

# An Exception Handling Model Applied to Autonomous Mobile Robots<sup>1</sup>

G.A. den Boer, G.D. van Albada, L.O. Hertzberger, G.R. Meijer<sup>2</sup>

Department of Computer System

University of Amsterdam

Kruislaan 403

1098 SJ Amsterdam

The Netherlands

J.-B. Thevenon, P. LePage, E. J. Gaussens, F. Arlabosse

R & D department

Framentec S.A.

Tour Fiat

Cedex 16

92084 Paris la Défense

**Abstract.** In traditional robot applications the successful execution of the robot task is ensured by reducing the uncertainty in the environment to an absolute minimum. Extending the applications of robots into less controlled environments requires that the robot be able to recognize and respond to unforeseen and non-nominal situations. Mobile robots or manipulators designed for a partially structured environment provide a prime example of such an application.

In this paper we describe a robot architecture suitable for robot applications in incompletely known environments. It is based on concepts of elementary operations and exception handling developed at our institutes

## 1. Introduction

System autonomy is a key technology to increase the reliability and performance of systems which goes well beyond that of robotic manipulators only. The most salient features which distinguish a system in a known or certain environment from one in a partially known or an unknown environment are the sheer number of parameters that play a role, the non-linearity of the parameters with respect to the status of the environment and last but not least the interplay of basically different technologies contributing to the behaviour of the system. When these conditions hold, a-priori planning of actions based on some model has proved not to guarantee failure free execution. System autonomy is needed to deal with these circumstances.

---

<sup>1</sup> Was published in the proceedings of the IAS-3 conference, Pittsburgh, U.S.A, Feb. 15-19, 1993: "Intelligent Autonomous Systems - IAS 3," F.C.A. Groen, S. Hirose, C.E. Thorpe (editors), IOS Press, p. 297-306 (1993)

<sup>2</sup> Now at CMG, Amstelveen, Netherlands

Examples are the complex systems built for space exploitation in which mechanics, chemistry and electronics must harmonize to a great extent to give a successful mission. Or, closer to home, the car industry which sees a growing competition to provide comfortable and reliable driver support in congested traffic situations.

The work reported in this paper was performed for ESPRIT project nr 2043 "Mobile Autonomous Robot in an Industrial Environment" (MARIE). The objective of this project was the development of integrated hardware and software systems to give autonomy to robot vehicles in a dynamically changing environment. The project involved collaboration between a number of industries, research institutes and universities from all over Europe<sup>3</sup>. It had a duration of four years (1989 - 1992). Two vehicle testbeds and a robot manipulator testbed were developed.

In previous publications we proposed an architecture for an robot manipulator [1], [2]. In this paper, we report on the concepts underlying the software architectures for the vehicle testbeds and describe the implementation approach for the vehicle testbeds. The architecture described is based on the concepts of "elementary operations" and "exceptions". Elementary operations describe the basic capabilities of the robot in the areas of sensing, control and path-planning. Several elementary operations may be executed simultaneously in an "action".

Exceptions signal the occurrence of non-nominal conditions during the execution of an elementary operation and/or during the subsequent selection of a follow-up action.

## 2. High-level control concepts

As stated, the use of autonomous systems lies in their ability to perform a given task (goal driven behaviour) in an unpredictable and changing environment (flexibility). Both goal driven behaviour and flexibility are essential.

Flexibility requires that a system be able to function in various different circumstances. At the low level, this implies a need for a versatile set of logical sensors [3], [4], and a choice of various control mechanisms [5], each suitable for a particular control domain. Selection of a particular set of sensors in combination with a control mechanism will lead to a particular behaviour of the system.

Within each behaviour, we can distinguish three types of "elementary operations", viz. control operations, sensing operations and path-planning operations (quite distinct from the task-planning operations that we shall describe later).

To a large extent, the low level mechanisms can be given the capacity to recognize whether or not they are valid in the current sensing/control domain. By allowing mechanisms to seize or yield control, depending on the perceived state of the system and its environment, a valid behaviour can be selected at all times. This

---

<sup>3</sup> The partners in the MARIE project were: VOLMAC (NL, prime contractor), Robert Bosch GmbH (D), Framatome (F), Framentec (F), Hitec (GR), IAI (SP), Indecon(GR), University of Amsterdam (NL), University of Strathclyde (UK).

approach leads to Brooks's [6] subsumption architecture. We have to state that if we use the term behaviour we do not use it as defined by Brooks. To avoid confusion we will use the word action.

However, in a given state of system and environment more than one action may be valid. Which action will be most appropriate is also determined by the immediate goal the system should achieve. E.g. when directing a vehicle to a nearby goal position, it may be most effective to rely on a dead-reckoning-based action, whereas a distant goal may be reached with better positional accuracy if reference is made to known features in the environment, such as walls and corners.

Design and understanding of an autonomous system are helped by separating the selection mechanism for action from the action themselves. It facilitates the introduction of new elementary operations, the combination of elementary operations to achieve new action and experimentation with the sequencing of actions to achieve a complex mission.

In our architecture, elementary operations are activated by a high-level control mechanism which we call the "action dispatcher". More than one elementary operation can be activated at the same time, e.g. a control operation plus several sensor operations. We call such groups of simultaneous elementary operations "actions". When an action has run its course, or when it encounters problems that it cannot solve (exceptions) by itself, it reports back to the high-level mechanism.

Of course, the mere fact that an action report back to the higher level does not suffice to create a flexible system; the higher level must be able to select an appropriate follow-up action. The choice of the follow-up action must depend on the attained state of the system and the immediate goal. This approach leads to what we call an exception-based architecture.

The normal execution of a task by the robot can be described as the execution of a sequence of sub-tasks, each subdivided further until eventually the action level is reached. Each (sub)task attempts to realize a certain (sub)goal state; each task can only be executed if certain initial conditions are met. This subdivision is described in the task-plan.

If an attempt is made to generate the entire plan-tree, with all possible alternatives, beforehand, we speak of "strong binding". Strong binding has the advantage that no on-line re-planning is required, and that the system can immediately select the next subtask to execute. In situations where each action can result in a number of different outcomes (including undesirable outcomes), strong binding will lead to an explosive growth of the plan tree and the effort to generate it.

At the other extreme we find "weak binding", where execution is started before a complete plan for the task has been generated and where the next subtask is planned on-line on basis of the outcome of its predecessor. From the point of view of the planner, this has the advantage that only a small part of the task-tree needs to be generated. Branches that are never needed, need not be generated. However, in complex situations and in situations requiring rapid action, on-line planning may be too time-consuming.

Intermediate degrees of binding can also be used, where a plan is pre-generated for the normal, desired outcome of each action and for a very limited number of undesirable outcomes or exceptions. Such pre-planned exception handling is efficient in the following situations:

- 1 The exception has a high probability of actually occurring.
- 2 The exception may have a low probability, but it requires a prompt and well-defined response.

Situations for which no a-priori plan is available will cause an exception to be raised at the level of the action dispatcher. This leads to the second way in which non-nominal situations may be handled, viz. by plan extension.

Plan extension requires an on-line task-planner/exception handler to be available. This module will be activated by the action dispatcher when the available partial plan tree does not contain a suitable follow-up action to achieve the current (sub)goal from the current state.

### **3. Intermediate level control concepts**

At the intermediate control level - the level of the elementary operations - the foundations of the required flexibility are laid. For a robot to be able to function under various circumstances in a changing environment, operations are required that can be used in each of the expected circumstances.

The usefulness of the system is greatly enhanced by providing powerful elementary operations that can handle various non-nominal conditions by itself. An example is collision avoidance. Though it is possible in principle to return to the higher control levels every time an unexpected obstacle is encountered, it is better to handle collision avoidance at a low level, where it can be done faster and with less overhead.

When a (partially) valid model of the environment is available, operations can be described, planned and executed most effectively by referencing to this model. If, however, no model is available, or the available model proves to be (locally) invalid, sensor based control becomes essential. Also, in situations where the relationship between the internal state information of the robot and the external model is lost, as may happen for mobile robots when the dead-reckoning becomes inaccurate, sensor-based control is needed to re-establish this relationship.

Therefore, we have included both model-based and sensor-based control elementary operations.

## 4. Functional and operational architectures

In defining the architecture for an autonomous system, such as the MARIE vehicle, we must first describe what the system should accomplish and what functionalities are required to accomplish those tasks, leading to a *functional architecture*, e.g. similar to the NASREM architecture.

Next we should describe the way in which each task should be accomplished, the components required, and the interaction between those components, leading to an *operational architecture*. An example is Brook's subsumption architecture [6].

Lastly, we must describe how we actually implement the operational architecture, what interfaces should be used for data exchange and task activation, resulting in the implementation architecture.

The basic characteristics of our approach are hybrid control and the application of the virtual robot concept [1], [2]. In some ways our architecture shows some resemblances with the architecture as presented by Crowley [7].

We propose a hybrid controller employing plan driven and event driven mission execution. Plan driven execution strongly relies on model based prediction and planning. It allows a robot controller to follow closely an a-priori defined plan and deals with exceptions with the objective to disrupt the a-priori plan as little as possible.

When exceptions are more numerous (decreasing environment structure), plan driven execution leads to inefficient execution. Event driven execution for reflective action, provides the necessary reactive capabilities to switch between alternative execution sequences of pre-compiled task achieving functions.

We also propose the application of a virtual robot concept [1], [8] to provide a structured approach to the creation of low level, robust, self correcting sensor-actuator feedback loops, which provide basic manipulation and mobility functions to higher level task oriented control functions. It allows to specify a task level user interface by hiding lower level execution details. Also the other way round, changing in the lower level control system does not affect the task level specification. This allows to develop task level interfaces in parallel to a further development of low level system functions.

One of the essential properties of our architecture is that at various levels within the control system alternate modules can be activated, depending on the task the vehicle has to perform. In the figures illustrating the architecture, the sensor-based wall-following mode for the MARIE vehicle has been selected as the example.

### 4.1. The functional architecture

The functional architecture adopted for the vehicle is illustrated in Figure 1. It shows the layering of the system and the flow of sensor and control information, but does not show the flow of "event information" and the ways of activating the various modules.

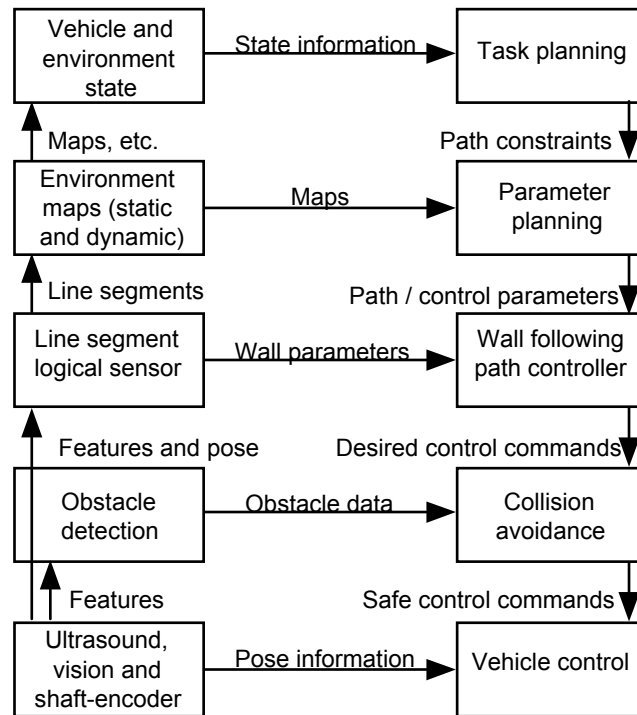


Figure 1. The functional architecture for the vehicle

The architecture has a regular layered structure, with at each level, a logical sensor, data, and a control module. It is a fairly standard hierarchical control architecture, except possibly for the location of the collision-avoidance module. In many architectures, collision avoidance is accomplished via explicit path-planning around (observed) obstacles in a local map, i.e. at a higher level in the system. In our architecture, collision avoidance is reactive in nature, i.e. low-level. The approach to collision avoidance developed for the MARIE vehicle is described in [9].

#### 4.2. The operational architecture

The operational architecture is illustrated in Figure 2. In addition to the more or less static flow of sensing and control information of the functional architecture, the operational architecture also addresses the dynamics of the changes in control mode. The principal mechanisms responsible for this are the monitoring tasks depicted in the right-hand column, and the action-dispatcher.

At one and the same place in the architecture, more than one control or sensor module may be available. A specific choice of a combination of these modules results in a specific action of the robot.

Each of the action will be valid only within a certain control domain. Sensors used for monitoring tasks will signal when the boundaries of such a domain are crossed and thus raise an exception. Such an exception will result in a change of the control mode for the vehicle initiated by a higher level of control than the one causing the exception.

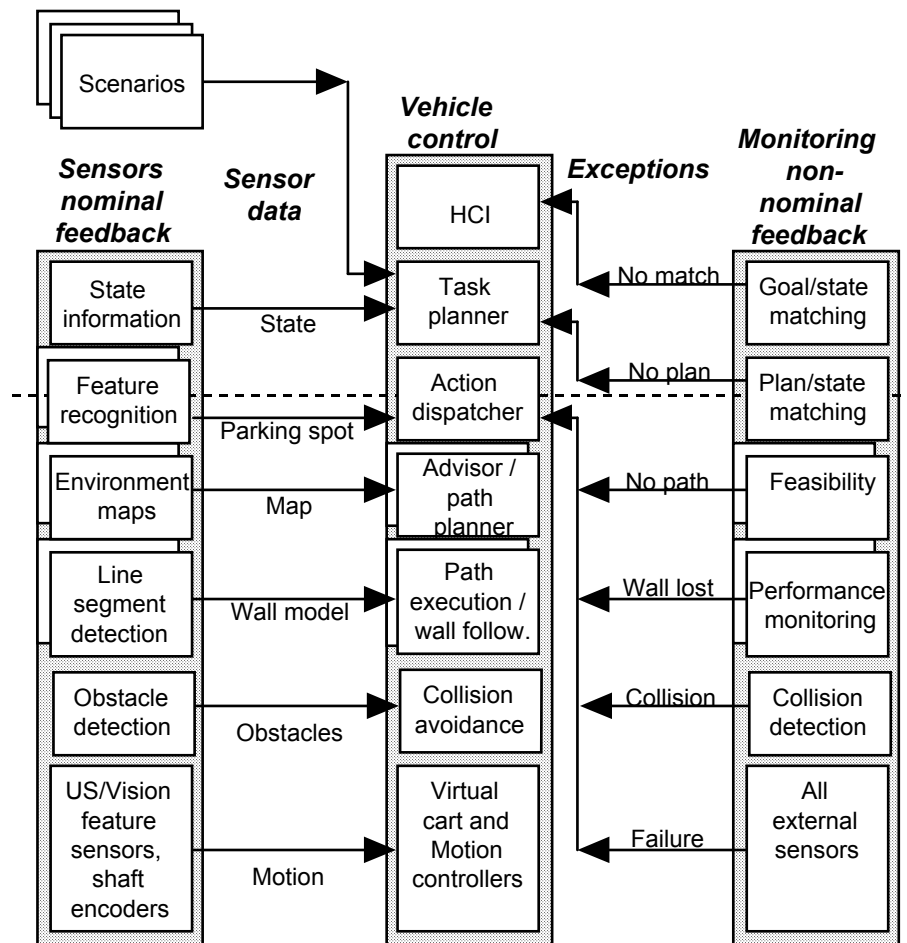


Figure 2 - The operational architecture for the MARIE vehicle. The horizontal dashed line at the action-dispatcher level indicates the transition from low-level to high-level control.

The combination of active modules shown in the figure could be used in a parking-spot search task. It will allow the vehicle to follow a wall (or a line of parked vehicles), while avoiding collisions with any obstacles. The wall following task will be broken off when the feature recognition task signals the action dispatcher that it has found a suitable parking spot. Simultaneously a path planner is active, e.g. generating a path to a subsequent search location in case no parking spot is found.

It will be clear that a number of essential relationships may exist between various modules. The "wall follower" action requires wall-data from a suitable wall sensor as well as the control operation "wall follower". This control operation is just one example of the possible control operations. For each such operation in the system a number of support modules may be needed:

1. Modules monitoring that the system remains in the domain for which the method of control is valid.
2. Modules with knowledge about how to use the control operation (e.g. path planners).
3. Modules with general knowledge about when to use the control operation. This knowledge will be pre-stored in the scenarios.

When, in a given situation more than one operation would be a suitable choice, the expected performance of each has to be evaluated in order to select the most appropriate operation. Therefore, we need

4. Modules with knowledge of the performance of the control operation in a given situation, e.g. based on simulation (advisors). Though the task of these modules is clearly distinct from that of the path planners, they are sufficiently closely related to combine the two.

## 5. Implementation architecture for the vehicle

The architecture described above has been successfully implemented and tested (albeit without an on-line task-planner) on the vehicle and arm testbeds developed within the Marie project.

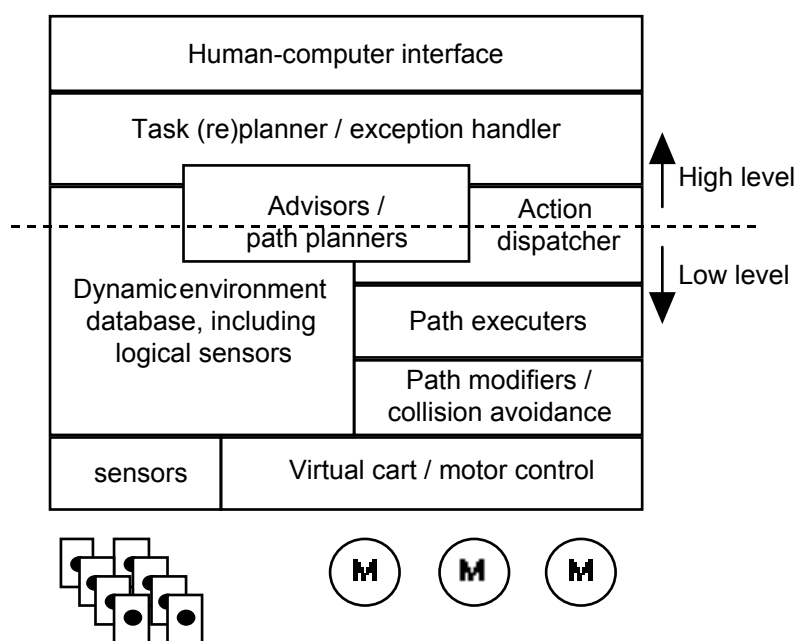


Figure 3 - an overview of the implementation architecture

Here we will discuss the implementation on the vehicle testbeds. Because an on-line planner/replanner has not yet been realized for the vehicle, only a strong-binding approach to task-planning could be tested.

Figure 3 illustrates the classes of modules in the implementation architecture. In subsequent paragraphs we will further clarify the details of this architecture and show the location of the interfaces.

This illustration is primarily intended to show the grouping of modules relevant for the control and monitoring of the vehicle. Modules in the same group generally share various properties and interfacing requirements. Especially the modules in the dynamic environment database form a complex hierarchy.

The indicated separation between low-level and high-level modules is a distinction in various module properties:

1. Low-level modules are generally time-controlled modules. Once they have been activated, most are controlled by a timed loop. High-level modules are generally event-based. When they are activated, they run until a specified result is produced, activate other modules and wait for the next event.
2. Low level modules deal mostly with continuous, numerical quantities; high-level modules with discrete state variables, i.e. symbolic information.

There is a strong trend from high-frequency, hard real-time to low frequency with relaxed timing requirements as one goes from the lowest to the highest level in the architecture.

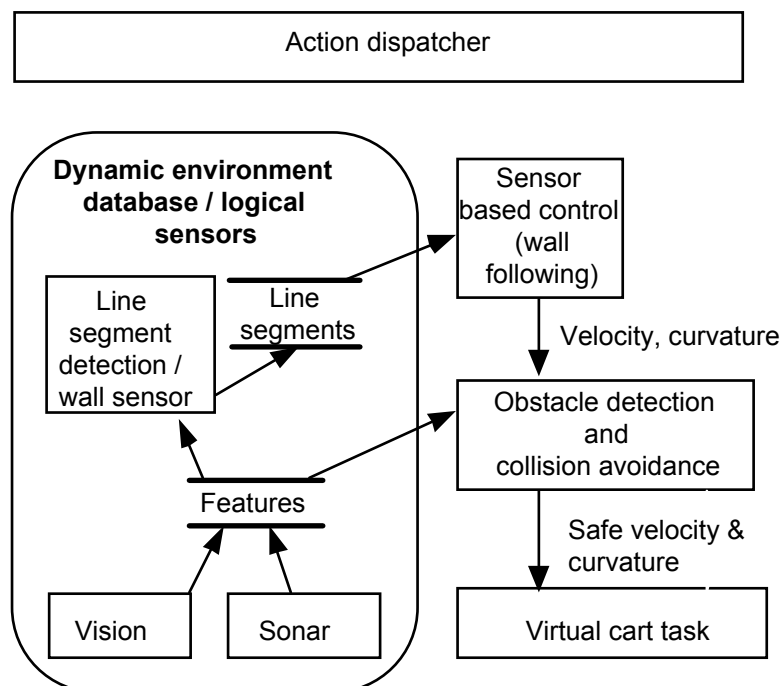


Figure 4 - the low- and intermediate-level communication structure for the vehicle.

In Figure 4 the *information exchange* at the lower and intermediate control levels of the vehicle is illustrated.

Two different types of information exchange occur - direct exchange between tasks, as is the case in the vehicle control column at the right - and indirect exchange via data-managers.

Data-managers implement the data-storage for the dynamic environment database, very similar to blackboards. Client tasks can query the data-manager for new data in a specified class, with the option to wait. In this way a data-flow network of logical-sensors can be implemented.

Data-managers allow that the source and the use of data can be decoupled very easily and consequently monitoring becomes very straightforward, also from a remote host. A drawback is that extra delays are introduced in processing the data.

The action dispatcher does not directly consult the information generated by the lower levels - but it uses the much more abstract status information provided by the various elementary operations upon completion.

Figure 5 shows the *activation* structure for some typical intermediate and low level control tasks for the vehicle. Downward arrows show activation commands (generally accompanied by some control information); upward arrows show task completion and exception signalling, accompanied by a certain amount of status information.

The modules shown present conceptual tasks in the system - things that must be done to make the vehicle run. Often, they are implemented as separate tasks in the operating system sense as well, but not always. E.g. the obstacle detection and collision avoidance module is implemented as a set of subroutines to be called by the path controller.

For clarity, not all relevant tasks have been included, nor have all possible signals. E.g., the virtual cart [9] will directly inform the path controller (either the model based or the sensor based controller) when a collision has occurred.

All of the tasks that are activated directly from the action dispatcher are provided with a standard interface - the elementary operation interface - that provides the required activation/suspension and signalling capability in a uniform and simple-to-use manner. The same interface can also be used if a task is to be activated by any other task.

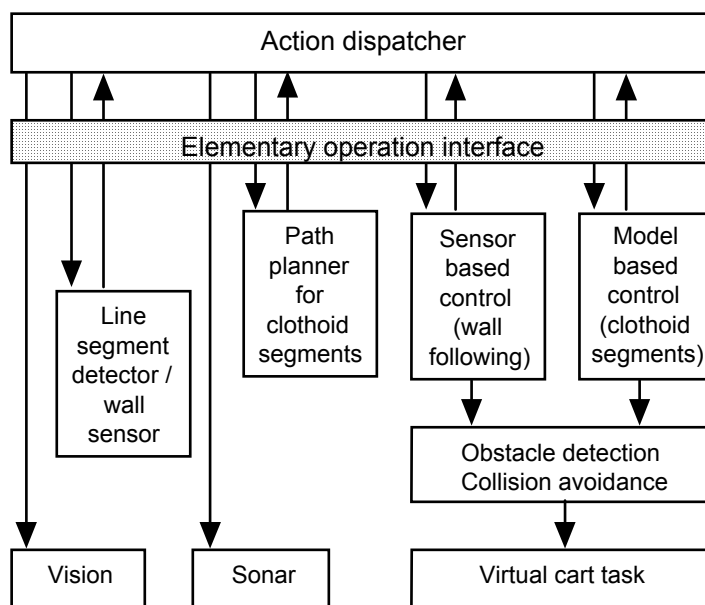


Figure 5 - the low- and intermediate level activation structure for the vehicle

The elementary operation interface does not provide for the exchange of large amounts of data. For that purpose tasks must make use of data-managers.

Figure 6 schematically shows the communication between the various types of module involved in the plan generation and plan execution.

The action dispatcher and the path planner are already familiar from the previous diagrams; the latter has an additional task in this context, viz. the

prediction of the performance of the system when a certain course of action is taken. Useful criteria are e.g. attainable position accuracy and travel time. The evaluation of such criteria should allow the task planner to choose between model based control and sensor based control in situations where both are possible.

The task planner will attempt to produce a path in the state space of the vehicle from the current state to a goal state. At the highest level, that goal state will be provided by the user.

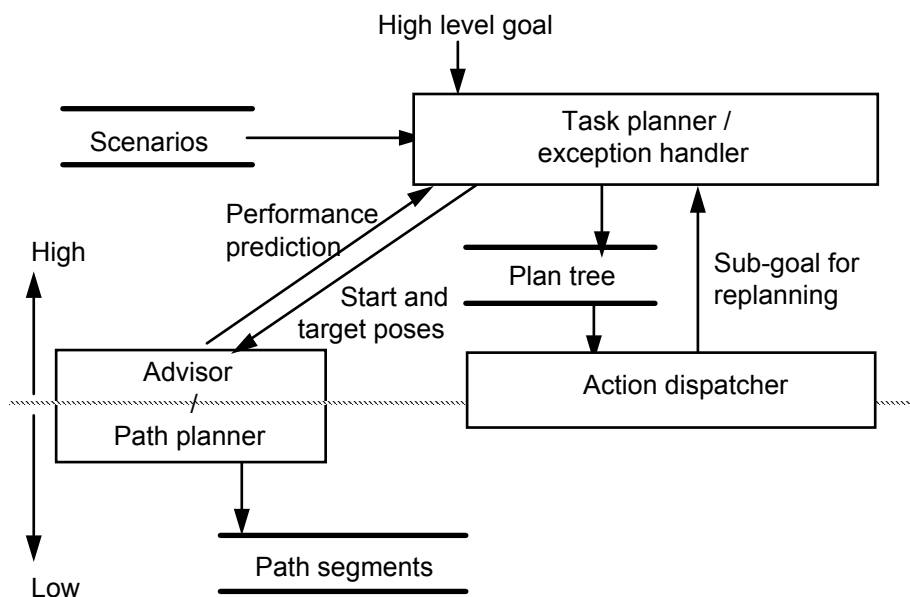


Figure 6 - the high level communication and activation structures for the vehicle

The required knowledge of the possible state transitions, and of the low level modules that must be activated to attain such a transition, is contained in the scenario data-base. Scenarios essentially are subtask descriptions, with specified initial conditions and an expected final state. In general, numerical parameters still need to be specified.

The interactive planner used for the current implementation allows for the use of scenarios in much the same way: scenarios provide partial plans for, accomplishing certain goals for the system, thus facilitating testing of the system and the construction of complex task-trees.

## 6. Conclusions

The functional and operational architectures described in this paper have been implemented, albeit in a different way, on two (semi)autonomous robot systems. Experiments with those systems show that flexible systems have been obtained that can respond in an appropriate manner to unexpected situations. The architectures provide uniform and well-defined interfaces, making modification of existing modules and addition of new modules simple and straightforward.

Yet, not all problems have been solved to complete satisfaction.

The most important of these results from the mixing of model-based and sensor-based control concepts. This leads to selection problems in the planning phase. A human programmer/operator can judge reasonably well when the one type of control is to be preferred over the other. For an automatic planner, this is much more difficult. A mechanism to assist an on-line planner has been defined in the advisor modules that can simulate and predict the systems behaviour, but this concept still requires testing.

## 7. References

- [1] Meijer, G.R. Autonomous Shopfloor Systems; A study into exception handling for robot control. PhD. Thesis, Amsterdam, June 1991.
- [2] Meijer, G.R. and L.O. Hertzberger. Plan- and event-driven architectures in task level control. In Proceedings of Fachgespräch Mobile Autonome Systeme, Karlsruhe BRD, Edited by Rembold, U., Levi, P. and R. Dillmann, 1991.
- [3] T. Henderson, E. Shilcrat, "Logical Sensor Systems", In Journal of Robotics Systems 1(2) 169-193, 1984.
- [4] Weller, G.A., F.C.A. Groen, and L.O. Hertzberger. A sensor processing model incorporating error detection and recovery. In Proceedings of the NATO International Advanced Research Workshop on Traditional and Non-traditional Sensing, Edited by T. C. Henderson, 351, NATO-ASI Series F63, Springer-Verlag, Berlin Heidelberg, 1990.
- [5] M. Mergel (ed), "Methodologies and Techniques for Sensing and Data Fusion", Milestone V, Deliverable DV3 (Public MARIE deliverable in preparation).
- [6] Brooks, R.A. A robust layered control system for a mobile robot. In IEEE Journal of robotics and automation, RA-2, 1986.
- [7] J.L. Crowley, O. Causse, P. Reignier, "Layers of Control In Autonomous Navigation", Workshop on Mobile Robotics for Civil Works, Esprit, Brussels June 1991.
- [8] A.S. Tanenbaum, "Structured Computer Organization", Prentice-Hall International, 1990.
- [9] G.A.den Boer, et al, "The MARIE Autonomous Mobile Robot", International Conference on Intelligent Autonomous Systems: IAS-3, 15-19 February, Pittsburgh PA, USA.