteristics, and the events in which they are involved is a strict requirement on both the architecture and the logic.
- The robot must be able to communicate with persons or with other robots in order to give and receive knowledge about itself and about observed objects. This means, in particular, that its knowledge representation must support the way human operators wish to understand and talk about observed objects and their characteristics and

processes, as well as of course the actions of the robot itself. Furthermore, this consideration suggests that the chosen logic for cognitive robotics should be able to represent the partial knowledge of each of several agents within one body of propositions.
– In particular, the actions of the robot should not only be represented as simple actions, such as 'fly to position (x, y, z) and hover there'. The operator may also e.g. wish to inquire about how the flight has been performed or will be performed, or impose restrictions on trajectory, velocity, altitude, etc.

There are a number of other requirements that must be made in the case of the WITAS application, or any other concrete application, but the present list will be sufficient for defining the direction of the present work.

2.2 Assumptions on the deliberative system

We focus on a deliberative system that is capable of the following functions, all of which refer to the use of actions and plans:

– Execute a previously defined plan, taken from a plan library, with consideration of real-time constraints. From the point of view of the cognitive system, 'executing a plan' means to communicate the plan, piecemeal or in a single shot, to the robot control system and to receive the corresponding feedback from it.
– Generate a plan for achieving a given goal.
– Interleave planning and plan execution, again taking real-time constraints into account. Planning consumes real time.
– Engage in dialogue with one or more users, which also may involve the use of, and the making of plans for dialogue actions. Communicate its current plans and the motivation for its past, current, and planned actions, in dialogue with these users or operators.
– As a long-term direction, the system should also be able to 'learn by being told', that is, to receive high-level advise about how it may modify its own behavior.

We also emphasize crisp, true/false distinctions rather than graded certainty. Some of the constructs described here will rely on uncertainty management on lower levels of architecture, but we assume that this is a separate issue. This is natural in particular because of the priority that is made on the dialogue aspect.

3 Related work

The topic of 'robotic architecture' recurs very frequently in the literature about A.I. robotics. The concept of architecture itself is not very precise. Often, such as in Dean and Wellman's book on Planning and Control [2], an architecture is seen as a prescription for how a number of modules or subsystems are to be assembled. Architectures in this sense are typically characterized using block diagrams, as was mentioned above. Others, such as Russell and Norvig in their standard textbook [14], view an architecture as a computational infrastructure that can be 'programmed' to perform the desired cognitive tasks. In this case the architecture may include both the robotic hardware, its software drivers, and implementations of specialized languages such as RAPS [6] that provide a high-level platform for building applications.

Our use of the term 'architecture' in this article is closer to the one of Russell and Norvig than the one of Dean and Wellman inasmuch as we emphasize the importance of being able to 'program' a generic software system by attaching and replacing software modules (plug-ins), written both in general programming languages and in specialized languages including, but not restricted to, logic.

In yet another interpretation, the word 'architecture' refers in a comprehensive way to all the significant design choices and facilities on all levels of a complex system. That is not the way we use the term here.

The notion of multi-level architectures gained acceptance within A.I. during the 1980's [1, 22], superseding earlier and often simpler notions of a sense-deliberate-act cycle. Robotic architectures in modern A.I. usually emphasize the need for the different layers to operate concurrently, so that low-level and intermediate-level sensori-motoric processes can proceed while high-level deliberation is allowed to take its time. Still there is not much conceptual integration between the different levels; the procedural languages that are often proposed for use on the higher levels of these systems do not usually have the expressivity that is required e.g. for characterizing the precision of sensors, or for specifying and adding to control laws. Please refer to [10] and to Chapter 2 in [11] for surveys of architectures for intelligent robots.

The extensive research on logics of actions and change includes several approaches that attempt to bridge the gap between the 'lower' and the 'higher' levels of robotic performance. This includes in particular the work by Rao, Georgeff et al on BDI architectures [13], the work by Reiter, Levesque, et al on GOLOG [9], Shanahan's work on robot control using the modern event calculus [23], and Nilsson's triple-tower architecture [12]. (The related topic of planning domain description languages is also potentially relevant in this context, although action planning in A.I. has until recently tended to stay away from the challenges of cognitive robotics). However, none of the mentioned approaches contributes to the clarification of the semantics of actions in terms of more elementary principles, which is a major concern of the present work. Also, with the possible exception of Nilsson's work, in all of these approaches the architecture seems to be chosen as an implementation of the logic being used. In our work we wish to see the architecture as a way of orchestrating the cohabitation between deliberative processes and conventional, procedural computation.

Two aspects of our own earlier work are used for the Double Helix Architecture and the associated logic. With respect to the representation of continuous change, we proposed in 1989 an approach to embedding differential equations in a logic of actions and change [15, 16]. This work was later extended for representing the distinction between actual values of state variables, observed values, and set values, and for the representation of criteria for the success and failure of actions [18]. These representational methods have been included in Cognitive Robotics Logic.

Secondly, we proposed in [19] a logic characterization of goal-directed behavior, which was defined as behavior where given goals cause an agent to select one of several possible plans for achieving the goal, to attempt to execute the plan, and to try again (possibly with the same plan, possibly another one) if the first attempt fails. This is a standard aspect of rational behavior. Other aspects of rational behavior such as the use of utility measures in the choice of plans was not represented in that work, and is currently a topic for research.

4 The Double Helix Architecture

The present implementations of DHA, i.e. DOSAR and DORP, are embedded logic systems. In their overall architecture there are several specific uses of logic formulas and of deduction and other inference operations. Logic formulas occur both as a kind of data structures or messages (typically for ground literals), and as rules that are used for forward inference from observations, or for answering queries deductively.

We shall describe the DHA in two steps. The first step corresponds to a variant of a three-layer architecture where the intermediate layer is minimal; the second step corresponds to a full three-layer architecture.

4.1 The Elementary Double Helix Architecture

Figure 1 shows a simplified structure of the computational processes in the system along a time axis that runs vertically and from top to bottom. The system has a cyclic behavior where each cycle consists of a relatively long *evolution and deliberation phase* and a shorter *information exchange phase*. The 'evolution' here refers to the spontaneous developments in the physical world within which the robot or (in our case) the UAV operates, together with the automatic control processes where the robot is engaged. Concurrently with it, and independently of it, there is the 'deliberation' where the system generates and checks out plans and predicts the likely immediate future for itself and in its environment.

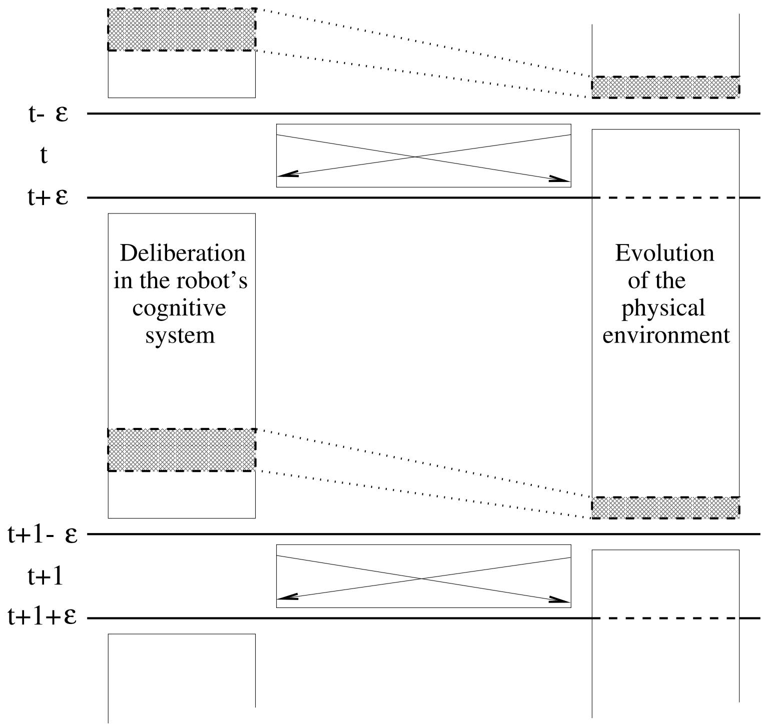

Fig. 1. *View of fhe Double Helix Architecture along the time axis. Shaded areas refer to user intervention (see section 8).*

For the purpose of developing the higher levels of the architecture, it is appropriate to replace large parts of the evolution subsystem by a simulation, provided that the simulation is a sufficiently good imitation of the target system *in those aspects that are relevant for the higher levels that are being developed.* Systematic use of simulation in this way is standard practice in concurrent engineering, but it requires of course that the relevance of the simulation is carefully monitored and evaluated.

The information exchange phase occurs each time it is appropriate to deliver sensor data, and observations based on them, to the deliberation system. At the same time, the deliberation system may update its decisions on ongoing actions or activities: helicopter maneuvers, camera movement, method for communication with the ground, etc. We expect that the information exchange phase will occur with fixed frequency most of the time, although the frequency may be reset corresponding to different flight modes and the system should be able to accomodate occasional deviations from the fixed frequency.

The data flow through this structure looks like a modified double helix: the evolution strand may lead to an observation that is transmitted to the deliberation system at time t, causing it to deliberate and to initiate an action at time $t+1$, whereby the evolution from time $t+1$ onwards is affected. The modification to the double helix occurs because both the evolution and the deliberation strand has a persistence of its own, besides receiving impulses from the other side. It is for this reason that we refer to this design as a Double Helix Architecture (figure 2).

Another and more conventional projection of the data flow is shown in figure 3, where the time dimension has been removed (it is orthogonal to the paper or screen), and one can distinctly see the cycle of data flow between the evolution line, represented to the right in the figure, and the deliberation line to the left. Atomic CRL formulas of the following kinds are used for transmitting information between the different subsystems:

- $H(t, obs(f), v)$ expressing that the value of the feature f is estimated as v based on observations at time t
- $H(t, set(f), v)$ expressing that the controllable feature f (e.g. the desired forward horizontal acceleration of the UAV) is set to v at time t by the procedure carrying out the current action selected by the deliberation system
- $D(s, t, a)$ expressing that the action a started at time s and terminated at time t
- $Dc(s, t, a)$ expressing that the ation a started at time s and is still going on at time t. This is an abbreviation for $\exists u[D(s, u, a) \wedge t \leq u]$ where the c in Dc stands for 'continuing'
- $H(t, fail(a), T)$ expressing that $D(s, t, a)$ and the action failed. (The

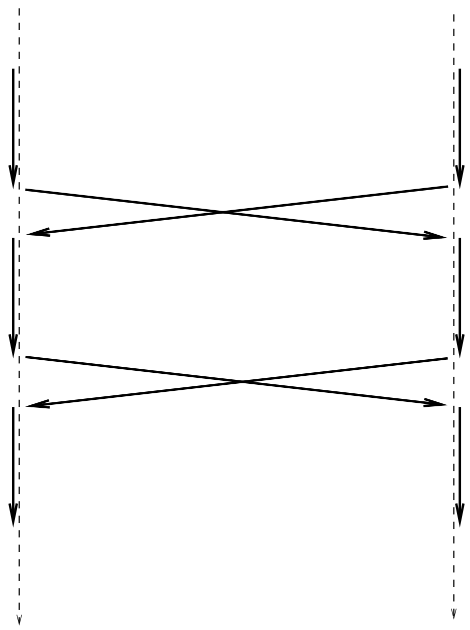

Fig. 2. *The Double Helix of the Architecture.*

execution of an action is classified in a binary way as success or failure)

These are the atomic formulas that are handled by the deliberation system and by the high-level action execution system. Formulas of the form $H(t, f, v)$ are also in the logic, and designate that the true value of the feature f at time t is v. This will of course not be directly known to the system, but the logic may make reference to such expressions in contexts such as $H(t, f, v) \wedge H(t, obs(f), v') \rightarrow abs(v - v') \leq \varepsilon$ which expresses that the observation error for the feature f is less than a known ε. In this way it is logically possible for the system to *reason about* the value of v in $H(t, f, v)$ although its exact magnitude is not known.

High-level and low-level action execution are handled differently in this design. As is shown in figure 3, there must necessarily be a number of feedback control loops that operate with a higher frequency, and whose detailed operation need not make active use of the logic. For example, one action may specify the set value for the forward acceleration of the UAV, in the course of high-level action execution. The feedback control may then use a 'controlled state variable' for the level of the throttle at each instant.

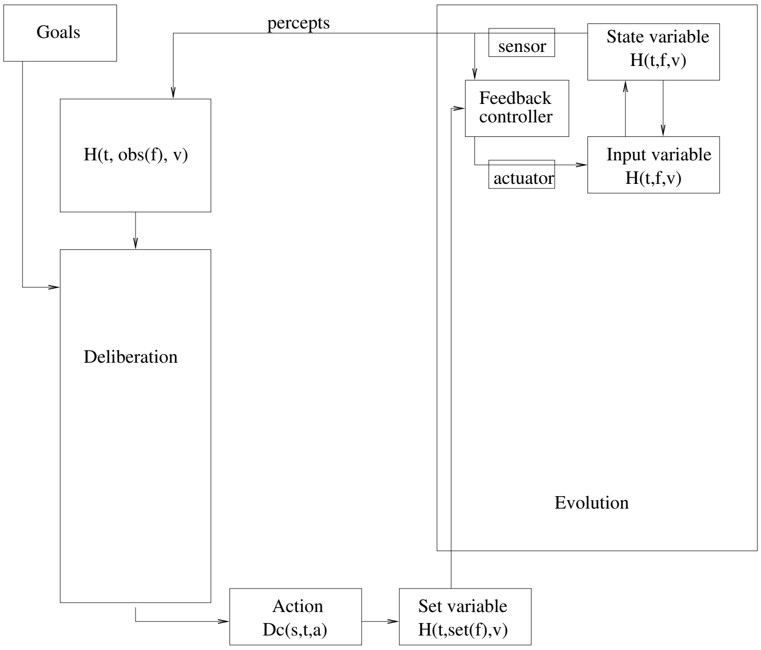

Fig. 3. *Dataflow-oriented view of the elementary Double Helix Architecture.*

The feature expressions f can be e.g. $pos(car4)$, meaning the current position of an object identified as car number 4, or $vel(car4)$ for its forward velocity, or again $acc(car4)$ for its forward acceleration. Observed instantaneous events are represented in the same way but often with boolean values. For example, an instantaneous event where $car2$ yields to $car4$ may be represented as $H(t, yield(car2, car4), T)$ where the capital T stands for reified truth-value. In such cases $H(t, obs(yield(car2, car4)), T)$ designates that the event recognizer has reported such an event; it may or may not have actually taken place. (Here we assume, for the sake of simplicity, that there is an objective definition of whether that event has taken place in terms of the actual trajectories of the cars being referenced).

The deliberation and evolution boxes in figure 3 represent ongoing activities that interact in each information exchange phase, but which also have their own continuity. Observations in the form of formulas $H(t, obs(f), v)$ do not persist intrinsically. For elementary features f they have to be calculated and reported anew in each cycle. The same applies for set-value statements of the form $H(t, set(f), v)$ which have to be recalculated in each cycle by the control procedures for the actions that are presently being carried out.

The expressions denoting actions do have persistence, however. When the deliberation system decides to initiate an action a at time s, it does so by asserting the formula $Dc(s, s+1, a)$, meaning that the action started at time s and is still executing at time $s+1$. This formula persists as $Dc(s, t, a)$ for successively incremented t, and for each timepoint t it is used to infer the appropriate values for 'its' set state variables or features.

There are several ways that an action can terminate: by a decision of its own control procedure, or by the decision of the event recognizer, or by deliberation. Also the termination may be with success or failure. In any case, however, the termination is done by adding a formula $D(s, t, a)$ saying that the action a started at time s (which was known before) but also that it ended at time t. Furthermore in the case of failure the formula $H(t, fail(a), T)$ is also added. In case of success the formula $H(t, fail(a), F)$ may be added explicitly or be obtained by default; this is an implementation consideration.

The persistence of formulas of the form $Dc(s, t, a)$ is therefore disabled by the appearance of the $D(s, t, a)$ formula. This *action persistence mechanism* in its simplest form can be characterized by the nonmonotonic formula $Dc(s, t, a) \wedge Unless\ D(s, t, a) \rightarrow Dc(s, t+1, a)$ An extended set of rules is used for also dealing with goal-directed composite actions, where the conclusion of one action with success or with failure may initiate additional actions.

4.2 Event recognition in the full Double Helix Architecture

If taken literally, the Elementary DHA would require observations to be fed from evolution to deliberation at every time-step. This is however not computationally realistic, and figure 4 shows the full architecture also containing a non-trivial intermediate layer. (Note that the usual, vertical ordering of the layers has been replaced by a horizontal one in figures 3 and 4. This facilitates comparison with figure 1, where deliberation and evolution are also placed side by side). The box labelled 'event recognizer' represents observation aggregation processes whereby a flow of detailed observations are combined into higher-level concepts, such as the arrival of an observed object into the field of observation, or the beginning and end of activities where the observed object is involved.

The term 'event recognizer' covers a fairly broad range of computations, therefore. From a propriosensoric point of view for the robot, it can

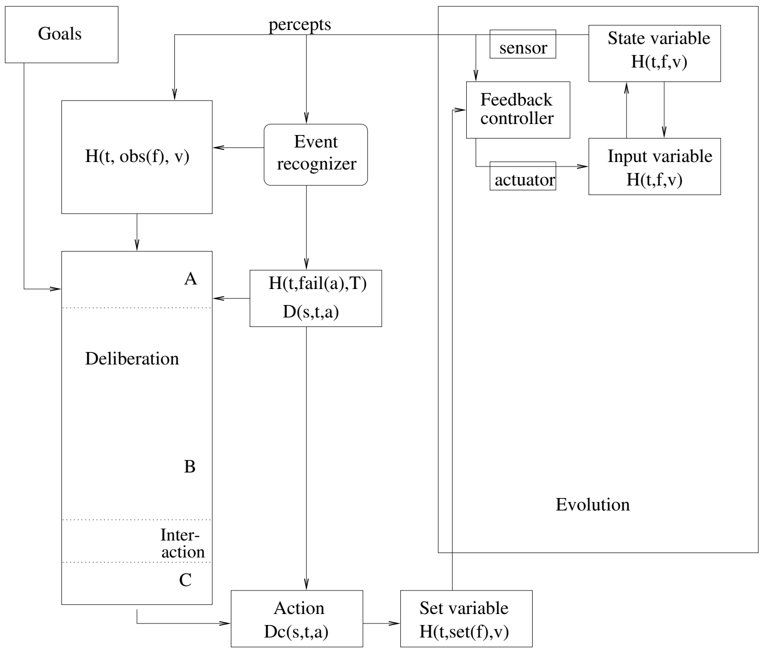

Fig. 4. *Dataflow view of the full Double Helix Architecture, including the middle layer.*

assume a fixed set of subsystems, sensors, and actuators, much like in an industrial process control system, and then the term event recognizer is accurate. For observed objects, on the other hand, the same computational process must both administrate the introduction and abandonment of data objects that represent observed physical objects, and the recognition of properties and events where these are involved.

In these respects, the 'event recognizer' is interdependent with the computer vision system. Each of them must adapt to the capability of the other one, and to the extent that they are developed independently each of them should specify what assumptions it makes about the other side. Consider, for example, a dynamic vision system that is able to segment incoming images obtaining more or less precise 'blobs', to track them reliably during certain intervals of time even when there are several of them at the same time in the field of vision, and that is also able to determine and to report whenever there is a significant risk that the presently observed blobs have been mixed up. This is a concrete and well defined situation for the event recognizer as well as for the deliberation subsystem.

The following is in outline how such a component of the architecture can be related to the cognitive robotics logic. When the event recognizer receives notice of a new 'blob' it generates a *tracker object* b and at least one *activity demon* that supervises to the tracker object. The tracker object keeps track of the continued evolution of the 'blob' as long as the underlying vision system is confident that the blob still refers to the same object. An activity demon of type p that is administrated by a perceiving agent a and that refers to a tracker object b that is considered (possibly after deliberation) to designate an actual object c, serves to identify an activity $p(a, c)$ with a particular duration $[s, t]$ that is identified by the demon while using the information collected by the tracker object. In logic terms, the occurrence of this activity is represented as $D(s, t, p(a, c))$, and this is the formula that shall be made available to the deliberation part of the system if the mapping from b to c is certain. Notice in particular that the 'events' that are recognized in this way can have extended duration, and that we are not only using instantaneous events.

The current DOSAR system contains an explorative implementation of the computational chain from tracker objects via activity demons, to activities expressed in logic, but without using any actual vision system. (As stated in section 1.2, our implementation is not part of, or connected to the current D/R software architecture used with the actual WITAS UAV platform[5]). Instead, tracker objects are generated directly from the simulator. This is sufficient as a first step, but additional work is needed in order to better understand the logical aspects of perception in relation to realistic assumptions about what information can be obtained from vision systems and other high-level sensor systems.

4.3 Action execution in the full Double Helix Architecture

A cognitive robotic system must at least provide two levels for characterizing actions and plans, namely as procedures in the programming language being used, and as formulas in the chosen logic of actions and change. The DOSAR implementation of the full Double Helix Architecture offers the following intermediate levels of description as well:

- As programmable hybrid automata. If the logic refers to one action as for example $D(s, t, flyto(\textit{building-12}))$, where s is the starting-time and t is the ending-time of the action characterized by *flyto(building-12)*, then that action may be implemented as a hybrid automaton where the state transitions are understood as mode change within the action.

- As composite actions. The third argument of the predicate D may use composite actions such as $seq(a, b)$, $try(a, b)$, and $if(p, a, b)$ where a and b are again actions, recursively, and p is a condition corresponding to and combining the second and third argument of the predicate H. Here $seq(a, b)$ executes first a and then b if a was successful, whereas $try(a, b)$ executes a and then b if a failed.
- By pursuing an agenda. An agenda contains a set of intended forthcoming actions whose ramifications have been checked out by careful deliberation. The system is able to interpret such an agenda and to execute the actions in it successively. The generation process for this agenda will be described in section 7.

There is no major difference between the expressivity in these levels of action specification as long as one only considers normal, successful execution of actions. However, the later description methods are more powerful when it comes to characterizing and controlling action failure, in order to diagnose the fault or take alternative action. The similarity of primary expressivity is then an advantage: we expect to implement facilities for automatic conversion between the action description levels in order to 'compile' and 'decompile' action scripts when needed. Because of space limitations it is unfortunately not possible to describe the hybrid automata and composite actions levels here.

5 The Logic

5.1 The use of logical inference

If the use of logical formulas in this architecture were restricted to their use as information units that are transmitted between the various computational processes, then it would be merely a choice of data structure. The use of logic becomes significant when one or more of those computational processes can be organized as deduction or other forms of inference, or if the specification of a process can be made in that way as a basis for verifying the implementation.

The deliberation process is the most obvious use of logic inference. We shall return to it in section 7, but it is not the only place for logical inference in DHA. The 'middle layer' of the full DHA can be understood as a dynamic rule-based system, where rules react to messages obtained from the two main strands of the architecture, and where also the set of rules changes dynamically. Action persistence, the management of composite actions, and the management of basic goal-directed behavior can be performed using a fixed set of rules. At the same time, event and activity recognizers as well as hybrid-automata definitions of actions are in principle situation-action rules that apply for limited periods of time. In all these cases the rules can be expressed in CRL for the purpose of uniform specification, although the implementation looks different in the interest of computational efficiency.

Action persistence was characterized by a simple nonmonotonic rule above, and that rule can be extended to also handling the successive invocation of actions in a linear script, and to the realization of goal-directed behavior. A strict application of the structure in figure 1 would imply that if an action fails then it is reported as discontinued, the corresponding set feature formulas $H(t, set(f), v)$ are no longer generated, but also the deliberation system is informed about the failure and is able to consider trying some other action towards the same goal, or even trying the same action again. However, this involves a delay, and one will often be interested in having a shorter path from the failure of one action to the initiation of a substitute action.

The formalization of goal-directed behavior in logic [19] can fill this need. That method uses a small number of axioms, operationally used for forward-inference, where the antecedents involve the kind of expressions that indicate the termination of actions in our present approach, that is, as $D(s, t, a)$ and $H(t, fail(a), T)$. These axioms, together with an appropriate plan library, serve to take the step from one action failure to the start of a substitute action without any intermediate deliberation cycle.

Furthermore, the implementation of specific actions can be done using rules of the general form $Dc(s, t, a) \wedge \neg D(s, t, a) \rightarrow H(t, set(f), v)$ with an opportunity to add additional conditions in order to control which features are to be set, and with what values.

5.2 DHA-based semantics

Almost all logics for action and change that have been proposed until now use integers, rather than a continuous domain as their concept of metric time. Surprisingly enough, however, there has been little consideration of the computational phenomena along that integer time axis. When combining the DHA and the CRL, we explicitly intend the time-steps in DHA to also be the time-steps in the integer time metric of the CRL.

The usual constructs in logics of action and change raise a number of definition issues, such as:

- Do we allow actions that take zero time, or must all actions take at least one time-step?

– Is it possible to invoke an action that is not applicable, and if so, does that lead to the action 'failing', or is it simply not semantically possible to invoke an action in a situation where it is inapplicable?
– Is it possible to have an action of a particular type from time $t1$ to time $t2$ and then another action of the same type from time $t2$ to time $t3$, or is it just one single action from $t1$ to $t3$ in such a case?

We propose that such questions ought not be resolved by vague notions of 'common sense' applied to particular scenario examples. Instead, they should be answered by defining the invocation, execution, termination, success, and failure of actions in terms of a precisely expressed agent architecture, and in particular the DHA. In doing so, one obtains a firm foundation for these concepts in themselves, and also for the subsequent project of defining high-level, discrete actions in terms of low-level, continuous or piecewise continuous phenomena.

As a topic of future research, we suggest that some aspects of symbol grounding, which is a topic of great current interest, can also be analyzed in this way.

6 The temporal situatedness of logical rules

The DHA time axis is important both for the logic and for the deliberative and other computational processes. In this section we shall argue that a precise understanding of the architecture's time axis is very useful when designing axiomatizations, deliberation systems, and function-specific software alike.

CRL predicates such as $H(t, f, v)$ and $D(s, t, a)$ treat time as some of the arguments, at a par with non-temporal arguments. From a computational point of view, however, it is natural to use the concept of a 'current state' that describes the state of the world at the 'current time'. Instead of placing all formulas in one single store and using the current time (as an integer) as a selector in that store, one may decide to collect all formulas of the form $H(t, f, v)$ and $Dc(s, t, a)$ with the same t into a heap of their own, which will be called a *state description* or simply a *state* for time t. Many parts of the computation can then be understood as performing a transformation from the state for one timepoint t, creating or amending a state for the timepoint $t + 1$. We use the terms 'old state' and 'new state' for those two states as seen from the computation. Note, however, that the considerations that follow apply as well if one uses current time as a selector into a global formula store.

The concurrent exchange that is illustrated by the two crossing arrows in the figures indicate the need for distinguishing old and new state, and not just doing assignments into one 'current' state. In this case the two states for information exchange at time t refer to the case 'just before' t and 'just after' t, respectively. This is indicated in the diagram as $t - \varepsilon$ and $t + \varepsilon$; if the information exchange is approximated as instantaneous then we are talking here of left and right limit values along the continuous time axis.

Other parts of the architecture relate in entirely different ways to the architectural time axis. For the computation in the deliberation subsystem, the 'old state' refers to $t + \varepsilon$, and the 'new state' that is constructed in the course of deliberation refers to $t + 1 - \varepsilon$.

For the computation in the evolution subsystem, finally, the distinction between 'old' and 'new' state is probably not useful at all. There will normally be one 'current state' that is updated a number of times between the discrete timepoints that are here designated as t and $t + 1$.

The cycling of the states, where what has been set up as the 'new state' becomes the 'old state', and a new 'new state' is initiated, can in principle be made either just before, or just after the information exchange phase, since those are the only natural synchronization points. It is more natural to do the cycling just before, so that the information exchange phase can be defined to construct essential parts of its 'new state' from its 'old state'. Correspondingly, if current time is represented as a single variable $currenttime$, there will be an assignment of the form $currenttime := currenttime + 1$ just before the information exchange phase. The location of this time shift is illustrated in figure 5 (time axis) and in figure 6 (cycle).

In our experience it was difficult to write the logical rules for the different processes before we were clear about the difference between the various computational subsystems with respect to their temporal situatedness. The same need for a precise understanding of information states and their changes along the time axis was even stronger when it came to writing specialized software that was to be embedded in ("plugged into") the architecture.

7 Deliberation

7.1 Robotic deliberation using natural deduction

Let us turn now to the deliberation part of DHA, which uses a variant of natural deduction. The WITAS DORP system is an early and partial implementation of the following design.

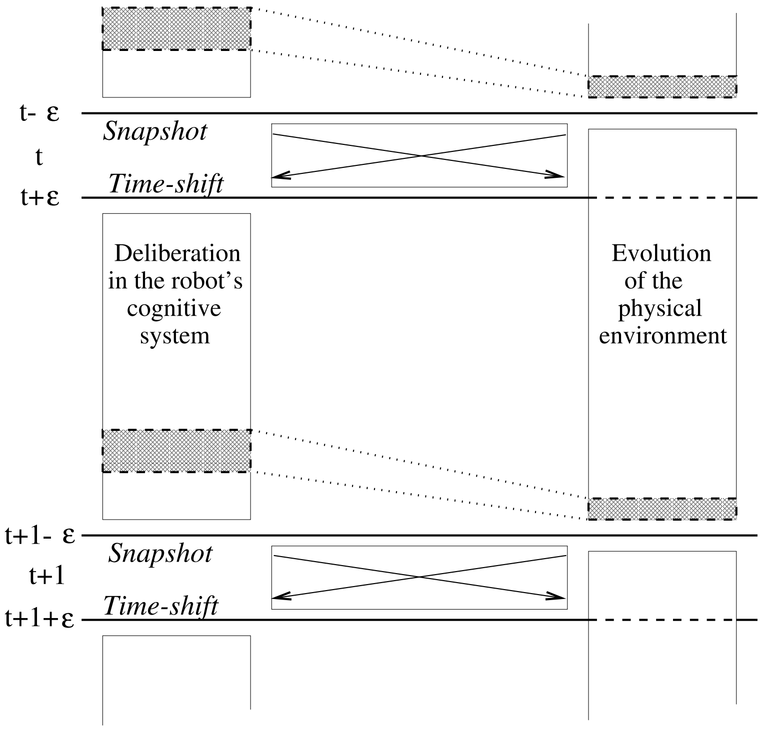

Fig. 5. *Location of Timeshift and Snapshot in the time axis view of the Double Helix Architecture.*

Natural deduction is an inference system (set of inference rules) that allows one to set up and pursue 'blocks' or 'subproofs'. Blocks can be nested recursively. Each block starts with a *hypothesis* that can be selected as any well-formed formula, and proceeds with conclusions that may rely on the hypothesis and on any previous conclusion in any outer block. A block that begins with the hypothesis H and has a formula P on a later line supports a conclusion $H \rightarrow P$ *outside* the block. Special restrictions apply for the use of variables in block hypotheses. Please see e.g. [24] for an introduction to natural deduction.

This technique is useful in the design of a cognitive robot that needs to reason predictively about its immediate future and to do progressive planning, in particular if natural deduction is modified slightly as we shall describe here. Consider for example the situation where an agent has adopted a goal g, and it considers two alternative ways of achieving the goal, characterized by the actions $seq(a, b)$ and $seq(c, d, e)$, respec-

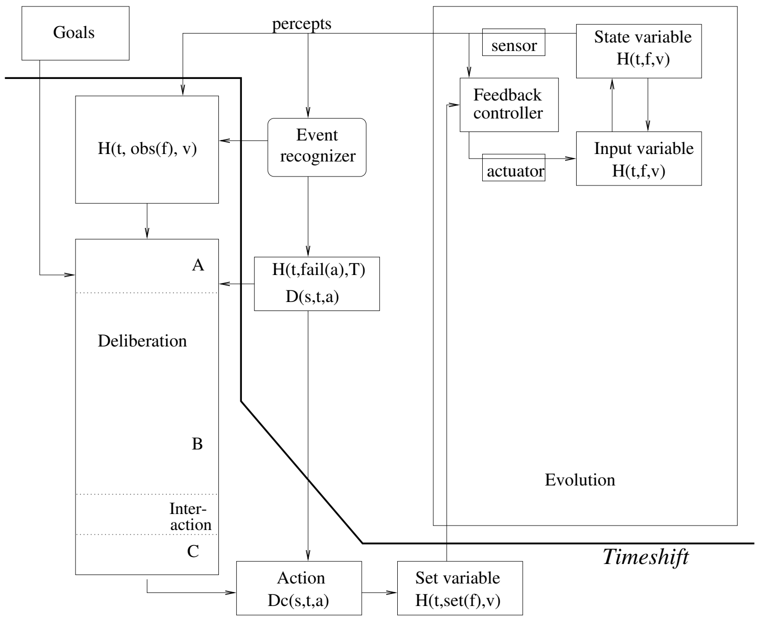

Fig. 6. *Location of Timeshift in the dataflow view of the full Double Helix Architecture.*

tively, where *seq* is the sequential composition operator for actions, as was described in section 4.3. The deliberation system will then create two natural deduction blocks, one of which uses the initial hypothesis to execute the action $seq(a, b)$ starting at some timepoint greater than the present time, whereas the other one uses the initial hypothesis of executing $seq(c, d, e)$ similarly.

The system's deliberations have the effect of (a) adding conclusions, natural deduction style, to one or both of these blocks, based on the hypotheses in question and on known facts that are available 'outside' the block, and (b) strengthening the assumptions when this is needed in order to have an adequate plan. Strengthening the assumptions can be done e.g. by choosing a particular itinerary for a move action, or a particular object or tool for the action.

These deliberations serve to identify consequences of the proposed action in the physical and operational environment at hand. They also serve to derive detailed aspects of the actions, such as how to initiate and conclude the action. For example, in a sequence of actions each of which requires movement of the robot or some of its parts, the final phase of one action may be constrained by the requirements of the initial phase of the succeeding action.

7.2 Opening of natural-deduction blocks

The formulas that are communicated in the operation of the double helix are consistently ground (i.e. variable-free) literals. The deduction in hypothetical blocks must be somewhat more general, however, since it is necessary to introduce variables for the starting-time and termination-time of actions that occur in the block-defining hypotheses. It is also sometimes necessary to introduce variables corresponding to the choices that need to be made, for example the choice of object or instrument for a particular action, and corresponding to feature values that are not yet known at the time of prediction or planning.

In all of these cases, however, the intention is that the variables that occur in a hypothetical block will later on be instantiated, either by the decision of the agent (starting time of actions, active choices), or by the arrival of information from observations (ending time of actions, in most cases, and feature-values that become known later).

In ordinary uses of natural deduction, free variables in the block hypotheses may be used for subsequent introduction of a universal quantifier on the implication that is extracted from the block, and after that for multiple instantiations of the all-quantified formula. In our case, however, we do not foresee more than one instantiation of a given block. Therefore, when an action or action-plan has been selected after having been analyzed in a natural-deduction block, one can simply 'open up' the block and declare that all propositions in the block are in fact propositions on the top level (or in the case of nested blocks, on the level immediately outside the block being opened). At the same time, the free variables in the block's hypothesis become reclassified as constant symbols. Some propositions must also be added, in particular, a proposition specifying the actual starting time of the block's action, and propositions that clarify how the invocation of later actions depends on the success or failure of earlier actions in the block.

7.3 Self-reference for natural-deduction blocks

Besides investigating consequences of particular plans on a logical level, a rational robot must also be able to compare and evaluate the effects of alternative plans. In order to make such considerations accessible to the logic formalization, we introduce a concept of *block designator* and make the following technical adjustment of the approach that was described in the previous subsection. Each new block that represents a possible course of action is assigned its own, unique designator bd_n which is a constant or function symbol. A block for a particular plan e.g. $seq(a, b, c)$ is represented by introducing a block with $Decide(bd_n, s)$ as its hypothesis, and a new formula of the form $Decide(bd_n, s) \rightarrow \exists t.D(s, t, seq(a, b, c))$ outside that block. It is clear that formulas of that kind can be freely introduced without significantly extending the set of theorems. The consequence $\exists t.D(s, t, seq(a, b, c))$ follows trivially inside the block.

The more general case where an action has additional parameters can be handled by a straightforward generalization. Parameters whose value is chosen by the agent become arguments of the block designator; parameters that emerge from the development of the world obtain an existential quantification similar to the action's ending-time t.

In this way, the block designator can be used as a handle at which one can associate utility information for the actions defining the block, as well as procedural information about the resources required for the deliberations in the block and the chances of obtaining a desired result.

8 External contributions and logs

We have already explained how different parts of the DHA system relate differently to the time axis and the breakpoints along it. In the practical system there are a number of other things that also occur along the same time axis, and that likewise benefit from a clear account of the temporal relationship between different computational tasks.

When designing and testing the deliberation subsystem, including the rules that it contains, it is convenient to have one well-defined point in the cycle where the user can intervene in order to inspect the current contents of the 'old' and the 'new' state, and also to change some of their contents. If the evolution is implemented as a simulation then one may need to intervene in a similar way in the simulation results. The shaded areas in figures 1 and 5 indicate the natural location for this interaction on the time axis.

Communication with a user via a language or multimedia interface, on the other hand, is better made as a part of the information exchange phase, and concurrently with the other exchanges there, so that the deliberative system can consider all its input at once in each cycle.

For the purpose of design and debugging, as well as for the tutorial purpose of understanding how the system works, it is also useful to produce a log of the current state of the system at some point in the cycle. We found it natural to let the log consist of such snapshots of the 'new state' just before information exchange phase, as indicated in figure 5 (time axis view) and figure 7 (dataflow view).

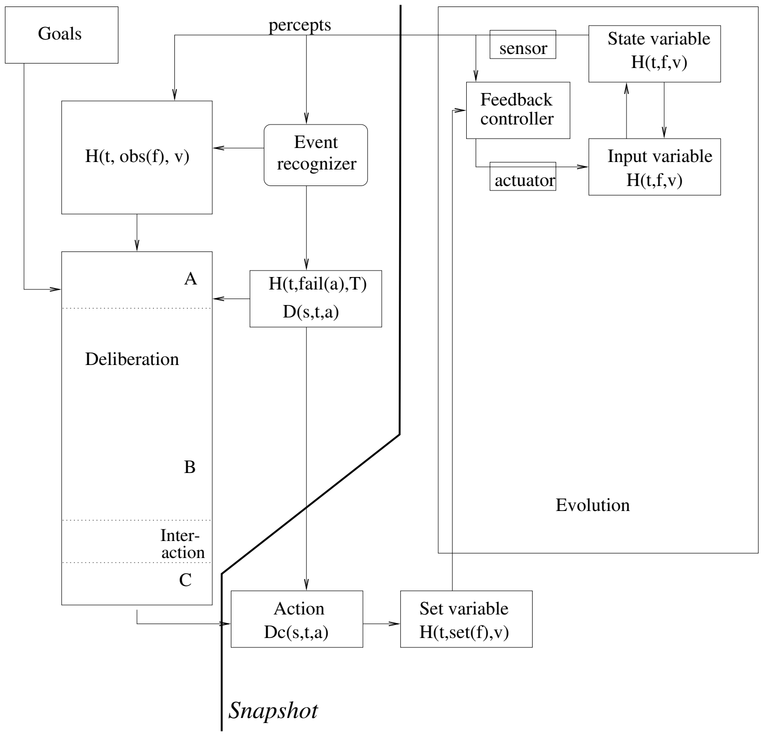

Fig. 7. *Location of Snapshot in the dataflow view of the full Double Helix Architecture.*

For detailed design and checking, it is useful to have a single-step capability where one can easily intervene at each point in (simulated) time. In an interactive and incremental programming system such as Lisp, it is natural to let the interaction (shaded segments) be done through the system's general command loop, and to define one interactively available operation that advances the system one cycle, that is, from one interaction point to the next one. This is yet another way that the DHA cycle can be split up in order to match the needs of a particular usage situation. The following sequence of operations is obtained for the single-step case, using the references A, B, and C of figures 6 and 7:

1. Deliberation, part C
2. Contribute snapshot to log
3. Information Exchange (two directions, performed concurrently)
4. Let 'new state' become 'old state', or advance timepoint counter by 1
5. Deliberation part A (interaction with operators), followed by Deliberation part B (deliberation proper); all of this concurrently with Evolution.

9 Implementation and validation

The software implementation of the Double Helix Architecture serves two very different goals. Ideally, we would like to have an implementation that can both be used as a significant component of the actual on-board system in the WITAS helicopter, *and* at the same time is such a crisp and transparent computer program that it can be formally characterized by axioms in CRL with provable consistence between axiomatization and implementation.

Neither of those goals has been achieved yet, but we try to balance the priorities so that both can eventually be reached. A crisp, analyzable software that is not used on-board, but which is reasonably isomorphic to the actually used software according to an informal analysis, would also be an acceptable final result.

The presently working DOSAR system contains a simulator where a simulated helicopter flies over simulated cars that move around in a small street network with an update frequency of 2.5 Hz. A graphics package allows to display the ongoing simulation in real time. Helicopter actions can be defined on the three levels that were described in section 4.3, and the present, small library of helicopter actions is being extended gradually. On the perception side, a few recognizers for car activities have been written. They interface to the logic level of the system in the way that was described in section 4.2.

DOSAR also has the capability to represent a catalogue of scripts that can be used as methods for achieving goals. Each script is represented as a composite action expression. On command, the system can create natural deduction blocks (subproofs) for each of the applicable methods for a given goal, and construct the corresponding elementary propositions in a way that is local to the block. On another command it can execute the actions in a given such block, which means that the 'decision' is presently left to the operator. The program parts for inference of additional consequences within a block, or in combinations of blocks are next in line for realization. The system presently relies on a library of plans, and does not at present provide for planning in the classical sense.

The decision what to include and what not to include in the present implementation was partly guided by the theoretical interests, and partly by the immediate practical use of the software. In the short term, it is to be used as the simulation counterpart for the continued development of the WITAS high-level operator dialogue facility that is being built for us at Stanford University [7]. This consideration dictated a priority on simulating car movements and arranging for logic-level observations on them, as well as the use of a plan library rather than autonomous planning, and the choice of a quite crude simulator for helicopter movements.

Later on, it is intended to use DOSAR as the bridge between the dialogue and on-board systems, at which point the simulation part will be removed and replaced by an interface to the on-board system. (This interface will also need to include a safety-switch under operator control). Different approaches to deliberation are presently being pursued in the on-board system for WITAS as well as in the DHA, and it remains to be determined how these will compete with, or complement each other.

On the theoretical side, our strategy is to iterate on the software design in order to gradually make it more crisp, and thus to move in the direction of an implementation that can be validated formally.

The DORP software is a separately maintained system variant where deliberation and agenda management facilities are checked out with an alternative set of actions. This testbench is particularly useful for verifying the consistency of the timing when actions are invoked, scripts are executed, and events and activities are recognized along a common timeline. Facilities that have been validated in DORP as working correctly are routinely transferred to DOSAR.

Both DOSAR and DORP have been implemented in CommonLisp in order to prepare the ground for the theoretical arm of the work. They are running in standard PC environments. Achieving the required real-time performance has never been a limiting factor in this work. The Software Individuals Architecture [21] is used as the lowest Lisp software layer, providing a systematic way of structuring software modules, data directories, command sessions, and software maintenance and exchange.

10 Conclusion

We have described the mutual relation between the Double Helix Architecture and the Cognitive Robotics Logic. Our main message is that this agent architecture, which explicitly describes the location of different computational processes and layers along a time axis with discrete steps, is useful for clarifying both the organization of the computation, the semantics of the logic, and the interdependence between those two issues.

11 Acknowledgements

The WITAS project is funded by a generous grant from the Knut and Alice Wallenberg Foundation.

References

1. Raja Chatila and Jean-Paul Laumond. Position referencing and consistent world modelling on mobile robots. In *IEEE International Conference of Robotics and Automation*, pages 138–145, 1985.
2. Thomas Dean and Michael P. Wellman. *Planning and Control*. Morgan-Kaufmann, 1991.
3. Patrick Doherty, Gösta Granlund, Krzystof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The witas unmanned aerial vehicle project. In *European Conference on Artificial Intelligence*, pages 747–755, 2000.
4. Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. Temporal action logics language. specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 2:273–306, 1998.
5. Patrick Doherty, Tommy Persson, Björn Wingman, Patrick Haslum, and Fredrik Heintz. A CORBA-based deliberative/reactive architecture for unmanned aerial vehicles. Unpublished manuscript., 2002.
6. R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain. An architecture for vision and action. In *International Joint Conference on Artificial Intelligence*, pages 72–79, 1995.
7. Oliver Lemon, Alexander Gruenstein, and Stanley Peters. Collaborative activities and multi-tasking in dialogue systems. *Traitement Automatique des Langues, special issue on dialogue*, 2002. To appear.
8. Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2:159–178, 1998.
9. Hector J. Levesque, Raymond Reiter, Yves Lesérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, April-June 1997.
10. Jörg Müller. Control architectures for autonomous and interacting agents. A survey. In Lawrence Cavedon, Anand Rao, and Wayne Wobcke, editors, *Intelligent Agent Systems*, pages 1–26, 1996.
11. Jörg Müller. *The Design of Intelligent Agents*. Springer Verlag, 1996.
12. Nils Nilsson. Teleo-reactive programs and the triple tower architecture. *Electronic Transactions on Artificial Intelligence*, 5B:99–110, 2001.
13. Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *International Conference on Knowledge Representation and Reasoning*, pages 439–449, 1992.

14. Stuart Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach.* Prentice-Hall, 1995.
15. Erik Sandewall. Combining logic and differential equations for describing real-world systems. In *Proc. International Conference on Knowledge Representation, Toronto, Canada*, 1989.
16. Erik Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In *International Joint Conference on Artificial Intelligence*, pages 894–899, 1989.
17. Erik Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press, 1994.
18. Erik Sandewall. Towards the validation of high-level action descriptions from their low-level definitions. *Artificial Intelligence Communications*, December 1996. Also Linköping University Electronic Press, http://www.ep.liu.se/cis/1996/004/.
19. Erik Sandewall. A logic-based characterization of goal-directed behavior. *Electronic Transactions on Artificial Intelligence*, 1:105–128, 1997.
20. Erik Sandewall. Cognitive robotics logic and its metatheory: Features and fluents revisited. *Electronic Transactions on Artificial Intelligence*, 2:307–329, 1998.
21. Erik Sandewall. On the design of software individuals. *Electronic Transactions on Artificial Intelligence*, 5B:143–160, 2001.
22. George N. Saridis and Kimon P. Valavanis. On the theory of intelligent controls. In *Proc. of the SPIE Conference on Advances in Intelligent Robotic Systems*, pages 488–495, 1987.
23. Murray Shanahan. A logical account of the common sense informatic situation for a mobile robot. *Electronic Transactions on Artificial Intelligence*, 2:69–104, 1998.
24. D. van Dalen. *Logic and Structure.* Springer Verlag, 1980.

This article was processed using the LaTeX macro package with LLNCS style