Learning and Performing Hard Real-Time Skills*

Jürgen Dorn

Technical University of Vienna, Christian Doppler Laboratory for Expert Systems

Abstract
An architecture for hard real-time applications is proposed that enables the user to care safety requirements and time restricted search explicitly. This is supported by an intelligent event-oriented interface to control the technical process. The knowledge-based system “programs” this interface. It stores always safe plans. If the knowledge-based system will eventually find a save and better solution for an existing problem, the program of the interface will be modified.

On this architecture a system was implemented that learns skills. Skills are complex compound actions where the temporal and causal coordination of the different actions is important. The system was used to control a hard real-time experiment.

1. INTRODUCTION
Real-time problem solving is one of the most difficult problem areas for knowledge-based systems. The usefulness and correctness of a real-time system relies not only on logical results of computation, but also on the time at which results are produced. Therefore in real-time applications knowledge-based techniques are still not very common.

It is argued that real-time requirements are hardly to be met by these techniques, because exhaustive search algorithms are applied and therefore the time for search is hardly predictable. Safety is the most important requirement for real-time applications. If a control system cannot guarantee a safe reaction in time it is worthless. Therefore usually no closed loop between technical process and knowledge-based system is established.

No hard time constraints can be violated, because no reactions are performed. Only an interpretation of sensor information is done. It is important that such a system react fast, but there is no absolute deadline. Such systems may support an user of a real-time system in several ways:

• The system supervises operations in an application and gives the user hints when and where he should maintain a machine in the application.
• If a process has broken down, it is diagnosed what has caused the break-down and perhaps a therapy is advised. The real-time system has to assure the safety and the knowledge-based system supports the operator.
• A knowledge-based system can be used as a redundant component of a fault-tolerant system to enhance the safety of the whole system. It will only be used for voting, but not for direct control.

Whether a knowledge-based system may be coupled also in the other direction is discussed heavily. The problem is that a control system must be save to prevent mortal danger or other damages.

One of the most important requirements that follows out of the safety requirement is the ability to react in time [1]. This stands in contrast to exhaustive search. There are ideas to overcome this problem. One approach in this direction is the search algorithm RTA* [2]. But this algorithm still lacks the possibility to reason explicitly about time. The search is constrained with a fixed time that is not domain dependent.

Another problem is the explicit reasoning about safety. Knowledge-based techniques reclaim the possibility to reason about uncertainty. Therefore, it is argued by some researchers, the results of knowledge-based systems are uncertain and no useful technique for real-time applications that rely on safe reactions.

Our opinion is that in knowledge-based systems it clearly must be distinguished between certain and uncertain knowledge and between actions that are relevant for the safety of a system and those that are not. Further a system has to argue about stable and unstable states. If an action shall be performed that results in an unstable state, before performing this action, first an action must be found that would transform the unstable in a stable state.

One approach to argue about safety is described in [3]. A hierarchic architecture for an autonomous robot at SRI is presented. The main point is the layered structure of competence for decisions. The lowest layer only reacts on single raw sensor data. Upper levels react on combinations of sensor data and additional knowledge about the environment. A level acts only if it knows that the action would be save. Levels may react independently of each other. If two or more levels react simultaneously, the highest level will have precedence. The planner operates in time slices which are called ticks. So reactions of the system are guaranteed within a tick. It is assumed, that the lowest level will always find a save reaction.

This approach assumes that real-time applications demand a fast treatment of events, but no explicit reasoning about remaining time is possible.

We propose a two-layered architecture with an intelligent event-oriented interface between knowledge-based system and technical process. The interface is responsible for timely reactions and the knowledge-based system for better solutions. The general procedure is as follows:
1. an event occurs in the application
2. a safe reaction is already programmed in the interface, but it will be delayed until timeliness demands it
3. the system searches for a better plan
4. if a better and safe plan is found in time, the safe action is replaced
5. the system searches further for a still better plan

With the event-oriented interface human reflex actions are modeled. These actions are performed without “thinking” and they will change over several experiences. New reactions have to be learned and some old reactions are forgotten. These reactions could also be complex actions build up by simple actions. The process of learning these actions that we call skills will be the main point in the further discussion.

In the next section the basic representation concept and the functionality of the event-oriented interface is described. Then the set-up is described that is used to illustrate the new concepts. In the fourth section the learning and performing of hard real-time skills is studied in detail.

2. THE EVENT-ORIENTED APPROACH

The basic language – an interpretation of first order logic – is supported by the event-oriented interface which is described later.

In AI a so called state-space representation or situation calculus is common. Knowledge about the application is described by states. If something is changing this yields in a new state or situation. This is computationally inefficient and not adequate for technical representations where changing is one of the most important concepts. Furthermore, parallel activities are quite difficult to represent in the situation calculus.

Our knowledge representation technique is based on temporal intervals and is called event calculus. Propositions about an application are constrained to be valid only in an interval. The “@”-operator is used to formulate temporal propositions. So the validity of a proposition is restricted to an interval:

\[
\text{temporal-proposition } @ \text{interval.}
\]

Different types of propositions are distinguished in [4] but they have no relevance for the further discussion.

2.1. Temporal Intervals

Granulation intervals are the smallest measurable time period. Using a clock to measure time in an application one gets a discrete quantity. After every increment of this quantity the time of one granulation interval has been elapsed. In many technical applications the granulation (or tick) will be one millisecond. Granulation intervals are used to define temporal intervals with attributes. Begin, end, and duration are granulation intervals of an interval.
In technical applications it must be accepted that the real interval in the application will deviate from the interval in the representation. The largest deviation of an interval in the representation from the real interval will be one granulation at both sides of an interval.

We model granulation intervals with a time axis that is built up like the natural numbers. The difference to natural numbers is only the interpretation as intervals. Infinity ($\infty$) is an element of the time axis. Addition, subtraction, and ordering relations are defined on granulation intervals.

Attributes of an interval maybe only constrained to a range, because they are often uncertain, incomplete, or unknown. A time bound describes a range of possible granulation intervals. It is defined by a pair $G_1 .. G_2$. During subsequent knowledge processing the attributes may be constrained stronger. An attribute is unconstrained if the upper time bound is the infinity: $0 .. \infty$. If an exact value is known, the upper and the lower bound are equal: $G_1 .. G_1$.

Addition, subtraction, and the ordering of time bounds are derived from the corresponding definitions of the granulation intervals. The intersection of two time bounds is the set of all granulation intervals that are in both time bounds. Intersections of time bounds are used to generate stronger constraints on absolute time specifications.

From begin and end of an interval the duration can be deduced. The specification of a third attribute is always redundant. But we suppose that interval specifications are often incomplete. Perhaps it is known, how long a process continues, but not when it will begin or end. If the process begins actually, the end can be computed. The three attributes of an interval are constrained by: Begin + Duration = End.

![Figure 1. Representation of Intervals](image)

Constraints between intervals are defined in order to allow qualitative reasoning about time. A set of intervals and interval constraints are interpreted as an interval graph. Through transitive constraints new knowledge is generated. With this propagation technique also consistency checking is done.

The quantitative and qualitative representation of time and the improved propagation of constraints in interval graphs is supported by TimEx [5]. In the following discussion we will use the well known abbreviations for interval constraints used by Allen [6].

- 4 -
2.2. The Event-Oriented Interface

A declarative representation relies on a search through a problem-space. The time needed for search is hardly predictable. The advantages of flexibility and maintainability in a declarative representation yield in this disadvantage. With the proposed architecture we try to reduce this disadvantage by means of the event-oriented interface. This part of an integrated real-time system supports time critical procedures and guarantees a fixed response time.

The event-oriented interface connects the technical process with the knowledge-based system. It buffers information and passes it through on demand. A prototype of the event-oriented interface was implemented on a RT-VAX 100 under ELN operating system. The software is written in ELN-PASCAL and the event-oriented interface is connected to the knowledge-based system via ETHERNET. The knowledge-based system was implemented in PROLOG and has used scripts [4] as a structured representation mechanism.

The event-oriented interface is domain independent. In the interface no knowledge about any domain is used. The code that a sensor will supply to the interface will not be interpreted. Only a temporal interval is added to this information. The code for an effector in the technical process is only passed through. The interface will delay this information for a time that is specified in the interval that will be supplied together with the code for the actor.

![Diagram of the Event-Oriented Interface](image)

**Figure 2. The Event-Oriented Interface**

The event-oriented interface consists of two memory parts. One is used for tasks that shall be performed and the other for sensor values. To synchronize the interface with the environment it has access to a real-time clock.

The main part of the interface is a scheduler that schedules the tasks in the memory in accordance with the specified intervals. If the memory is restricted in the number of tasks a guarantee for a response time can be given.

There may be several interfaces from the event-oriented interface to the technical process. These are addressed by numbers.

The knowledge-based system sends messages to the event-oriented interface in order to initiate a task or to get measured values.

The message to initiate a simple task consists of a code, a port address and an interval. The interface supplies the code during the given interval to the port. Arguments that must be instantiated are underlined in the following notation. Intervals may be instantiated partly. Not instantiated arguments may be instantiated by performing a task.
simple_task(ID, Code, Port, Interval)

Tasks may be changed or removed from the event-oriented interface by another message.

Sensors often have to be polled regularly to supervise a process. It would be fruitless for the knowledge-based system to generate for every polling a task. Therefore a cyclic task may be generated. The following message specifies a cyclic polling with a specified delay. The delay is given as a number of ticks.

cyclic_task(ID, Delay, Code, Port, Interval)

Sensor values will be stored in a memory. If a new value is delivered by the sensor it will be stored. If it has changed significantly, a new interval will be created. Otherwise only the interval of the old value is extended. The following figure illustrates this:

Figure 3. Representation of Measured Sensor Values

The knowledge-based system uses a query procedure to obtain measured values. The supplied interval must be instantiated partly.

query(ID, Code, Interval)

Often an event in the technical process needs a fast reaction. Therefore a conditional action is defined. It has some similarities with a production rule. But conditional actions can not be chained like production rules in a rule-based system. A further difference is that left-hand-side and right-hand-side are enriched by intervals stating when the condition shall hold. A delay between the occurrence of the condition and the action or an interval constraint between the two intervals may be specified optional.

conditional_task(ID, Event, Port1, Interval1, Delay, Action, Port2, Interval2, Constraint)

What still is needed is a procedure to initialize the interface and to synchronize time in the knowledge-based system and the event-oriented interface.
3 THE EXPERIMENT

The “Catapult” is an experimental set-up that was designed to have a technical process with hard real-time constraints. These constraints cannot be met by processors running under a standard real-time operating system.

![Diagram of the Catapult](image)

**Figure 4. The Catapult**

An iron bolt (50 mm long) is free to move up and down inside a vertical tube. A magnetic force can be exerted on the bolt by applying an electrical impulse to an coil surrounding the tube. This force will cause the bolt to be accelerated vertically. With a second coil the bolt may be accelerated further.

Two light-gates mounted directly above the coils are used to sense the position and velocity of the bolt at two distinct points. The coils and the light-gates are connected to the event-oriented interface via a parallel interface.

The first goal of the experiment is to optimize the “height-of-flight” of the bolt by varying the duration of the electrical impulse on the first coil between 15 and 30 ms in a series of tests.

The bolt may be accelerated even further by activating the upper coil at the right time. If the sequence of both impulses has been chosen correctly, the bolt will hit the upper end of the tube before falling back and hitting the base-plate.

Most difficult is the third step of the experiment: “catching” the falling bolt by switching on the lower coil just before the landing. If done properly, the bolt will land softly.

The switching times must be determined experimentally due to many nonlinearities in the model. Also a series of tests is needed, because often runaways occur due to differing friction between tube and bolt. Nevertheless the last switching time to catch the bolt can be computed in dependence of the upper light-gate if the temporal relation was learned in several tests.
4. LEARNING SKILLS

To guarantee fixed response times it is necessary to restrict the memory of the event-oriented interface. Therefore often the event-oriented interface will be modified by the knowledge-based system. The usual operation of the interface will be that the knowledge-based system recognizes a specific situation or uses a specific script. A script is a declarative representation of a complex action consisting of several single actions and events [4]. The knowledge-based system knows that specific errors or other asynchronous events may occur in the situation or during the performance of the script. For these errors specific treatments may be necessary. Therefore the knowledge-based system loads some conditional tasks down into the interface. If the situation changes or the script is finished some or all tasks will be deleted.

4.1. Reflex Actions and Skills

With the event-oriented interface human reflex actions are modeled. On an external stimulus a response is generated. This is done by the conditional action. The response can be delayed. Typically these responses will be actions that are performed due to safety reasons. This is also the cause for most human reflex actions.

But these reactions are not always the best reaction. If the environment of man or machine changes over some period of time, these reactions may become senseless or even dangerous. The senseless reactions may be forgotten over time. The dangerous reactions should be overwritten by better reactions due to bad experiences or through intelligent reasoning.

Reflex actions are of very simple structure. Often more complex actions are needed. These actions can be divided into several basic actions. These must be performed in the right order. If this order is gained through much experience, we speak of skills. Different people and different machines can achieve different skills. Skills are dependent on resources and experience.

A typical human skill is performing a smash in tennis. Some people do it better than other. We can give theoretical hints to improve the skill, but this alone will not help. We know that it will be better to hit the ball at a higher point, but the problem for a player is to translate this into a movement. So every player can learn the theoretical concepts, but there will still be differences between several of them. This is also caused by physical condition, but most important are experience and coordination of several aspects of the movement for a good performance.

In technical processes these skills will be very important. For an autonomous mobile robot that is equipped with compliant sensors it will be necessary to learn such skills. If a new screw joint will be used in a factory, the compliant sensor will help the robot to establish this new joint. The robot will use causal knowledge and a kind of trial-and-error method. But it will not make much sense if it must try it always new. The robot should learn this skill. The first trial will not be sufficient, because some aspects of the movement will change over different trials. But with some experience the robot should be capable to perform this skill without errors.
4.2. Example

As an example for learning such a skill we have taken the described real-time experience. A robot was not taken, because no intelligent sensor was available. Furthermore, the intention was to show that the proposed techniques are applicable in hard real-time applications. In this application it is clearly to be seen that with other knowledge-based techniques the problem would not be solvable.

We assume we have a preplanned set of actions - a script - that should be performed to solve the problem. This plan could result from a cased-based planner like that described in [7].

The plan consists of actions restricted to temporal intervals. The flight of the bolt is supervised with the two light-gates. Such a sensor information is an event constrained by an interval. The interval is constrained against other intervals by means of interval constraints.

\[
\begin{align*}
\text{apply-magnetic-force(} & \text{lower-coil)} \ & @ i_1 \\
\text{vertical-acceleration(} & \text{bolt)} \ & @ i_2 \\
\text{lower-lightgate} \ & @ i_3 \\
\text{apply-magnetic-force(} & \text{upper-coil)} \ & @ i_4 \\
\text{upper-lightgate} \ & @ i_5 \\
\text{falling(bolt)} \ & @ i_6 \\
\text{upper-lightgate} \ & @ i_7 \\
\text{lower-lightgate} \ & @ i_8 \\
\text{apply-magnetic-force(} & \text{lower-coil)} \ & @ i_9 \\
\text{user-ok} \ & @ i_{10}
\end{align*}
\]

\[
i_1 \ s \ i_2 \ w \ i_1 \ < \ i_3 \ < \ i_5 \ w \ i_4 \ < \ i_5 \ w \ i_2 \ m \ i_6 \ w \\
i_6 \ < \ i_7 \ w \ i_7 \ < \ i_8 \ w \ i_9 \ < \ i_{10} \ w \ i_{10} \ d \ i_2 \ w \ i_9 \ < \ i_6
\]

Some further interval constraints may be concluded from the stated. But about the temporal relation between \(i_1\) and \(i_4\) and between \(i_8\) and \(i_9\) nothing is known. The correct relation must be learned through experience. Also the durations of the different intervals are not known yet. But through additional constraints the durations are restricted. We assume that the duration of the two movements will range between 100 and 750 ms. It is assumed that the duration of the signal of the light-gates will somewhere between 5 and 40 ms. This is estimated by the heights of the set-up. The granularity is 1 ms and for the following illustration the experience will start at granulation interval 0. Estimating the durations, using the interval relations, and starting the experience at 0, the following values are result of a propagation technique described in [5].

\[
\begin{align*}
i_1, & \ 0 \ .. \ 0, \ 15 \ .. \ 30, \ 15 \ .. \ 30 \\
i_2, & \ 0 \ .. \ 0, \ 0 \ .. \ 800, \ 100 \ .. \ 750 \\
i_3, & \ 1 \ .. \ 728, \ 6 \ .. \ 743, \ 5 \ .. \ 40 \\
i_4, & \ 0 \ .. \ 1485, \ 15 \ .. \ 1500, \ 15 \ .. \ 500 \\
i_5, & \ 7 \ .. \ 744, \ 12 \ .. \ 749, \ 5 \ .. \ 40 \\
i_6, & \ 100 \ .. \ 750, \ 200 \ .. \ 1500, \ 100 \ .. \ 750 \\
i_7, & \ 101 \ .. \ 1488, \ 106 \ .. \ 1493, \ 5 \ .. \ 40 \\
i_8, & \ 107 \ .. \ 1494, \ 112 \ .. \ 1499, \ 5 \ .. \ 40 \\
i_9, & \ 0 \ .. \ 1485, \ 15 \ .. \ 1500, \ 15 \ .. \ 500 \\
i_{10}, & \ 200 \ .. \ \infty, \ 201 \ .. \ \infty, \ 1 \ .. \ 1
\end{align*}
\]
The knowledge-based system has to optimize several goals. Causal explanations of the case-based planner are used to decide which parameters should be varied. Causal relations are given by an implication.

The first goal the knowledge-based system has to achieve is the following:

\[ \text{goal}(\text{minimize}(\text{begin}(i_3)) \Rightarrow \text{duration}(i_1)) \]

The causal relation is the output of the case-based planner. The system will perform several trials to minimize the begin of interval \( i_3 \). This results in a high acceleration of the bolt. After having found an optimum it tries to optimize the second goal resulting in a optimal height-of-flight:

\[ \text{goal}(\text{minimize}(\text{begin}(i_5) - \text{end}(i_3)) \Rightarrow (\text{duration}(i_1) \land \text{duration}(i_4))) \]

This optimum is dependent on two parameters. One was found to be optimal for the first goal, but it can be changed again for the new goal. The third and last goal is to achieve the user-ok. This user-ok is dependent on \( i_9 \).

\[ \text{goal}(\text{user-ok}) \Rightarrow (\text{begin}(i_9) \land \text{duration}(i_9)) \]

The knowledge-based system estimates values and sends all actions for one trial to the event-oriented interface. The estimation is done by choosing one value of the allowed values.

The action for catching the bolt is a conditional action, because we want to have it in a fixed temporal distance after getting the signal from the upper light-gate. The messages sent to the event-oriented interface are the following. We have taken the values used in the last trial of our knowledge-based system. So all time bounds are restricted to single values.

\[
\begin{align*}
\text{simple_task}(t_1, \text{power_on}, 1, (0..0, 16..16, 16..16)) \\
\text{simple_task}(t_2, \text{power_on}, 2, (16..16, 186..186, 170..170)) \\
\text{conditional_task}(t_3, \text{light_gate}, 4, (X_1..X_1, X_2..X_2, X_3..X_3), 5, \text{power_on}, 1, (X_5..X_5, X_6..X_6, 200..200), <)
\end{align*}
\]

The result is a program with two tasks with fixed intervals. The third is a conditional action that depends on the event “light_gate”. When signal “light_gate” is given on port 4, after a delay of 5 ticks a “power_on” is supplied for 200 ms on port 1.

5. CONCLUSIONS

The event-oriented interface was developed to achieve an architecture for a knowledge-based system that can argue about real-time constraints and safety. In this framework reactions with a fixed response time can be guaranteed. On this foundation programs that care safety requirements are possible.

We have depicted a knowledge-based technique for complex real-time systems that uses the event-oriented interface as a kind of short-time memory. Depen-
dent on the actual situation the knowledge-based system programs the event-oriented interface. To make this effective we have introduced an approach to learn skills that have hard real-time constraints.

The presented experience was intended partly to show the capabilities of the event-oriented interface and partly to show how one aspect of machine learning for hard real-time tasks may be done. Learning is a very complex field in AI and we can not give here a general solution. But we feel that we have given a valuable contribution to one aspect of machine learning.

The event-oriented interface has proved to be mostly adequate for such a kind of real-time application. One additional feature was proposed in [8] as a consequence. Because the temporal variance in the set-up is very large, also the duration between getting the signal from the upper light-gate and "catching" the bolt varies. But it was identified that this duration was functionally dependent on other times. So the new proposal is an action for that the interval attributes will be computed by a function. This will be given to the event-oriented interface together with the action. Now both light-gates could be used to compute the velocity of the falling bolt. This can be used to compute the time for applying the lower coil.

In the example we have shown the capabilities of the event-oriented interface and the principle of learning a skill. The appropriateness of the architecture depends on the complexity of the application. For such a small application like the "catapult" the described technique is exaggerated. But the intended application are mobile robots or other applications with changing environments.

In these applications a case-based planner would be meaningful. The case-based planning concept assumes that old cases (scripts) are used for a new problem. With every case a set of goals that this case will achieve is stored. By means of the goals the planner must choose between several old cases. In the chosen script some aspects must be adapted to the actual problem.

We have only used one "artificial" old case. So the usefulness of this aspect could not be shown here, but we believe that the taken approach is reasonable for complex real-time applications.
6. REFERENCES

1. Thomas J. Laffey, Preston A. Cox, James L. Schmidt, Simon M. Kao, Jackson Y. Read.


4. Jürgen Dorn.

5. Jürgen Dorn.
   “TimEx – A Tool for Interval-Based Representation for Technical Applications”.


8. Frank Splettstößer.