LiTH-IDA-R-88-24 ISSN-0281-4250

INSTITUTIONEN FÖR DATAVETENSKAP
UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

Realizing Action Plans and Response Rules in a System Tool for an Autonomous Vehicle

by

Johan Hultman

Anders Nyberg

INDUSTRISERIEN
RKLLAB, August 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

LiTH-IDA-R-88-24 ISSN-0281-4250

Realizing Action Plans and Response Rules in a System Tool for an Autonomous Vehicle

by

Johan Hultman

Anders Nyberg

Abstract: This report describes the relationships between the various levels in a system tool for an autonomous vehicle. The tool referred to in this report is described in two earlier reports. The system tool consists of three levels of layers, each one having a representation of the action plan the vehicle is to carry out. We suggest how tasks and action plans can be represented in the various levels of the system tool. Furthermore we discuss how unforeseen events can be dealt with at the various levels.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

Realizing action plans and response rules in a system tool for an autonomous vehicle

1. Introduction.

In this report we describe the relationships between the various levels in a system tool for an autonomous vehicle. The tool referred to in this report is described in [Sandewall 88-1] and [Sandewall 88-2]. We suggest how tasks and action plans can be represented in the various levels of the system tool. Furthermore we discuss how unforeseen events can be dealt with at the various levels.

2. Action planning.

In previous papers we have exemplified how a system operator can define tasks to be carried out by an autonomous vehicle [SUTEC 88]. The vehicle must be able to translate such tasks to some form of action plan which specifies which actions are to be carried out at a certain time. This transformation is known as action planning.

An action plan consists of a set of partially ordered actions, ie. actions that may be executed both sequentially and in parallel. Parallel execution of actions puts heavier demands on the planning system than when actions can only be executed sequentially. The planning system must also take into consideration which resources the actions use. Can two actions which employ the same resource be executed in parallel? Do two actions have to be executed in parallel? This section deals with such questions.

The system tool is described in [Sandewall 88-2] and consists of three levels or layers. Each layer must have a representation of the action plan the vehicle is to carry out. Figure 1 shows the relationship between the various levels of the system. The operator specifies the task for the vehicle by means of a graphic interface as described in chapter 1 of [SUTEC 88]. This task is then translated into a textual representation in a task specification language. It is also possible to define the task directly in the task specification language. The task specified is then translated to an action plan in the analysis layer and this action plan is in turn translated into an action plan in the rule layer.

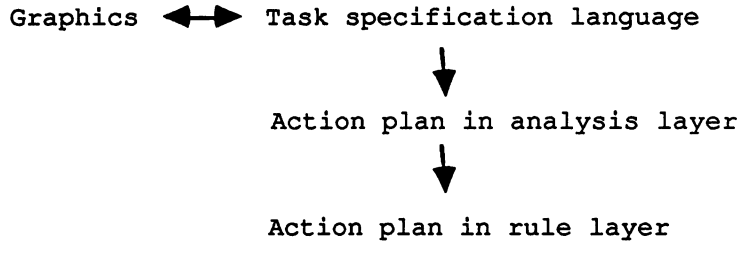

Figure 1. The various levels of representation in the system tool

In the section below we describe how tasks and action plans can be represented in the system tool and how the translation between the various levels is achieved.

2.1. Task representation.

"Three scenarios for an AROV" in [SUTEC 88] contains a description of how a user interface for defining tasks can be built. The graphical representation described is highly suitable for a human operator, but is not particularly appropriate as input data to a planning system. Consequently the graphical representation must be translated to a textual one that the planning system can use and process. In this section we sketch the outlines of such a representation. The actual translation from the graphical representation to the textual is not described here as it is not expected to involve any difficulty.

The representation we have chosen is not the most general, ie. it is not possible to express all types of action plans in it. The limitations are, however, fairly minor and we do not believe that they can influence which tasks the vehicle can carry out.[1] The representation given below has been chosen for its ease of understanding and use, an advantage in the absence of the graphical interface. Coding the task in the given representation then becomes possible.

2.1.1. Task specification language.

A requirement on the textual representation of tasks is that it can cope with the representation of all the tasks an autonomous vehicle can be expected to carry out. This implies, for instance, that we must be able to represent sensors, actuators, navigation- and perception-actions. Defining events to be carried out in a certain sequence or in parallel must also be possible. To handle this, the task representation must be more or less a programming language, even if it is specially designed for tasks to be executed by an autonomous vehicle. For example, the language must allow for the definition of variables, loops and parallel actions (see further below).

Assume the vehicle is to be sent on an assignment to search a given area. This type of task may be represented as follows:

```
(ACTIVATE SONAR)
(GO (POS x))
(ACTIVATE CAMERA)
(SEARCH (AREA y z)(PATTERN ZIGZAG))
(DEACTIVATE SONAR)
(GO (POS HOME))
(DEACTIVATE CAMERA)
```

The representation is similar to a program consisting of a number of statements to be executed sequentially. These statements can be divided into two groups, those that define navigation actions and those that define perception actions. Start and end points are given for perception actions, which makes it possible to express the parallel execution of several perception actions. Navigation actions are given as atomic statements, which means that we can only express sequential execution of such actions (see also 2.1.5).

Certain of the navigation actions given in the assignment are not primitive, ie. they can be broken down into one or several other navigation actions. In the example above, SEARCH, for example, can be broken down into a series of GO actions. This breaking-down is done by the planning system at the same time as the translation to an action plan representation takes place (see further section 2.2).

---

[1]Compare the language used in the COPPS system (see,for example, A comparison of COPPS and the Base Model [SUTEC 88]) in which it is possible to describe several other types of action plans.

2.1.2. Conditional statements.

If during an assignment certain tasks are to be carried out depending, for example, on a sensor value at a certain time, the task assignment language must allow for conditional statements. This means that we can define branched action plans. An example of what a conditional statement can look like is:

```
(IF boolean expression
      THEN arbitrary task statements
      ELSE arbitrary task statements )
```

2.1.3. Variables.

Defining variables may also be necessary. This implies the creation of a named entry in the state vector of the PARA (programming paradigm for robotics applications) model [Sandewall 88-2]. Later the entry can be assigned a value.

```
(DECLARE Foo AN INTEGER)
(SET Foo 25)
```

In the example above, first an input to the state vector is created and given the name Foo. Foo can take on integer values. Then Foo is assigned the value 25.

2.1.4. Loops.

To carry out a task several times in succession the task specification language must allow for the definition of loops. For instance, a while construction can be defined as follows:

```
(WHILE boolean expression
      DO arbitrary task statements  )
```

This statement implies that the task defined after DO is repeated as long as the boolean expression is true.

2.1.5. Parallel actions.

In the section Task specification language the impossibility of defining the parallel execution of several navigation actions was mentioned. We amy wish to get round this limitation on occasion. Imagine, for example, that the vehicle can carry out the following actions: "Maintain constant distance from bottom" and "Move forward". The above syntax does not allow us to define the parallel execution of these two actions. If it did, greater generality would be achieved. This would obviate the need to define a primitive action "Maintain constant distance from bottom *and* move forward".[2]

Parallel actions are also required if the vehicle is to carry out other types of atomic actions such as, for example, "Stretch out arm". It will then be possible for us to specify that the arm is to be stretched out at the same time as the vehicle moves forward.

We introduce the following notation to allow the definition of parallel actions:

```
(PAR ( arbitrary task statements )
      ( arbitrary task statements  ) ... )
```

[2]A precondition for two navigation actions to be carried out in parallel is that either they use different effectors or that the output data from the two actions can be combined before being sent to other effectors involved. This is not possible for all types of navigation actions, so the system must be able to check whether a given parallelism is allowed.

This means that the groups of arbitrary task statements will be carried out in parallel.

2.2. Action plans in the analysis layer.

The analysis layer's work is twofold: to translate a given task to an action plan, and to ensure that the plan is followed during execution. If the plan is not kept to, or cannot be followed, the analysis layer must take steps to ensure that the task can be carried out, or if this is not possible, choose other appropriate steps. Thus the analysis layer must contain a representation of the action plan including those actions that have already been carried out. This enables the planning system to see what has happened and as a result, together with information about the goals the vehicle is to achieve, be able to draw correct conclusions about what to do in a given situation. If the analysis layer is to include a history, this means that the representation must be updatable during the execution of a task. Whenever the vehicle is forced to carry out an unforeseen action , this must be added to the history. By way of illustration, see the action plan as given in figure 2. Movement is constantly forward along the x-axis as time elapses. Ahead of the now-line are those actions yet to be carried out; behind the now-line are those actions the vehicle has already carried out; and those actions that cut the now-line are taking place at this very moment.

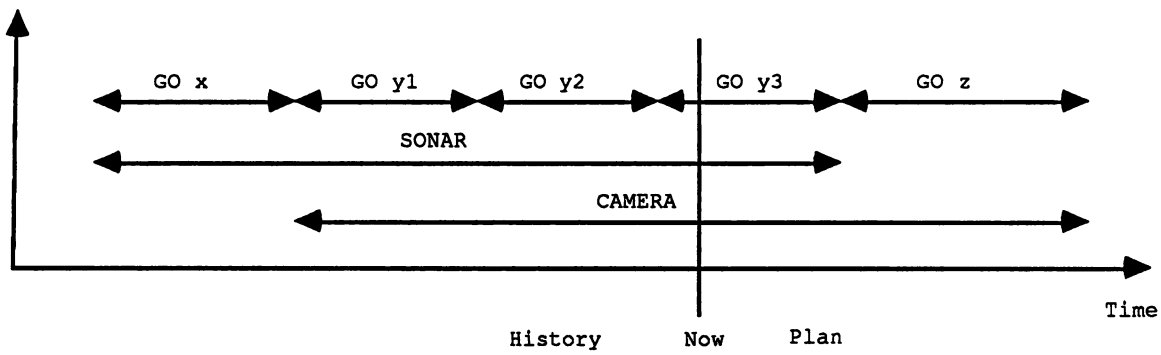

Figure 2. Action plan including history. The action plan is for the task specified in section 2.1.1.

In [SUTEC 88] we mentioned that the actual execution of a task could be shown on the operator console. To achieve this the action plan in the analysis layer must be visible for the system in the operator console. This is done by sending the action plan of the analysis layer to the console system each time the plan is changed. Transmitting this information allows the operator to follow any re-planning (see section 3.2). If the analysis layer's history is also transmitted, the operator will be able to see the actual execution of the task.

2.2.1. Representation av action plans.

This discussion suggests that the representation of an action plan in the analysis layer can be divided into two parts, namely a plan part and a history part.

The basis for the representation of the history is that for each action a start and an end time-point is given. As an example, the history from figure 2 can be shown as follows:

```
(START (SONAR)(TIME 234))
(START (GO (POS x))(TIME 234))
(END (GO (POS x))(TIME 538))
(START (CAMERA)(TIME 539))
(START (GO (POS y1))(TIME 539))
(END (GO (POS y1))(TIME 712))
(START (GO (POS y2))(TIME 713))
(END (GO (POS y2))(TIME 810))
(START (GO (POS y3))(TIME 811))
```

where START and END specify the start and end times for an action. TIME specifies the exact time of the event. It is highly conceivable that other information may be associated with start and end times, for example the value of certain sensors.

The same syntax is used to represent the action plan for the future. However, certain points must be borne in mind. When representing history there is always a total ordering[3] of events that have occurred. This is not the case when representing the plan for the future actions of the vehicle Instead there is a partial ordering of the actions. This means that another level in the representation is called for, where we can specify that a number of actions are not mutually restricted by time ordering. This can be achieved by, in addition to START and END, introducing a reserved word PAR used analogously to 2.1.5.

The example in figure 2 does not require the PAR construction. The order of actions is specified in the task. A representation of the plan for future actions may look like this:

```
(END (GO (POS y3)))
(END (SONAR))
(START (GO (POS z)))
(END (GO (POS z)))
(END (CAMERA))
```

Of course here, too, information the analysis layer may have user for in, say, re-planning may be associated with the various events.

To ensure that the above model with separate history and plan works, it is required that the analysis level has access to information about when an action starts and ends. This can either be transmitted by underlying levels or by allowing the analysis level itself to check the ongoing execution of actions. Both have pros and cons. The latter implies that the analysis level can discover whether an action has failed, but also that the analysis level must be constantly active and thereby require computer resources. The former allows less possibility for the analysis level to discover discrepancies during execution, but requires fewer computer resources. Supporting the latter is the fact that, if the analysis level is to carry out successive planning while a part of the plan is being executed,[4] it must be constantly active and can thus gather information on the course of the execution.

---

[3] In strict mathematical terms there is not a total ordering as there are events which occur at exactly the same time. The order, however, is more strict than the partial ordering we have for future events where unrelated events in the partial order are not restricted to occur at the same time.

[4] This may be required, for example, if the vehicle is planning a lengthy route. The analysis level can work out a rough plan which specifies the approximate direction of the vehicle. This planning is tuned as the vehicle travels. If all planning were to take place before departure, the starting up time would probably be very long.

2.2.2. Transformation of task to action plan.

Transforming a given task to a representation as shown above can be seen as a kind of compilation. Certain statements in the task are not primitive actions, but must be translated into such. This translation might require that the system must carry out some kind of planning. If the task specifies that the the vehicle is to move to position X, the analysis layer must check with the map module that this is possible. A direct path to X may not be available, which means that the system has to plan a route. This is discussed in the chapter: Realizing picture and map information in the system tool.

The analysis layer must also be able to decide on which actions can be carried out in parallel. This implies that the analysis layer needs information about the resources each primitive action requires. Two actions requiring the same resource can not normally be executed in parallel.

The analysis layer must also be aware of vehicle limitations which may influence the course of the task. As an example, take the task "Move through the narrow sound S". This puts strict demands on the vehicle's positioning ability. If the vehicle's internal position is faulty, there is a good chance that it will not be able to succeed in its task; it could, for example, run aground. To avoid this the analysis system can add an extra calibration action just before the vehicle passes through the sound, thus minimizing the risk of the internal position being wrong.

All this knowledge required by the analysis layer must somehow be stored. One candidate representation is to store the knowledge as rules, thereby making it easy to make changes to the knowledge.

When compiling from task to action plan, the rules always operate on one task statement at a time. The statement can be matched (by unification, for example) with the precondition part of the rules in the system. The rules thus matched can then be carried out. As an example of rules:

```
((ACTIVATE ?X) => (AddToPlan (START ?X))
((DEACTIVATE ?X) => (AddToPlan (END ?X))
((GO ?POS) => (AddToPlan (PLANROUTE ?POS))
```

In the last example PLANROUTE is a call to the route planner which plans a path to a given position. Note that this function can return a provisional route if the time to plan a complete route is too long.

2.3. Action plans in the rule layer.

One of the primary duties of the rule layer is to administrate the flow of control between the various actions so that they are executed at the right time. To be able to do this the rule layer must have access to an action plan which specifies which actions are to be carried out. The rule layer uses this plan to initiate new actions etc.

2.3.1. Representation of action plans.

Everything in the rule layer is closely connected to the state vector defined in the system tool. As the name suggests, the rule layer contains a number of rules. These operate on the various entries declared in the state vector. This is accomplished as follows: Each rule has a precondition part and a post-condition part. The precondition part specifies the values that certain entries in the state vector must have in order that the post-condition part can be carried out. The post-condition assigns values to one or more entries. The value could, for example, be a message to the analysis layer that it should take some precautionary measure (see further in [Sandewall 88-1]). The action plan,

too, is represented in the form of rules. These define the order in which the actions are to be executed.

The system's state vector can be divided into four groups: one for sensor signals etc., one for actuators, one for actions and finally one for rules. Every primitive action the vehicle can carry out has an entry in the state vector. For each action it is indicated whether the action is OFF, ACTIVE or COMPLETED. There is also an entry for every actuator in the vehicle. This indicates which action at the moment is using which actuator. For each rule in the rule layer there is an indication as to whether the rule is OFF or ACTIVE.

Figure 3 shows the division of the state vector into groups. We can also see that the vehicle is just now executing the actions GO and SONAR. The action GO uses actuators P1 and P2. There is one active rule, namely R1. The figure also shows that the vehicle is located at position 231 at a depth of 34 metres with the water temperature at 6° C.

Parameters	Depth	34
	Temp	6
	Pos	231
Actuators	P1	GO
	P2	GO
	P3	FREE
Actions	GO	ACTIVE
	CAMERA	OFF
	SONAR	ACTIVE
Rules	START	OFF
	R1	ACTIVE
	R2	OFF
	R3	OFF
	R4	OFF

Figure 3. State vector.

To exemplify how an action plan can be represented by means of rules, we can examine the task given in section 2.1.1.

Rule	Precondition	Post-condition
START:	START = ACTIVE	START := OFF SONAR := ACTIVE GO x := ACTIVE R1 := ACTIVE
R1:	R1 = ACTIVE GO x = COMPLETED	R1 := OFF GO x := OFF GO y1 := ACTIVE CAMERA := ACTIVE R2 := ACTIVE
R2:	R2 = ACTIVE GO y1 = COMPLETED	R2 := OFF GO y1 := OFF GO y2 := ACTIVE R3 := ACTIVE

```
R3:     R3 = ACTIVE              R3 := OFF
        GO y2 = COMPLETED        GO y2 := OFF
                                 GO y3 := ACTIVE
                                 R4 := ACTIVE

R4:     R4 = ACTIVE              R4 := OFF
        GO y3 = COMPLETED        GO y3 := OFF
                                 GO z := ACTIVE
                                 SONAR := OFF
                                 R5 := ACTIVE

R5:     R5 = ACTIVE              R5 := OFF
        GO z = COMPLETED         GO z := OFF
                                 CAMERA := OFF
```

At the start all rules are OFF. Once the operator signals the vehicle to carry out its task, the start rule is set to ACTIVE. The precondition is immediately fulfilled and the post-condition is carried out. This switches the start rule to OFF as this rule is no longer required. The actions GO and SONAR are set to ACTIVE. An action being set to ACTIVE implies that the process layer activates the corresponding processes that carry out the action. When the action has been executed, this is indictated by the process layer by switching the action to COMPLETED. Finally a new rule, R1, is activated. The precondition of this rule is not fulfilled until the GO action has been executed. Only then can the post-condition be executed whereby new actions are initiated.

The entry in the vector for actuators is used when several parallel actions are to be carried out. Here we can forbid actions that use the same actuators from being executed in parallel. This is achieved by reserving actuators for an action by assigning the action's name to the relevant actuators' entries. In the precondition of the rules which activate the relevant actions, a check is made that the desired actuators are not reserved by any other action. On completion of the action that reserved the actuators, they are released and can be reserved for a new action.

The system can fall into a deadlock situation if the rules in the rule layer are improperly defined. This must be analyzed by those systems that define the rules to ensure that such a situation can not occur.

By representing a plan of action by means of rules as described above, all action plans derivable from a task expressed in the task specification language (as described) can be represented in an efficient and clear way.

2.3.2. Transformation of action plans to rules.

The rule layer's plan of action is calculated from the analysis layer's plan of action. Unlike the compilation of the task to an action plan, no planning is needed here. We can, therefore, make use of more ordinary compiling techniques.

Compiling does not consist solely of creating rules. The system must also associate the action plan of the analysis layer with information about which rules there are, which actions are controlled by which rules etc.

2.3.3. Compiling rules.

Rules can also be compiled in order to increase the efficiency of the rule layer. In Lisp, for example, this could be achieved by defining every rule as a function. If such a function is subsequently compiled by the usual Lisp compiler, the execution of rules will be much faster that if the rule layer itself has to interpret every rule. Ways of doing

this will, naturally, be highly dependent on which programming language is chosen to implement the rules.

3. Response rules.

The previous section dealt with how the vehicle's planning system generated and then executed an action plan for a task. This section deals with how and what happens when the vehicle discovers that the current action plan is not appropriate for the present situation.

Such situations can arise in different ways. While navigating the vehicle may discover previously unforeseen obstacles which make the current navigation action impossible to execute. In chapter 1 of [SUTEC 88], scenario 2 an example of this was given; the vehicle comes across a patch of reeds which was unknown of when the task was planned. Another unexpected situation can arise when the source of energy begins to run out, or if the vehicle is suddenly attacked by another vehicle when on a reconnaissance assignment. In these cases the vehicle ought to abandon the previous plan and concentrate on survival instead.

Even foreseen circumstances can lead to a failure of the action plan. Imagine the vehicle patrolling a certain area and inspecting all objects of a certain type that it comes across during the assignment. Here you are expecting these objects, but you do not know when you will discover them. In this case the original plan contains no inspection actions; these are added to the action plan as the objects are discovered.

The above example illustrates that the vehicle must be constantly prepared for different situation to arise. The rest of this paper discusses how the vehicle realizes such preparation and also how it revises its action plans to cope with such situations.

3.1. Realizing the response.

To program the vehicle to be able to react in various situations, you often want to say: if situation A occurs, carry out action plan B. Typical examples such as "if the energy reserve is running out, abandon plan and come home", "if you get shot at, make a run for it" etc are best expressed just in this way - as rules. A rule machinery is thus the most natural way to realize them.

A rule machinery makes use of a set of rules to ensure that the action plan corresponds to correct behaviour for the vehicle in the current situation. These rules are permanently active and check various parameters: sensor data, current actions, the internal state of the vehicle, fault detectors etc. If and when a rule's preconditions are fulfilled, the rule's post -conditions are executed.

In the simplest cases such a rule states, for instance, that the state of operation is to be changed or the TV camera started. Such cases can be carried out without needing to re-plan the rest of the task. In other cases a rule may state that more complex actions that do not fit into the current action plan are to be carried out. In such cases the system must revise its plan of action. How this revision is done is described in more detail later in this paper.

In the typical case, when a rule is triggered, an interrupt in the system is generated so that the steps mentioned in the rule can be incorporated into the action plan. However, it may be appropriate to divide the rules between various levels in the system, for two main reasons:

First, reaction time, ie. the time that elapses from the time-point a rule may be executed to when it actually is executed. A rule requiring fast response is, for instance, one that would prevent the vehicle from crashing into a rapidly approaching object. Such rules

ought to be at a low level in the system where reaction times are quicker, while rules with lower priority that do not need immediate execution can be at a higher level.

Secondly, a rule ought to be at the level where is can best be dealt with. Rules controlling a single action work best at the same level its execution is handled, ie. at the lowest level. Rules dealing with the general state of being of the vehicle ought to be at a higher level.

There are three different levels in the system model described in this report: process-, rule- and analysis-levels. The response rules are divided, for the reasons given, between these three levels.

3.1.1. Definition response rules.

Before discussing the response rules at the different levels, a brief mention of how these rules are entered into the system follows. We divide the rules into two main types: permanent and task-specific response rules.

Permanent rules are those pertinent to every task the vehicle carries out. As the rules are independent of the task, it would be inefficient and unnecessary to feed them in for each new assignment. They can be loaded into the system once and for all, for example when the system is initialized.

During preliminary testing of the vehicle, changes in these rules be be required, for example to test various combinations of rules. Different response rules may be required for various vehicle configurations. Thus the ability to easily make changes in the permanent rule definitions may be required and they should accordingly not be "hard-coded" into the system. It is better to separate the rules from the basic system. The rules can then be loaded in separately, like a task. Analogous to the previously described task specification language, this requires a special response rule language.

Task-specific response rules are special rules to be valid during one, or part of one, task. It must therefore be possible to load them with the task. The previously described task specification language can therefore be extended with statements that define, activate and deactivate response rules. Task-specific response rules can, however, be treated internally in the same way as permanent rules.

The patrol assignment mentioned earlier, where all objects of a certain type should be inspected, could be specified be means of a task-specific response rule. Navigation actions for the task could be expressed as normal in the specification language. Only during the task, though, would a response rule be activated. This rule would state that when an object of the required type is discovered, ie. a sensor gives out a signal, the object is to be inspected before the patrol task continues. An inspection of the object is defined as a normal task and is specified in the post-condition of the response rule. This rule will augment the normal action plan with a subplan to carry out the inspection each time it is triggered, ie. each time an object is discovered.

3.1.2. Response rules in the process layer.

The lowest level, the process layer, is also the fastest. This is where the vehicle's rapid reactions and reflexes are realized. Single tasks are executed at this level by means of data flow in a process network. There are also processes ensuring that tasks have been completed. These tasks control the data flow.

We illustrate with an example. An action of the type (GO X) is realized by two subprocesses, one that drives the vehicle towards X and the other which signals when the vehicle reaches X. The signalling is achieved by assigning the value COMPLETED at the corresponding slot for the GO action in the global state vector. The process-module handler switches off the GO action process.

A rule which checks whether a GO action is functioning can be realized in the control process described above. This can also check that the vehicle is moving towards X the whole time. Otherwise a fault has arisen. The vehicle may have got caught in a strong current. The control process is therefore extended so that it can signal if the corresponding action is not working. The signal might be named FAILED, and would be sent to the action entry in the state vector just like the COMPLETED signal. The rule layer is then responsible for taking suitable steps.

Of course a control process could be expected to generate more signals than COMPLETED and FAILED. It might perhaps explain why the action failed, not just the fact that it has failed, ie. there may be several types of FAILED signals.

It is, therefore, appropriate to implement the response rules in the process layer as control processes. This provides very fast reaction times. In the GO task example, the control process might also check if the obstacle warner has reacted, and, if so, stop the vehicle quickly to avoid collision.

3.1.3. Response rules in the rule layer.

The rules that control the vehicle's somewhat more intelligent response behaviour can be at the system level where the task plan is executed, ie. the rule layer. Examples of such rules are:

- if the energy supply is too low, re-plan
- if a GO task has failed, re-plan
- if a "surveillance warner" reacts, change the operation state to SILENT

At this level the response rules are appropriately implemented as rules which work with the previously described state vector. All the information required is found in this vector. There the rules can watch what is happening in and around the vehicle. The rules can do more than simply read off the vector. They can also write to the vector. In this way actions can be quickly started and stopped, operational state changed etc.

If, for example, the surveillance warner's entry contains a T, a rule is triggered which sets the value of the operational state entry to SILENT. The vehicle quickly changes to a quieter behaviour; this is achieved by the lower process layer sensing and adapting to the current operational state. These processes behave differently depending on the current operational state.

It is also possible to allow different rules to be active in different situation. Every operational state could possibly use different response rules. In certain states you may wish to suppress rules purely for reasons of efficiency. In the manual state, you may wish to suppress rules that automate vehicle behaviour.

Suppressing and reinstating response rules can be realized in the same way as the ordinary plan execution rules are. Each rule is assigned an entry in the global state vector. The value of a rule entry may be T or F depending on whether the rule is active or not.

Changes in the vector, though, are not always enough. A situation may call for radical changes of the current action plan. Consequently the response rules must be able to signal this upwards to the analysis layer. It is then up to this layer to take suitable steps. This signalling can be achieved by, for instance, reserving one or more entries in the state vector for this purpose. The analysis vector can then watch these entries so as to be able to intervene and re-plan when necessary.

Re-planning, though, can put heavy demands on time and resources. The vehicle cannot therefore be allowed to be totally inactive while the analysis layer is re-planning.

The vehicle might be under attack and ought to be on its way out of there. Obviously the vehicle cannot take too long in considering how to get away. It is probably better to escape in a panic and plan a more intelligent escape route while the more primitive escape is under way.

This can be achieved by the response rules linking in more primitive behaviour, for example by changing to an operational state PRIMITIVE ESCAPE, at the same time as a signal is sent to the analysis layer. In this way the vehicle can deal directly with an attack at the same time as the analysis layer has a little time to analyze the situation. When the analysis is done, the new and hopefully more intelligent action plan can take over control of the the vehicle's actions and behaviour.

Using a set of rules for continual update of a set of parameters involves the risk that several rules will assign values to the same parameter simultaneously. The probability for this happening increases with the number of rules. There are at least two ways of dealing with this problem.

First, try to see if this is so when the rules are being written, ie. before any rules are executed. This check may be difficult to carry out completely automatically, but a control system to warn the programmer in doubtful cases should not be too difficul to construct.

Another method, that can be used in combination with the first, is by giving the rules priority and to decide at execution time which rule has highest priority. When rules are competing to assign a value to an entry, the rule with the highest priority succeeds.

3.1.4. Response rules in the analysis layer.

A subsystem in the uppermost analysis layer presumably ought to consist of a rule machinery that decides how the vehicle's action plans are to be revised with respect to those situations that can occur. These rules will need to call the various planning systems at this level.

By way of example, there is a rule that states that the vehicle is to try and move from X to Y, simply in order to get to Y, and an unforeseen obstacle turns up on the way. That part of the action plan taking the vehicle to Y must be revised. Furthermore the rule states that this may be done by exchanging the interrupted GO action for a new subplan which avoids the obstacle. This new subplan is generated by means of the route planner.

Another example: imagine the vehicle is out searching an area by navigating according to a certain pattern while, simultaneously, camera shots are being sent back to the mother ship. Assume now that communication ceases to function. Via the state vector the analysis layer receives a message from the lower rule layer that re-planning may be necessary. The rule states in this case that, if there is a breakdown in communication during an assignment that requires communication with the mother ship, then the task should be interrupted and the current action plan be replaced with a new one that takes the vehicle straight back to the mother ship. The new plan is generated by means of the route planner.

In this way it is possible to realize tasks such as to travel according to a certain pattern and inspect all objects that appear to be of a certain type. From the beginning the action plan only contains navigation actions to travel according to the given pattern. When the plan is executed there is a response rule in the middle rule layer which signals upwards about possible re-planning if the vehicle's sensor reacts in a way that suggest the object is of the given type. The rule at the analysis level then states that the action plan should be revised by including inspection actions at the beginning of the present plan.

3.2. Plan revision.

Plan revision occurs when a change is needed in the current action plan because a response rule in the analysis layer has so ordered. These response rules also state how the action plan is to be changed. It may be a case of changing one subplan for another, to change the whole plan for another or to insert a new subplan somewhere within the present one. You could say that the response rules edit the action plan.

When the action plan in the analysis layer is changed, the compiled rules in the rule layer must also be updated so that they agree with the new plan. The simplest way to do this is simply by generating completely new plan rules and replacing the previous ones with them. This is extremely inefficient, though, especially if a minor change is to be made in a major plan. There is a high risk, too, that the vehicle has to stop for a while each time re-planning revises the action plan. The vehicle will not be able to achieve smooth execution of its actions if the task requires constant re-planning.

Hence the need for incremental changes to the current action plan, not only in the analysis layer but also in the rule layer. This implies that the compiling of rules from the action plan (as described in a previous section) must be incremental. This should be possible by saving during the transformation, for example, information on which rules and which actions belong together .

4. An example.

A complete example is given in this section. We have chosen a variant of the previously described scenario 1, ie. the vehicle is to check whether there are traces of oil in three different locations specified by the operator by means of the graphic operator panel described in chapter 1 of [SUTEC 88].

In this variant we imagine that the task continues normally as far as the second sample-taking location. Then the oil detector sensor ceases to function. The vehicle realizes there is no point in continuing with the task. Instead it returns to the mother ship.

Below follows a description of how action plans and response rules are represented at the various levels of the system tool.

4.1. Representation of the task.

First the system must translate the graphical representation of the task to a textual representation as shown in section 4.2.1. Below we see how the task is represented:

```
1 (GO (POS 100))
2 (OIL-DETECTION)
3 (GO (POS 200))
4 (OIL-DETECTION)
5 (GO (POS 300))
6 (OIL-DETECTION)
7 (GO (POS 0))
```

The task consists of seven task statements. The task GO means that the vehicle is to travel to a given position. The action OIL-DETECTION means that an appropriate sensor is to examine if traces of oil are present in the water. This action is a primitive perception action, ie. its post-condition is not linked to another action. An atomic action "knows" when to stop.

4.2. Representation of the action plan in the analysis layer.

The task is then translated to an action plan in the analysis layer. This implies, among other things, that several of the GO actions that are defined in the task have to be expanded to several actions. In figure 4 we can see the route chosen for the vehicle by the planning system.

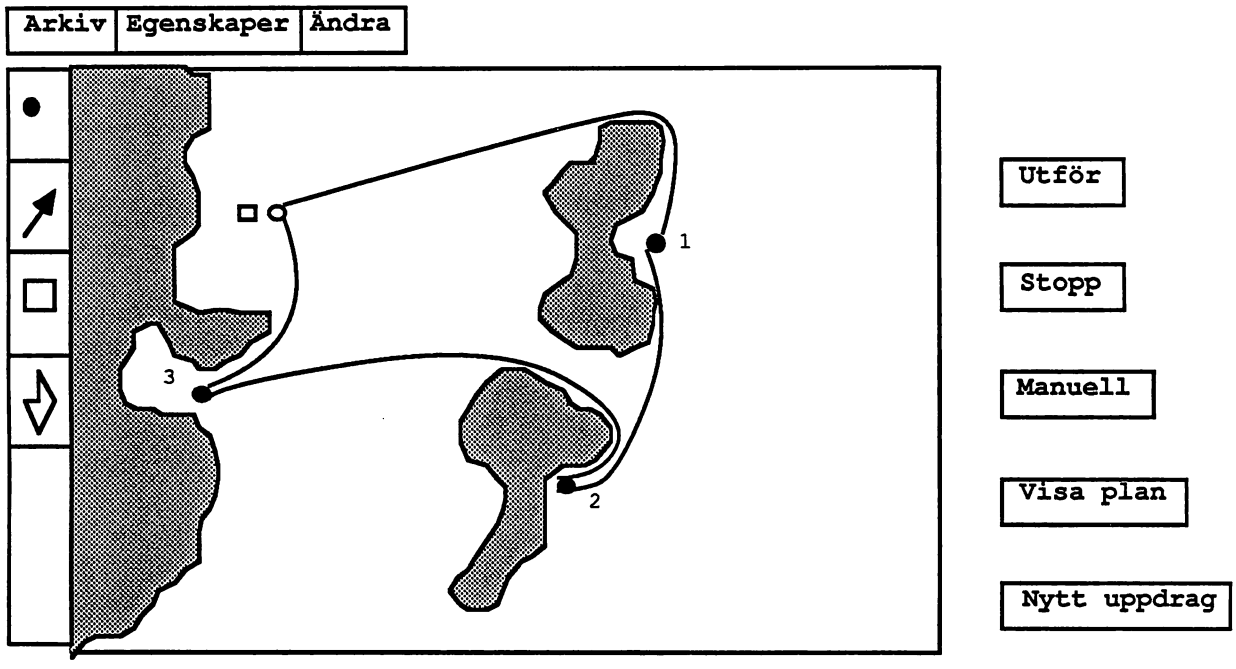

Figure 4.The choosen route for the vehicle.

The representation of the action plan in the analysis layer looks like this:

```
1 (START (GO (POS 80) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 80) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 90) (RUDDER 20) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 90) (RUDDER 20) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 100) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 100) (RUDDER 0) (DEPTH 5) (SPEED 3)))
2 (START (OIL-DETECTION))
  (END (OIL-DETECTION))
3 (START (GO (POS 130) (RUDDER 5) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 130) (RUDDER 5) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 177) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 177) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 200) (RUDDER 9) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 200) (RUDDER 9) (DEPTH 5) (SPEED 3)))

4 (START (OIL-DETECTION))
  (END (OIL-DETECTION))
5 (START (GO (POS 189) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 189) (RUDDER 0) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 230) (RUDDER -15) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 230) (RUDDER -15) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 300) (RUDDER -6) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 300) (RUDDER -6) (DEPTH 5) (SPEED 3)))
6 (START (OIL-DETECTION))
  (END (OIL-DETECTION))
7 (START (GO (POS 263) (RUDDER -4) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 263) (RUDDER -4) (DEPTH 5) (SPEED 3)))
  (START (GO (POS 0) (RUDDER -8) (DEPTH 5) (SPEED 3)))
  (END (GO (POS 0) (RUDDER -8) (DEPTH 5) (SPEED 3)))
```

We can see that the route planner has used the ability of the vehicle to travel in a circle segment by using the rudder. When translating, the system has also included information about speed, depth and other variables required for a GO action.

4.3. Representation of an action plan in the rule layer.

The action plan in the analysis layer is then translated to a set of rules. These operate on the global state vector in the system. This means that all identifiers to the left of = or := in the rules are named entries in the vector. In this example we have the following set of rules:

```
Rule        Precondition                  Post-condition

START:      START = ACTIVE                START := OFF
                                          GO := ACTIVE
                                          Destination := 80
                                          RUDDER := 0
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          R1 := ACTIVE


R1:         R1 = ACTIVE                   R1 := OFF
            GO = COMPLETED                GO := ACTIVE
                                          Destination := 90
                                          RUDDER := 20
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          R2 := ACTIVE

R2:         R2 = ACTIVE                   R2 := OFF
            GO = COMPLETED                GO := ACTIVE
                                          Destination := 100
                                          RUDDER := 0
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          R3 := ACTIVE

R3:         R3 = ACTIVE                   R3 := OFF
            GO = COMPLETED                GO := OFF
                                          OIL-DETECTION := ACTIVE
                                          R4 := ACTIVE

R4:         R4 = ACTIVE                   R4 := OFF
            OIL-DETECTION = COMPLETED     GO := ACTIVE
                                          Destination := 130
                                          RUDDER := 5
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          OIL-DETECTION := OFF
                                          R5 := ACTIVE

R5:         R5 = ACTIVE                   R5 := OFF
            GO = COMPLETED                GO := ACTIVE
                                          Destination := 177
                                          RUDDER := 0
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          R6 := ACTIVE

R6:         R6 = ACTIVE                   R6 := OFF
            GO = COMPLETED                GO := ACTIVE
                                          Destination := 200
                                          RUDDER := 9
                                          Required_Depth := 5
                                          Required_Speed := 3
                                          R7 := ACTIVE
```

```
R7:        R7 = ACTIVE                  R7 := OFF
           GO = COMPLETED               GO := OFF
                                        OIL-DETECTION := ACTIVE
                                        R8 := ACTIVE

R8:        R8 = ACTIVE                  R8 := OFF
           OIL-DETECTION = COMPLETED    GO := ACTIVE
                                        Destination := 189
                                        RUDDER := -6
                                        Required_Depth := 5
                                        Required_Speed := 3
                                        OIL-DETECTION := OFF
                                        R9 := ACTIVE

R9:        R9 = ACTIVE                  R9 := OFF
           GO = COMPLETED               GO := ACTIVE
                                        Destination := 230
                                        RUDDER := -15
                                        Required_Depth := 5
                                        Required_Speed := 3
                                        R10 := ACTIVE

R10:       R10 = ACTIVE                 R10 := OFF
           GO = COMPLETED               GO := ACTIVE
                                        Destination := 300
                                        RUDDER := -6
                                        Required_Depth := 5
                                        Required_Speed := 3
                                        R11 := ACTIVE

R11:       R11 = ACTIVE                 R11 := OFF
           GO = COMPLETED               GO := OFF
                                        OIL-DETECTION := ACTIVE
                                        R12 := ACTIVE

R12:       R12 = ACTIVE                 R12 := OFF
           OIL-DETECTION = COMPLETED    GO := ACTIVE
                                        Destination := 263
                                        RUDDER := -4
                                        Required_Depth := 5
                                        Required_Speed := 3
                                        OIL-DETECTION := OFF
                                        R13 := ACTIVE

R13:       R13 = ACTIVE                 R13 := OFF
           GO = COMPLETED               GO := ACTIVE
                                        Destination := 0
                                        RUDDER := -4
                                        Required_Depth := 5
                                        Required_Speed := 3
```

4.4. Normal execution of the task.

Nothing unusual occurs during the first part of the task so that rules START through R7 are activated and executed in turn. Each time actions start or finish this is noted in the analysis layer's history. This may appear as follows after R2 has been activated:

```
(START (GO (POS 80)(RUDDER 0)(DEPTH 5)(SPEED 3))(TIME 1008))
(END (GO (POS 80)(RUDDER 0)(DEPTH 5)(SPEED 3))(TIME 2098))
(START (GO (POS 90)(RUDDER 20)(DEPTH 5)(SPEED 3))(TIME 2099))
(END (GO (POS 90)(RUDDER 20)(DEPTH 5)(SPEED 3))(TIME 3067))
(START (GO (POS 100)(RUDDER 0)(DEPTH 5)(SPEED 3))(TIME 3068))
```

This history is used to generate a report of the course of the task.

4.5. An interruption in the action plan.

When the vehicle is to carry out oil-detection at position 2, the sensor fails to work. This is discovered as follows: Rule 7 has been executed and so the action OIL-DETECTION has started, ie. the corresponding entry in the state vector has been set to ACTIVE. Now when the sensor does not work, the action OIL-DETECTION will fail. This is discovered by a response rule (control process) in the process layer.This rule informs the other layers in the system by resetting the action's entry in the state vector to FAILED.

The following response rule is in the rule layer:

```
Resp1:      Resp1 = ACTIVE              RE-PLAN := T
            OIL-DETECTION = FAILED      Resp1 := OFF
                                        CAUSE := Failed oil-detection
```

By setting the entry RE-PLAN to T, the rule layer signals to the analysis layer that re-planning may be required. The cause of the possible re-planning is noted in a special entry called CAUSE. The response rules in the analysis layer are then responsible for deciding if and how re-planning is to be carried out.

As the action OIL-DETECTION fails, execution of the action plan will halt. Rule 8 will not be triggered as the precondition will never become true. This means that the vehicle will not set off towards position 3.

In our example the following rule in the analysis layer will be triggered:

```
CAUSE = Failed oil-detection  => Replace action plan with (KEEP_STILL).
                                 Call route planner with task
                                    (GO (POS 0)).
                                 Replace action plan with result provided by
                                 the route planner.
```

First this rule will stop the vehicle by changing the normal action plan to KEEP_STILL. This means that the vehicle will wait in the present position until a new plan has been devised. The new plan consists of actions to get the vehicle straight back to the mother ship. As soon as the new plan has been generated, it will replace the KEEP_STILL plan and the vehicle will set off on its way home (figure 5).

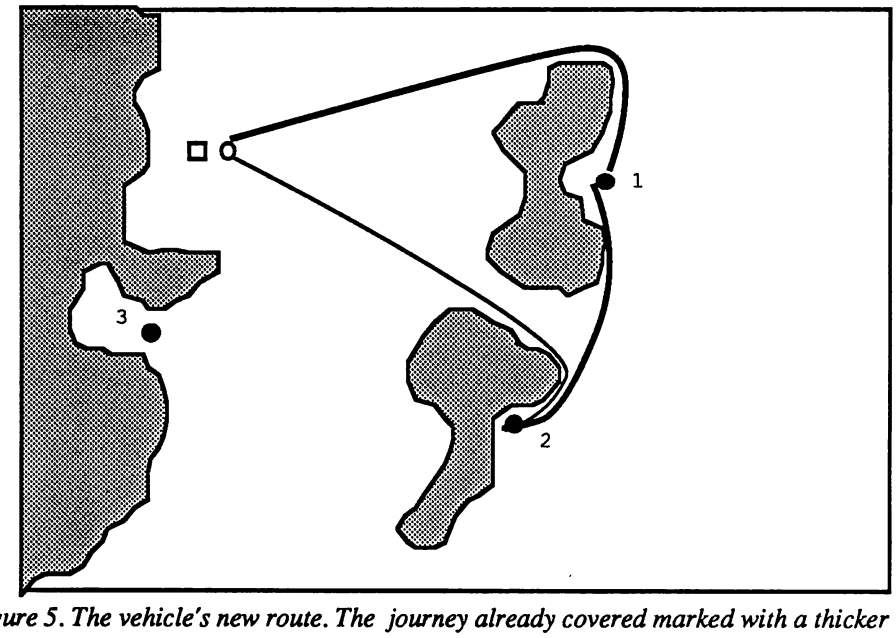

Figure 5. The vehicle's new route. The journey already covered marked with a thicker line.

The system can generate from the analysis layer's history a report explaining to the operator how the task has been accomplished. Below is the history from when the vehicle arrived at position 2:

```
...
...
(END (GO (POS 200)(RUDDER 9)(DEPTH 5)(SPEED 3))(TIME 5003))
(START (OIL-DETECTION)(TIME 5004))
(FAIL (OIL-DETECTION) (TIME 5005))
(START (KEEP_STILL) (TIME 5007))
(STOPPED (KEEP_STILL) (TIME 5100))
(START (GO (POS 245)(RUDDER -13)(DEPTH 5)(SPEED 3))(TIME 5101))
(END (GO (POS 245)(RUDDER -13)(DEPTH 5)(SPEED 3))(TIME 5890))
(START (GO (POS 0)(RUDDER 0)(DEPTH 5)(SPEED 3))(TIME 5891))
(END (GO (POS 0)(RUDDER 0)(DEPTH 5)(SPEED 3))(TIME 6704))
```

If an action fails, it is marked FAIL in the history. If the system tool is forced to interrupt an on-going action, it is marked STOPPED. In the above example the analysis layer's action KEEP_STILL is interrupted when the new route has been generated.

5. References

[Sandewall 88-1] Sandewall, Erik, **Om grundläggande programvaru-arkitektur för autonoma eller partiellt autonoma farkoster**, RKL Memo 88-1, IDA, Linköpings Tekniska Högskola, Sweden.

[Sandewall 88-2] Sandewall, Erik, and Hultman, Johan, and Tengvald, Erik, **Software architecture and programming paradigms for robotics applications**, Institutionsrapport 88-12, IDA, Linköpings Tekniska Högskola, Sweden.

[SUTEC 88] Sandewall, E., and Hultman, J., and Nyberg, A., and Tengvald, E., and Finmo, L., **Utredning om system och system-verktyg för autonoma undervattensfarkoster.** IDA, Linköpings Tekniska Högskola, Sweden.

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-24

Upplagans storlek:
Number of copies:

Datum:
Date: August 1988

Projekt:
Project:

Titel:
Title: Realizing Action plans and Response Rules in a System Tool for an Autonomous Vehicle

Författare:
Author: Johan Hultman, Anders Nyberg

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

Abstract: This report describes the relationships between the various levels in a system tool for an autonomous vehicle. The tool referred to in this report is described in two earlier reports. The system tool consists of three levels of layers, each one having a representation of the action plan the vehicle is to carry out. We suggest how tasks and action plans can be represented in the various levels of the system tool. Furthermore we discuss how unforeseen events can be dealt with at the various levels.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

IDA - institutionen för datavetenskap
vid Universitetet och Tekniska högskolan i Linköping

Tidigare rapporter i industriserien:

1982 **Erik Sandewall:** Ny teknik i kontorsdatasystem. (LiTH-MAT-R-82-17)
(Föredrag vid Nord-Data 82 i Göteborg, juni 1982.)

Sture Hägglund: Informationshantering i det elektroniska kontoret. (LiTH-MAT-R-82-27, ingår även i rapportserien från LIBLAB)
(Föredrag vid den 5:e Nordiska IoD-konferensen, Trondheim, juni 1982)

Anders Haraldsson: INTERLISP - en avancerad integrerad programmeringsomgivning för LISP-språket. (LiTH-MAT-R-82-29)
(Föredrag vid Nord-Data 82 i Göteborg, juni 1982.)

Hans Grunditz, Uwe Hein, Erik Tengvald: Artificiell intelligens i framtidens CAD/CAM-system (LiTH-MAT-R-82-32)
(Föredrag vid de nordiska CAD/CAM-dagarna, Göteborg, november 1982)

Software Systems Research Center. Progress Report 1982.
(Översikt över forskningsprojekt, personal och publikationer vid Datalogicentrum, LiTH.)

1981 **Ola Strömfors, Lennart Jonesjö:** The Implementation and Experiences of a Structure-Oriented Text Editor. (LiTH-MAT-R-81-03)
samt
Ola Strömfors: ED3 - Användarhandledning.

Sture Hägglund m fl: 80-talets elektroniska kontor: Erfarenheter från LOIS-projektet. (LiTH-MAT-R-81-04)

Peter Fritzson: Distribuerad PATHCAL: Förslag till ett distribuerat interaktivt programmeringssystem för PASCAL. (LiTH-MAT-R-81-05)

Dan Strömberg: Datorn - hjälpreda eller hot i det lilla företaget? (LiTH-MAT-R-81-06)

Software Systems Research Center. Progress Report 1981.

1980 **Kenth Ericson, Hans Lunell:** Redskap för kompilatorframställning (LiTH-MAT-R-80-39)

Software Systems Research Center. Preliminary Report 1980.

1979 **Arne Börtemark, Hans Lunell:** Implementering av Pascal på minimaldatorn: en tillbakablick. (LiTH-MAT-R-79-3)

Arne Börtemark: Felbekämpningsmedel, en första översikt. (LiTH-MAT-R-79-10)

Jerker Wilander: An interactive programming system for Pascal. (LiTH-MAT-R-79-37)

1978 **Kenth Ericson:** Pascal i Sverige. (LiTH-MAT-R-78-8) --- *Slut* ---

Jerker Wilander: Interaktiv programutveckling i Pascal - Programmeringssystemet Pathcal. (LiTH-MAT-R-78-20)

Erik Sandewall: Programmeringsteknik för flexibilitet. (LiTH-MAT-R-78-21)

Erik Sandewall m.fl.: The Linköping Office Information System (LOIS). An Overview of Facilities and Design. (LiTH-MAT-R-78-22)

Claes Strömberg, Henrik Sörensen: Beskrivning av fontsystemet och erfarenheter vid systemutvecklingen. (LiTH-MAT-R-78-23)

1977 **Harold W. Lawson, Jr.:** Future Trends for Mini-computer Systems. (LiTH-MAT-R-77-4)

1976 **Sture Hägglund, Östen Oskarsson:** En teknik för utformning av användardialoger i interaktiva datasystem. (LiTH-MAT-R-76-13)

IDA - institutionen för datavetenskap
vid Universitetet och Tekniska högskolan i Linköping

omfattar ämnena *administrativ databehandling, datalogi*, och *telesystem*, med undervisning främst inom civilingenjörsutbildningen, datavetenskaplig linje och systemvetenskaplig linje. Inom IDA finns en bred forskningsverksamhet och ett forskarutbildningsprogram som leder fram till licentiat- och/eller doktors-examen. Institutionens forskning bedrivs f.n. i nio stycken grupper (laboratorier), som bl.a. arbetar inom områdena algoritmkomplexitetsteori, AI omgivningar, applikationssystem, CAD för elektronik, biblioteks- och informationssystem, logikprogrammering, databehandling av naturligt språk, programmeringsmiljöer, kunskapsrepresentation i logik samt i administrativ databehandling.

Industriserien

är en serie forskningsrapporter som särskilt vänder sig till den som arbetar med program- och maskin-varufrågor i industriell miljö eller inom den offentliga sektorns databehandling. I serien ingår dels rapporter från forskningsprojekt i Linköping som är av särskilt intresse för praktiker, dels översikter över aktuell forskning i världen i övrigt. Dessutom ingår presentationer av målsättning och strategi för institutionens forskning.

Hittills har utkommit:

1988

Jonas Löwgren: History, State and Future of User Interface Management Systems. (LiTH-IDA-R-88-14)

Arja Vainio-Larsson, Rebecca Orring, Peter Åberg: Metoder för utvärdering av människa-maskin gränssnitt: En litteraturstudie. (LiTH-IDA-R-88-20)

Johan Hultman, Anders Nyberg: Realizing Action Plans and Response Rules in a System Tool for an Autonomous Vehicle. (LiTH-IDA-R-88-24)

1987

Arne Jönsson: Naturligt språk för användardialog och databasförfrågningar. En sammanfattning av ett föredrag vid konferensen *Effektiv människa - dator-interaktion*, Linköping, 18-19 augusti 1987. (LiTH-IDA-R-87-25)

1985

Harold W. Lawson, Jr.: Sabbatical Report. (LiTH-IDA-R-85-05)

Sture Hägglund, Henrik Nordin, Roland Rehmnert, Kristian Sandahl: Utveckling av kunskapsbaserade applikationssystem i samverkan högskola - näringsliv (LiTH-IDA-R-85-09)

Harold W. Lawson Jr.: Swedish Participation in the Malaysian National Microelectronics Programme Also in *Proc. of ERSA (Economic Relations with Southeast Asia) Symposium*, Stockholm, October 21-25, 1985. (LiTH-IDA-R-85-11)

1984

Bengt Lennartsson: Programvarumiljöer. Produktionsteknik för programvara i Ada och andra språk. (LiTH-IDA-R-84-01)

Michael Pääbo: CAD-elektronik idag och i framtiden. (LiTH-IDA-R-84-03)

Erik Sandewall: Fjärde generationens Programvaruutbildning. (LiTH-IDA-R-84-13)

1983

Michael Pääbo: Introduktion till datorgrafik. (LiTH-IDA-R-83-02)

Mats Lenngren et al.: Datorgrafikdagar 7-9 juni 1983. (LiTH-IDA-R-83-05)

Pär Eman [illegible] -R-83-06)

Sture Häg [illegible] eknik för applikations [illegible] tem. (LiTH-IDA- [illegible]

Erik Sande [illegible] kunskapsöverföringsprogram. (LiTH-IDA-R-83-10)

(*Fortsättning på pärmens insida.*)