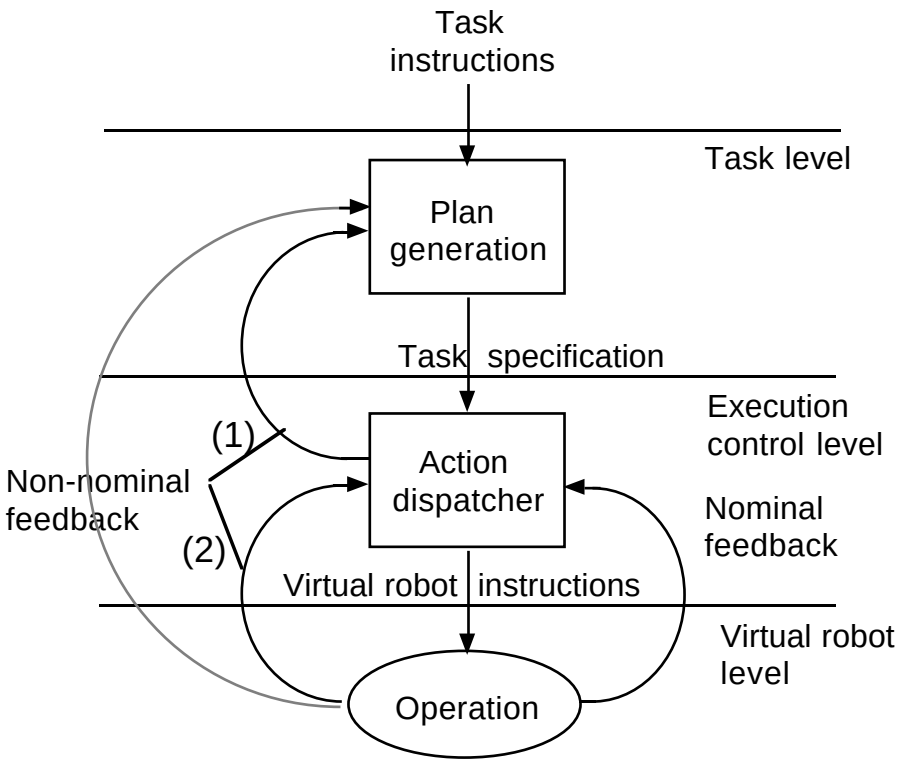
Task
instructions
Task level
Plan
generation
Task specification
Execution
control level
(1)
Non-nominal
feedback
(2)
Action
dispatcher
Nominal
feedback
Virtual robot instructions
Virtual robot
level
Operation

INFORMATION ARCHITECTURE CONCEPTS FOR AUTONOMOUS CONTROL[1]

L.O. Hertzberger[1], G.D. van Albada[1], G.A. den Boer[2]

[1] *Department of Computer Systems*
University of Amsterdam
Kruislaan 403
1098 SJ Amsterdam

[2] *CMG Informatietechniek B.V.*
Laan van Kronenburg 14
1180 AK Amstelveen

Abstract. In this article concepts of an information architecture for complex autonomous systems will be discussed, that should be considered when autonomous control systems have to be designed. These concept are: architectural abstraction, virtual machine layering and logical sensoring. They are based on experience gained with various robotics and CIM projects. They will be illustrated using our MARIE mobile robot architecture as example. In our opinion these concepts are more widely applicable than to autonomous control systems only.

Keywords: Autonomy, system architecture, system design, exception handling, virtual machine layering

INTRODUCTION

Information systems for autonomous control systems are part of a general class of systems that can be qualified as complex information systems. The typical characteristic of this type of system is that the sensor-actuator control loop plays a dominant role in the information architecture. Not only does this loop put a heavy demand on real-time interaction, it is absolutely essential for the system capabilities. The task of the information control architecture is not only to make it possible to extract all that information, but also to create a flexible system in which all the foreseen autonomous system functionalities, demanding decision support capabilities are realised. To that extent modern systems distinguish different levels of abstraction, such as the mission, task, action, or elementary operation levels. The purpose of this is to hide unnecessary details and only concentrate on aspects that are important to realise the capabilities of that level. Information hiding and data abstraction are the key issues when designing information system architectures including those for autonomous robotics.

In autonomous robotics it can be observed that the architecture and the design of such systems is more or less done on an ad hoc basis. This is not a great problem as long as toy systems or experimental platforms are involved, that have as a their main goal to experimentally verify some of the concepts of autonomous behaviour. Examples could be the verification of planning concepts, or logical sensor concepts, or concepts of behaviour, or neural network control, for instance, for road following. In all these situations it is sufficient only to illustrate the possibilities of the concepts under study and to abstract, as much as possible, from issues that are important for the rest of the system.

However, in situations where the emphasis is on the design and implementation of complete systems that have to operate in real world environments (like in air traffic or road traffic control, space or CIM systems) issues of complex information system design play a far more dominant role. Not only do these systems have a far longer life cycle in which modifications to the software system will become necessary, the design of the software of these systems will almost always be the result of a collaborative effort of team of people. This makes a more systematic approach to system architecture necessary. From that point of

[1] This paper was published in in "Intelligent Autonomous Systems - IAS-4", Proceedings of the International Conference, Karlsruhe, Germany, March 1995, eds. U. Rembold, R. Dillman, L.O. Hertzberger, T. Kanade, pp. 190-197, IOS Press, Amsterdam, Oxford, Washington DC (1995)

view the fundamental issue in those systems is that the various system components have to work properly as entities, while they also have to work together, in order to help fulfilling the system task as a whole. Therefore other (additional) aspects that partly have to do with the intended real world application (like redundancy, or software engineering aspects like interfaces including human interface etc.), play a far more important role than in toy, or purely experimental systems, that have a short life span.

ARCHITECTURE CONSIDERATIONS

In the research we undertook, which mainly concentrated on aspects of exception handling as one of the major properties for realising autonomous control, we concentrated on hybrid architectures combining symbolic techniques for decision making at the high level and numeric control techniques at the low level. Because the research activities comprised a large number of researchers, not all at the same location, it became necessary to define the architecture concepts already in an early phase of the project. However, because of the research orientation, it could be expected that substantial modifications would become necessary to add new functionalities in a later stage. Such new functionalities can have severe impact on the system architecture, particularly when modifications involve several different modules that influence each other. As a consequence a choice for a structured approach was made, allowing flexible modification when more functionality becomes necessary.

SYSTEM ARCHITECTURE ABSTRACTION

In our system architecture three levels of abstraction were distinguished, each represented by its respective model, viz.:

- Functional architecture model: concerns itself with the functional behaviour the system should exhibit,
- Operational architecture model: comprising the way in which the desired behaviour can be realised taking into account constraints on the system, as well as the environment in which it has to operate,
- Implementation architecture: comprising the way in which the system is realised. At this level a full implementation of hardware and software modules is realised.

The rest of this article will concentrate on the functional and operational architecture models. Given the fact that the most important design decisions will have to be taken at the operational level, this level will be discussed on most detail.

FUNCTIONAL ARCHITECTURE MODEL

In the functional model the emphasis is on the functionality the system should posses. It has to be decided at this level which type of features the system should posses in order realise its overall functionality. This level describes in no way how these features are realised. Given the example of the MARIE autonomous robot, Den Boer et al. (1993a, 1993b, 1995) it was decided at this level that the system should have three functionalities; perception, reasoning and control. The computations involved in the reasoning process were decided to be mainly of symbolic nature, the computations in control would be based on numerical techniques. Perception would apply numeric techniques for low level sensor processing and symbolic techniques when providing state information for path planning, exception handling and collision avoidance.

The part of the system that performs the reasoning process will be defined as the task level or *plan generation* system, whereas the perception and control system responsible for the actions of the system will be defined as *plan execution* system. The interaction with the plan generation system is at the task level. Typically, a human operator will give a task instruction to the plan generation system. Such an instruction is inherently incomplete, i.e. information is only partly specified or even invalid. The reasoning process has to decompose the task into a set of actions that will have to be executed in order to achieve the desired state change. During the execution of these actions exceptions will have to be coped with. An exception is defined as an unexpected occurrence, being the difference between what was defined during plan generation and the actual situation observed.

Because the task definitions the human operator presents to the system are inherently incomplete, several alternative ways of execution, depending on the state of the environment, will have to be considered. For this

reason and to cope with exceptions, the plan generation system in our situation comprises task planning and scheduling, as well as parameter planning. The objective of the planning is to define all required activities necessary to complete a given task. Run-time control over the order in which activities are executed is important in order to realise the flexibility required to operate in an unstructured environment. Therefore a task scheduling activity has to be defined. The end situation should always be a series of activities that can be executed in order to achieve the desired goals of the task presented to the system.

OPERATIONAL ARCHITECTURE MODEL

Most of the system design decisions for the robot architecture have to be made at the operational level. At this level not only the system capabilities are defined, but important environment constraints also have to be dealt with. For instance, it makes a large difference whether a robot system, be it an arm or a vehicle, has to operate in a structured environment where planning is straightforward, or whether the environment is unstructured and obstacles and exceptions are likely to occur. In the last situation perception is indispensable to inform the robot system about the state changes in the environment. In discussing the operational architecture model, the last situation is presumed. As was already observed, the mechanism that was designed to cope with unpredicted changes in the environment is the exception handling model.

However, before going in detail about the architecture of the operational model, some more general architectural concepts that we have applied in our work will be discussed. We have adapted a method of abstraction and information hiding stemming from computer architecture. This method is called the layered virtual machine approach. Analogously, we have defined the layered virtual robot approach.

The Virtual Machine and Virtual Robot Methodology

The virtual or abstract machine concepts stem from computer architecture research. A virtual machine layer Tanenbaum (1990) is defined by an instruction set, and a virtual machine that can interpret and execute this instruction set. The instruction set fully defines the capabilities of that particular layer. An N+1 level virtual machine will express an instruction that it wants to accomplish into a sequence of instructions of level N. The lower level will then interpret each of those instructions, one at a time, and produce a number of instructions for level N-1. A software system is built by placing several layers on top of each other, each having more complex instructions at its disposal. A human operator could communicate with the system through an interface often connected to the highest level, using the instruction set of that level. As was observed, a high level instruction is interpreted by several lower level interpreters, one instruction at a time. The lowest level of the instructions are executed by the computer hardware. However, for reasons of increase in execution speed, straight compilation of a high level machine towards machine level instructions is often realised.

By going up in the hierarchy of layers information hiding and data abstraction are applied. A virtual machine is capable of executing its instructions without further assistance of the higher levels. This approach makes it possible to hide system dependent details from the higher levels. The virtual machine concepts allows the programmer to concentrate on the problems in his particular virtual machine layer without being distracted by implications at other layers. This makes it possible to break complex problems into smaller problems by applying the principles of virtual machine layering, abstraction and information hiding. Thus we can impose order, clarity, consistency and tractability in a multitude of interacting system components.

A straightforward translation of these techniques to autonomous control results in the model of a stack of control levels, each represented by a language or a program representing a certain functionality in a particular language. The bottom layer is that of the robot controller's electronics (or general purpose processor with I/O controller), taking data from sensors and driving the robot actuators. The immediate level above corresponds to the interpreter of the robot controller[1]. The

[1] Sometimes, a level is internally divided up into sub-levels. E.g. at the lowest level one may find the drivers for individual actuators; at the next highest level, coherent robot control is realised by using all actuators in concert.

language of this level is either a special purpose robot language supplied by the robot controller manufacturer, or a general purpose programming language augmented with robot motion instructions.

The power of the method is that the implementation details of one level are not visible for the user of the higher levels. Therefore, the execution of robot programs by the robot controller or a general purpose computer can be hidden at a sufficiently high level.

However, in our research, we have gone one step further in the application of the virtual machine concepts towards autonomous robotics. The power of the methodology allows one to define several high level application specific programs, each specifying an application specific instruction level with its application characteristic instruction set. By applying the methodology of virtual machine layering in robotics the higher level instruction sets are, from robot programming point of view, more powerful and specify different levels (such as planning, task, execution control etc.) that can be used to deal with robotics specific problems. The lowest level of this hierarchy will specify the robot basic (lowest) sensor as well as actuator capabilities. For this specification a high level programming language (like C or FORTRAN) can be used that is executed either through compilation or interpretation on the robot controller hardware.

Fig. 1 presents a simplified illustration of our autonomous robotics specific control model illustrating two autonomous robotics specific layers, the task level and virtual robot level.

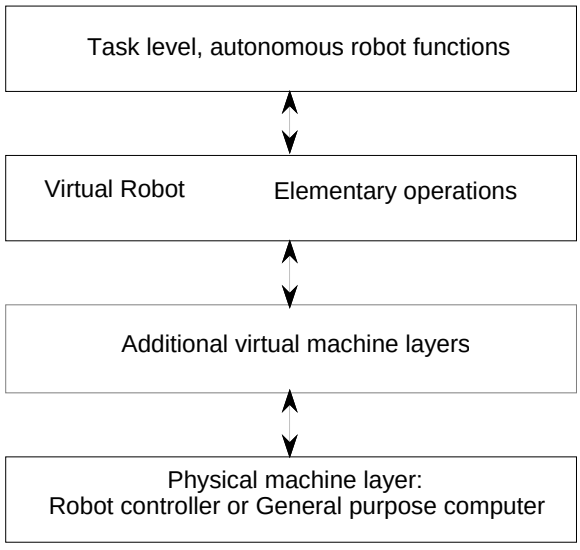

Fig. 1. The virtual robot and task level of the autonomous robot controller.

Let us assume that the task level of a robotics system has to be realised in order to study autonomy. To define the architecture at this level, the next lower level has to be defined. This is called the virtual robot level and represents a virtual machine capable of executing instructions for the robot defined at the highest (task) level. It hides the implementation details from the task level user.

The virtual robot is modelled with the concept of elementary operations analogous to elementary or assembly instructions of the virtual computer architecture. The Elementary Operations (EO's) are the generalisation of all available actions of a robot and its sensors. An action of a robot is always a combination of some actuator motions, some sensor readings and a particular feedback law. The feedback law specifies how the sensor readings are used together with the information on the state to be reached, to compute the steering signals for the actuator. The elementary operations are instructions which define the manipulation and sensing capacities of the virtual robot. The set of all elementary operations of a robot describes all of the robot's capabilities.

To fulfil a task or to reach the objectives of a mission, the robot has to carry out a sequence of elementary operations. A task of a robot can be a complex mission or it can be very simple and be accomplished by one or two elementary operations of the robot. The actions of the robot describe how the task should be carried out, based on the explicit task description. For one implicit task, there might be different explicit task descriptions possible. The important factors are:

- the capabilities of the robot which carries out the task, and
- the external conditions under which the task must be completed.

The capabilities are described by the EO's of the robot. The combination of a task instruction and the external conditions determine how the task is going to be expressed in the explicit EO's. The task and the external conditions are, therefore, the *meta instruction* of the task level control. A meta instruction defines what has to be accomplished, and implicitly specifies how this will be done.

The task level interprets each task instruction and expresses it in terms of the explicit EO's of the virtual robot. Depending on the nature of the task to be executed, the execution order these instructions need not be fully specified. It can, e.g. be expressed in terms of a precedence graph, allowing the run-time system to select the most appropriate ordering. The virtual robot executes the EO's according to a certain sequence of sub tasks in the graph by interpretation. If all the EO's are interpreted by the virtual robot, the task instruction is completed. At the task level, the next task instruction can be selected for execution. Fig. 2 shows a snapshot of the flow of execution between the task level and the virtual robot level.

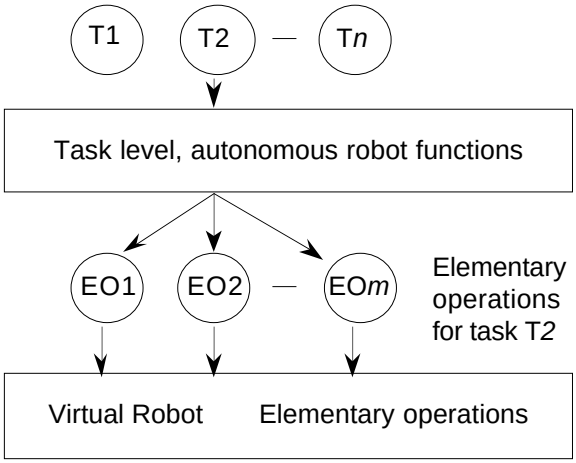

Fig. 2. Flow of execution between task level and virtual robot.

In our mobile robot control architecture, the task level consists of the task tree in a symbolic language. If only one possible execution flow is possible, the execution model of the task tree follows a depth first search strategy. However, we are interested in the freedom of execution of the task decomposition when there are more ways to execute a sub task. In such a case, the execution flow of the sub tasks not only follows the depth first search, but includes disjunctive elements (OR connections) of alternative execution sequences as well.

Historically autonomous robotics architectures use hierarchical decomposition or abstract layering for decomposing a robot system into more manageable units. Virtual machine layering is, however, not the same as abstract layering, although they both work on more abstracted data as the level increases. The difference is that with abstract layering obtained via hierarchical decomposition the higher levels always have to be in control of the lower ones. Virtual machine layering does not impose such a restriction. It defines instruction sets, not processing hierarchies. In addition , as we have seen, virtual machine layering provides an executable specification of abstract layering when that is required.

We have applied the virtual machine concepts to the operational as well as to the implementation architecture models of our MARIE mobile robot architecture (Den Boer, 1993b, 1995). As an illustration of the power of this methodology the operational architecture model will be presented.

MARIE OPERATIONAL ARCHITECTURE MODEL

Within the MARIE operational model, which is illustrated in fig. 3, the methods and techniques for accomplishing the autonomous behaviour have to be defined. The description of the various levels is presented at the left hand side in this figure, the corresponding virtual machines on the right hand side. In brackets the type of (virtual) machine instructions are given. The actual implementation level is not touched upon at this point. Instead, operational constraints that could have consequences for the methods and techniques by which the actual problems are solved are taken into consideration. For the MARIE research the most important operational aspect was the capability to handle exceptions for the purpose of realising autonomous behaviour. Therefore, exception handling had to be an integral part of the system architecture. The consequences for the design of the operational model were that plan generation as well as plan execution must support the incorporation of exception handling in their respective task trees. The consequences of this approach for the task generation level are that alternative series of activities have to be added tot he task tree. Exception handling further has a considerable impact on the various levels of the plan execution system, and in particular on the execution control and virtual robot levels.

From fig. 3 it can be observed that the plan generation system comprises only one layer, whereas the plan execution layer comprises three levels; the execution control, virtual robot level and the interface level. At the bottom the actual hardware can be distinguished which is shielded by an interface layer to impose further hardware independence.