LUND UNIVERSITY

Integrated Architecture for Industrial Robot Programming and Control

Nilsson, Klas; Johansson, Rolf

Published in:
Robotics and Autonomous Systems

DOI:
10.1016/S0921-8890(99)00056-1

1999

Link to publication

Citation for published version (APA):
Nilsson, K., & Johansson, R. (1999). Integrated Architecture for Industrial Robot Programming and Control. *Robotics and Autonomous Systems*, *29*(4), 205-226. https://doi.org/10.1016/S0921-8890(99)00056-1

Total number of authors:
2

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

ELSEVIER

Robotics and Autonomous Systems 29 (1999) 205–226

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Integrated architecture for industrial robot programming and control

Klas Nilsson [a,*], Rolf Johansson [b]

[a] *Department of Computer Science, Lund Institute of Technology, Lund University, PO Box 118, S-221 00 Lund, Sweden*
[b] *Department of Automatic Control, Lund Institute of Technology, Lund University, PO Box 118, S-221 00 Lund, Sweden*

Received 31 October 1997; received in revised form 20 October 1998; accepted 20 May 1999
Communicated by F.C.A. Groen

Abstract

As robot control systems are traditionally closed, it is difficult to add supplementary intelligence. Accordingly, as based on a new notion of *user views*, a layered system architecture is proposed. Bearing in mind such industrial demands as computing efficiency and simple factory-floor operation, the control layers are parameterized by means of functional operators consisting of pieces of compiled code that can be passed as parameters between the layers. The required interplay between application-specific programs and built-in motion control is thereby efficiently accomplished. The results from experimental evaluation and several case studies suggest the architecture to be very useful also in an industrial context. ©1999 Elsevier Science B.V. All rights reserved.

Keywords: Robot control architecture; Autonomous industrial robots

1. Introduction

The term advanced robot systems is commonly used, but with a wide variety of different meanings depending on the field of application. Consider, for example, robots used for manufacturing purposes in well-defined environments, or autonomous robots operating in poorly defined environments. In the former case, the difficulty is to make the robot flexible and easy to program for new applications in combination with the high-level performance demands of robot productivity. In the latter case, the control problems involve coordination tasks with active sensing, perception, reasoning, and actuation in a time-specific manner. Specifically, all kinds of control require support for the introduction of tight software couplings of sensing and actuation [42].

As currently available industrial robot control systems generally fail to provide the flexible and powerful environment that permits complex task-level programming as well as improved autonomy and intelligence to be added [19], we need an appropriate architecture that provides a framework for robot control. Here we will mainly be considering robots for manufacturing purposes, while keeping more intelligent robots in mind. Many of the available high-level control concepts/architectures should then be possible to superimpose on our foundation.

Inherent in the present and particularly the future increased capabilities of robots is the problem of complexity. The purpose of many current research projects

* Corresponding author. Tel.: +46-46-222-4304; fax: +46-46-131021.
E-mail addresses: klas.nilsson@cs.lth.se (K. Nilsson), rolf.johansson@control.lth.se (R. Johansson)

0921-8890/99/$ – see front matter ©1999 Elsevier Science B.V. All rights reserved.
PII: S0921-8890(99)00056-1

is to cope with this complexity by defining a control system architecture. Moreover, it is a common goal that such an architecture should also promote modularity and reuse of control and software components. In the field of industrial robot control, robot autonomics is currently attracting keen research interest. Neglecting or overlooking the importance of advanced robots for manufacturing, however, causes problems such as:

- The need to redevelop manipulator control subsystems for adaptability to the intelligent control system. Ability to utilize industrial robot controllers would reduce development time and cost by sharing the large user base of extensively tested modules.
- With a pure top-down or functional decomposition approach to the design of intelligent robot control systems, "the community will be trapped, forever making interesting, expensive and unique creations which never see widespread application" [23].

In this paper, we propose an architecture, architectural principles, and a bottom-up approach that should serve as a basis and a complement to other architectural approaches within intelligent robot control. This is approached in the following way:

- A review and classification of architectures found in the literature will be presented in this section.
- Section 2 treats architectures and abstractions used for so-called intelligent robots with particular attention to robots operating in relatively well-defined environments as in manufacturing.
- The architecture we propose is presented in Section 3, where a concept of *user views* will also be introduced.
- Section 4 contains an evaluation, and Sections 5 and 6 contain discussion and conclusions.

1.1. Role of software architectures

There is no possibility of the system designer being able to foresee every application demand. This is particularly manifest in motion control, where its properties must be amenable to modification in advanced applications. Moreover, to provide sufficient scope for task-level programming, it is desirable to preclude unnecessary interference or obstruction deriving from software architectures, a problem that has been overlooked within robotics research.

The "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)" [3] is one of the earliest and best known (and criticized) architectures for robots. One of its greatest disadvantages is "its use of a large global memory which violates basic software engineering principles of encapsulation and information hiding" [42]. A number of problems are brought to light by a closer scrutiny from two points of view — robots acting in *uncontrolled environments* and *intelligent industrial robots*. However, as such analysis is predicated upon certain basic distinctions as to functionality, we must first consider some fundamental concepts.

1.2. Fundamental concepts

In this context, the term architecture may generally be defined as denoting structure or style of structure with various — sometimes not very precise — meanings. For the purpose of industrial robot control systems, and for this paper, architecture is defined as denoting *the concepts and techniques that characterize the structure of the system*. The approach adopted here (Section 3) is to base the architecture on the usage of the system rather than on the internal design. The following list is an attempt to classify different architectures.

- *Hardware*. When the computing power of microprocessors was still the major limiting factor for advanced robot control, the main purpose of an architecture was to define a hardware structure that provided the required real-time computing power. Descriptions of several such architectures have been published [5,6,35,62,63]. Though the hardware structure is still important, experimental systems now need to be modular in such a way that a variety of interfaces, sensors, special computers, etc. can easily be added or replaced. Low cost is also important.
- *Control*. The manipulator dynamic control problem is a challenging problem that has inspired many research efforts [16,31,32,44,60]. The scheme solving the control problem can be seen as an architecture as it may involve many interacting control modules. From a control theoretical point of view, other aspects of the system design are often considered to be merely a matter of implementation. More

experiment-oriented research, on the other hand, often defines the control and the physical architecture jointly [4,51,63], whereas other researchers explicitly address control architecture [33].

- *Task specification*. Convenient task description is another source for control system architectures and end-user robot programming. Solutions range from explicit manipulator programming [59] to systems where the system automatically generates or links robot programs although no general purpose system has yet been developed. However, systems have been designed for supporting some particular aspects of task-level programming [37,64], or for specific applications [10,29,36,56], or using special algorithmic concepts [18,28,52].
- *Abstractions*. Robots operating in an unstructured and largely undefined environment must make excessive use of external sensors, and they must perform dynamic world modeling and real-time planning. Such systems are more complex than the control systems for robots in manufacturing. For the purposes of this paper we will focus on the issue:
  - *Can we find any, possibly new, abstraction that allows us to further enhance robot programming capabilities in typical industrial applications?*

  An architecture meticulously attuned to the manufacturing issues will not, of course, solve complexity problems encountered in conjunction with so-called intelligent robots working in uncontrolled environments. For such robots, the authors acknowledge that top-level architectures based on certain abstractions comprise an important issue. However, an additional aim here is that industrial robots should be useful as modules in fully autonomous systems, thereby promoting a bottom-up approach to make things work in practice. Furthermore, suitable abstractions are also important for task-level robot programming and autonomous operation in some manufacturing situations. Therefore, these questions have also been of great concern for the proposed techniques:
  - How should the industrial robot controller be organized to permit the use of industrial robots as modules in autonomous systems?
  - How can principles from intelligent robot control systems be beneficial also for robots used in manufacturing systems?

Hence, we need to take a closer look into the issue of abstractions.

2. Intelligent robots — preliminaries

We will now outline how different forms of abstractions have been used to define architectures for so-called intelligent robots. We will then see how these principles can be used for more conventional industrial robots, and how industrial robots can fit into so-called intelligent systems.

2.1. Abstraction beats complexity

The purpose of abstractions is to cope with complexity. When a complex system is divided into smaller manageable parts, those parts can be given a new simplified interface with aspects of the internal behavior omitted or simplified in some sense. We find this principle in organizations, industrial production (e.g., production cells), control theory (e.g., cascade control), and in software engineering (e.g., abstract data types). The abstraction then allows more powerful hierarchies to be built, using more abstract interfaces at higher levels of the hierarchy.

One way of using abstractions is to formulate pure science problems that are well suited to formal analysis. This is a popular approach within academic research, mainly because theory can be used to prove optimality which concludes the work (remaining practical aspects are left to the industrial user to solve). Though important — consider for instance problems ranging from system identification, adaptive control, minimal time/energy motions, and up to overall factory scheduling — solving these individual problems does not give us an optimal (or even useful) system. Therefore, here we are concerned with the overall system architecture. If properly designed, such architecture will then allow optimization techniques to be used as a means of improving performance.

An interesting question about any control system architecture is what set of abstractions or hierarchies it is built upon. Consider a multi-layered real-time control system for fully autonomous robots. Such a system will contain both hard real-time software and artificial intelligence (AI) related features (planning). The detailed implementation of such a system would of course make use of data abstraction and other soft-

ware paradigms, but some kind of high-level abstraction is needed to build hierarchies to cope with the complexity [40]. As pointed out by Schoppers, several such hierarchies have been proposed (see, e.g., [48] for further details and many references):

- *Frequency hierarchies* are based on the standard real-time principle that the real-time processes in a lower layer run more frequently than those in the next higher layer of the system.
- *Data abstraction hierarchies* are closely related to object-oriented programming. A lower software layer provides an abstract machine for the adjacent higher layer.
- *Representational abstraction hierarchies* are used within AI as a means of constructing an abstraction by suppressing or ignoring information.
- *Deresolution hierarchies* are often used in motion planning. Two layers are functionally capable of doing the same computations, but the degree of resolution is higher at the lower level. Deresolution is related to the above mentioned, previous hierarchies, but is not the same.
- *Subsystem hierarchies* are based on grouping the control of subsystems — e.g., control of individual joints — to control of the composed system — e.g., the arm driven by the joints. This approach is often combined with data abstraction.
- *Competence hierarchies* are built by combining simple behaviors of lower layers into more competent behaviors at a higher level of the system. For example, vibrations in a robot gripper caused by a simplified control can be utilized at a higher level for an advanced 'non-stiction' assembly operation.
- *Temporal extent hierarchies* are designed so that higher levels manage behaviors of longer duration. Actually, long time-horizon computations may require updating and re-computation more frequently than lower levels do.

Behavior abstraction is another abstraction which is an attempt to unify several of the abstractions listed. Although we need to bear these approaches in mind when designing advanced industrial controllers, further discussion of them here is beyond the scope of this paper. Instead, we want to emphasize the externally visible (to the end user, etc.) properties of the systems — i.e., those that affect the usefulness of the robot system in a normal manufacturing situation involving operators, production engineers, etc.

Software paradigms

Even more internal to the system are the implementation techniques of the actual control system, and the system software might be considered as merely a matter of implementation. However, we need open systems where the boundary between system programming and user programming will not be completely rigid. Furthermore, when developing an architecture, the implementation and software aspects become important.

Robot control systems are typically heterogeneous, both in terms of hardware and the required real-time properties [34]. One might then expect that there is no single software paradigm that is most suitable for implementation of the entire system. For parts of the system, declarative and formal methods could be used, such as the synchronous approach [8,21,57] which offers a uniform approach and some formal verification tools concerning the logical and temporal properties of the software. However, each software component has to be implemented in a traditional way using imperative languages such as C/C++ as most commonly used in industry.

There is no general agreement on the criteria for a 'good' paradigm. We may strive for code reuse, efficiency, maintainability, ensured correctness, or programs that are easily understood. The most appropriate language or paradigm may depend on system level, application, cost-effectiveness demands, actual hardware, etc. In fact, hybrid techniques appear to be most appropriate. Our approach has been to use or combine whatever paradigm that suits the actual situation. Currently we are using C, C++, Modula-2, and Java for system programming, as well as vendor or application-specific languages at an end-user level. Our architecture does not prescribe any particular software paradigm, but object-oriented programming appears to be most convenient to use for the core design and implementation of robotic systems.

In addition to programming paradigms, modern high-level software design is usually based on so-called patterns and frameworks to maintain overall structure and to support code reuse. Design solutions are often referred to as architectures. Use of the term architecture in software design stems from the fact that different software solutions have overall properties in common. This is also the basis for the development of *Design Patterns* and *Frameworks* [20]. Adding

another level of abstraction, even Architectural Patterns have been developed [12]. Such architectures still constitute a way of organizing classes, objects and their interactions to facilitate development and maintenance (but not usage) of the system.

2.2. Uncontrolled environments

A major problem for robots in space, and for autonomous robots and vehicles in general, is to maintain a model of the dynamically changing environment and to replan the motions according to environmental changes. An architecture should, of course, aid in the development of such systems including a large number of sensors, some possibly using active sensing as well as advanced sensory processing and world model updating.

From an AI point of view, an upper layer plans and transmits detailed actions down to the next lower layer, keeping the abstract plan in the higher layer. Unless some more sophisticated management of the interplay between the layers is introduced, the lower layer cannot cope with changes of goal or environment and will remain confined to dealing with the originally expected situation. Such a modification appears to be very hard to combine with hard real-time requirements. A more promising approach would be to also supply more abstract actions and replanning functions along with the detailed orders, a problem which is still the subject of research. Though the selection of suitable abstractions and associated architectures is still a topic of ongoing research [24,40,55,67], a promising prototype implementation of high-level concepts for robots in space is underway [53,54].

2.3. Intelligent industrial robots

Among architectures supporting planning and task-level programming, there are few systems that have proved workable. The AI Laboratory at the University of Edinburgh has developed a complete assembly system called SOMASS [22,38]. The system plans and executes assemblies in a special artificial block-type world, namely the Soma world. SOMASS has been demonstrated to work well despite a number of possible sources of failure. The uncertainties that may cause assembly failure include part tolerance, physical characteristics such as friction or stiction and the like. The following quote from [22] is central to the purposes of the paper:

"... the position that the planner should concentrate on those aspects of the problem that can tractably be expressed in symbolic form, leaving the execution agent to cope with the specifically manipulative difficulties of the assembly problem. Since the agent is hand-crafted, most of the consequences of the uncertainties in the parts and their manipulation are dealt with by the human programmer who has years of experience of object manipulation to call on when diagnosing and repairing failures in the tacit skills of the executive agent."

The fundamental standpoint in this quote is shared by us. In the context under consideration, however, the main interest is not planning or activity orchestration, but rather the executive agent and the supporting software layers for application programming.

3. Open Robot Control architecture (ORC)

We shall first justify the the choice of basic principle for our architecture, and then present the architecture as such. This is followed by discussion of some implementation issues which should also help to clarify the basic concept.

3.1. User views

From the perspectives of various categories of programmers who need to configure or program industrial robot systems, several programming situations have been identified. When solving a specific application problem, we may need to modify the system in several ways (control laws, operator interfaces, etc.) requiring different types of competence. Assume that we have one user type for each type of required competence, each user type viewing the control system in a certain way. Unless the system is carefully designed, any particular such view will be unnecessarily complex (involving a variety of computers, programming environments, special restrictions on use of interfaces, etc.).

If we instead base the architecture on properly selected user views, it is more likely that programming

can be done conveniently. A major difference from other current approaches is that the external view, rather than the internal implementation, is the primary matter. A view may map well onto internal modules based on some software paradigm or abstraction principle, but this is not an essential criterion.

Relations to available software principles provide some support concerning the design and implementation of the system, but the key issues are which views we need, how they are organized, their type of interfaces, and how to implement them. These issues will be treated in the sequel. In this paper we focus on the embedded control parts because the interconnections between high-level and low-level control are crucial in advanced robot application, and other architectures fail to solve this in an industrially feasible way.

A small step in the direction of user views is the use of dedicated user interfaces for different types of users. Industrial systems have various kinds of such interfaces, e.g., for control tuning, user programming, IO configuration, etc. The problem is that added tools either have to access several different parts of the system which implies that much complexity have to be dealt with and the tools will be dedicated to that particular type of controller (Fig. 1(a)), or the system has to provide an embedded real-time database for system-wide data access which adds overhead to the time-critical interplay between different levels of control and it makes it hard to limit data accesses to preserve safety (Fig. 1(b)).

Instead, our experiences from full-scale industrial systems suggest that the place where system functions (such as access to control signals) are defined and referred to from a user point of view should be decoupled from the level of control where those functions actually are computed/evaluated (Fig. 1(c)). Of course, a decoupled function has to be accompanied by a virtual context (such as a data type describing the subspace of accessible control states) that permits specification of control extensions. We will return to how such a decoupling can be efficiently implemented, but to get a first flavor of it the reader may study Fig. 2.

It is as a design principle for complex systems with a rich variety programming or operation possibilities that user views are beneficial. Then, a user view is a delimited set of services that has been made available for a certain type of user or usage. One user view can be expressed in terms of a collection of inter-related use cases [30], or possibly in terms of the grammar and semantics of a dedicated programming language or system. Thus, the concept of *user views* adds a new level to the design of complex systems and stands out as the primary architectural principle for robot control systems.

3.2. Software layers

The Open Robot Control (ORC) architecture is based on layers and typical users as shown in Fig. 3. The layers are characterized by the following:

- The layers are dedicated to programming cases requiring a certain type of competence. Within industrial production development, such a separation of concerns is crucial for system improvements.
- The servo control has been split up for control engineering reasons. The motion control layer coordinates and commands arm controller(s) and motor control of external axes. The popular research topic of advanced feedback control of robot motions is encapsulated by the arm control layer.
- The intermediate level has a specific layer for application-specific motion control, admitting more general and advanced control features than other systems do. Handling external sensors, or internal servo signals exposed by the motion control layer, in the application layer permits faster enactment and higher performance than user-level control does.
- The system programming level of other systems is mainly covered by the executive layer in ORC, a layer that also serves as a holder of the robot programming language, which is fixed in other systems.
- There is both an on-line and an off-line programming layer. These are uniquely integrated on an equal level basis as outlined in Fig. 5. The need for transformation of robot program, opposed to the simple down-load/upload used today, stems from differences in the way work-pieces are preferably referred to in the on-line and off-line programming cases.
- Task-level features are today usually implemented on top of off-line systems. Such features also define a higher-level user interface or programming environment. Therefore, task-level programming has its own software layer in ORC, see Fig. 4.

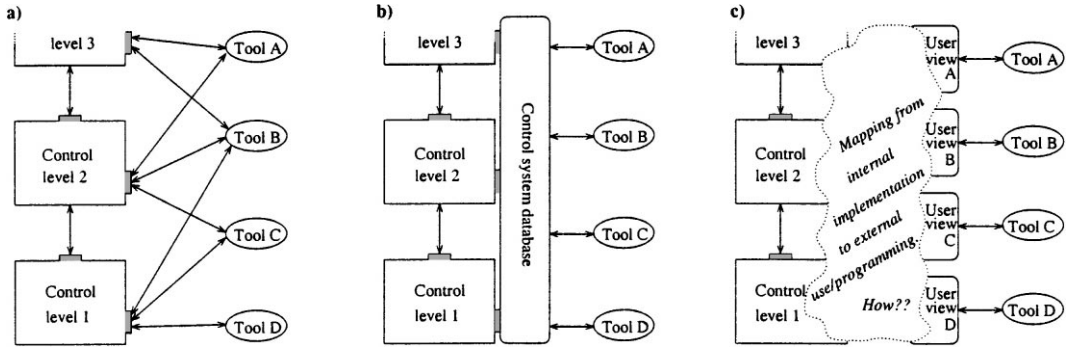

Fig. 1. Different ways of connecting user interfaces to multi-level control systems: (a) traditional, (b) database, (c) decoupled.

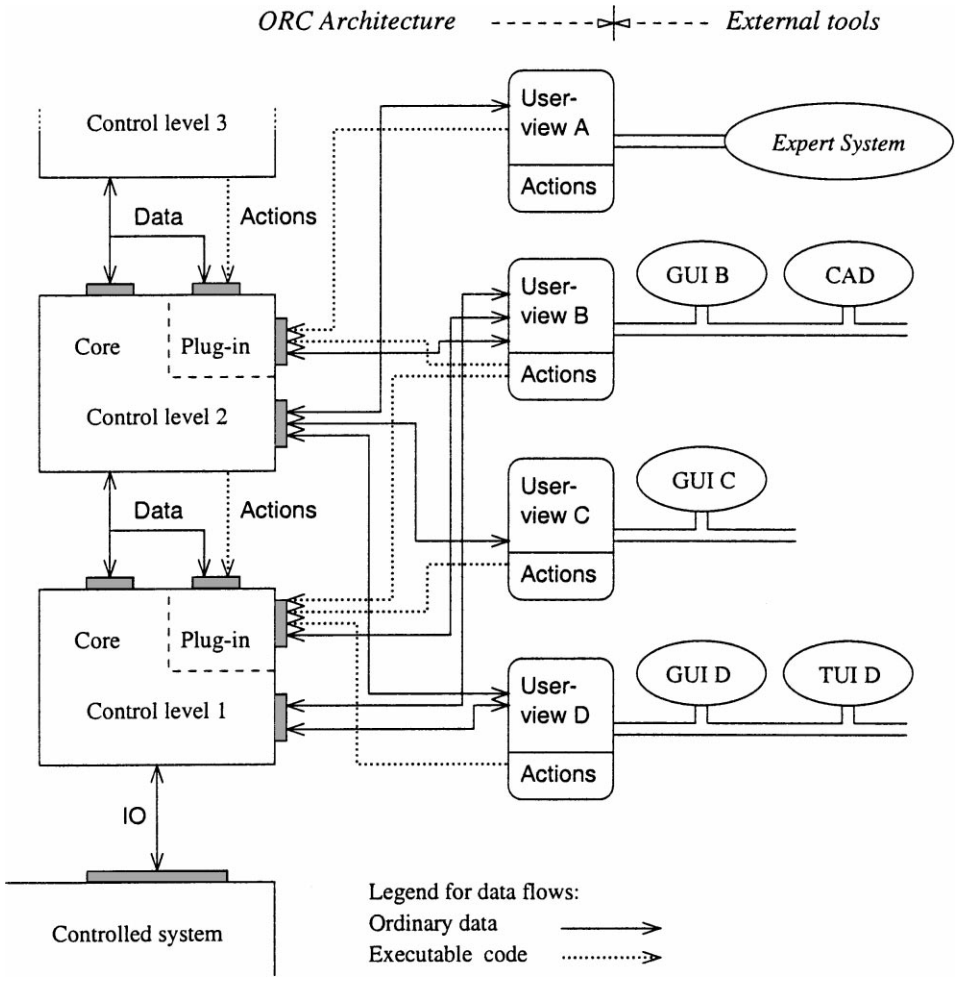

Fig. 2. Example of separation between implementation layers and user views which exposes the functionality needed for certain users and tools.

ORC layer	Encapsulates	Typical programmer	Typ. exec.	
Task level programming	Automatic programming from design data	Implicit from work–piece design (not possible today)	Interpreted	Host comp.
Off–line programming	Programming without use of robot	Robot programmer with computer experience		
On–line programming	Programming with use of robot	Production engineer or robot operator		Embedded system
Executive	RPL and control system interface	Computer programmer and exp. application engineer	Compiled	
Application control	Application specific motion control	Experienced application engineer		
Motion control	General control of workcell motions	Control engineer		
Arm control	Arm specific motion control	Robot control engineer		
Motor control	Control suitable for impl. in distributed hardware	Servo control engineer		

Fig. 3. Users and properties of software layers/views in the ORC architecture.

3.2.1. End-user programming

The aim that standard industrial robots should be possible to use as components for so-called intelligent robot control implies that task-level programming principles should be superimposed on explicit robot programming tools. Task-level programming facilities can of course be accessible directly from an on-line programming tool, but its software should rely on off-line programming. The reason is that off-line and task-level programming have the same need of abstract world modeling, whereas on-line programming utilizes the physical equipment which results in different requirements on the programming environment. The integration of the on-line (tangible) and off-line (abstract) interfaces is a topic of its own, including transformation of robot programs as depicted in Fig. 5 and described in [48].

Superimposing task-level programming on off-line programming results in the set-up shown in the upper part of Fig. 4. Note that even if task-level and off-line programming are based on the same tool (IGRIP [17] in our case), the architecture specifies that the task-level features should expose a uniform view to the user.

3.2.2. System-level programming

Properties of computer programming also appear within robot programming. On a system-level this means implementation of (robot and process independent) libraries, and conventional implementation of drivers for IO devices and sensors. This is simply a matter of (computer) programming appearing at any level of the system, and it is therefore not further discussed here. More important, system programming (from an end-user point of view) deals with aspects of specific robot functionality and control of the physical world. We prefer to separate these issues into two categories: (1) implementation of robot-specific libraries, robot programming tools, and robot programming languages; and (2) implementation of application-specific motion control.

From an engineering point of view, the latter — i.e., the tailoring of motion control — requires control

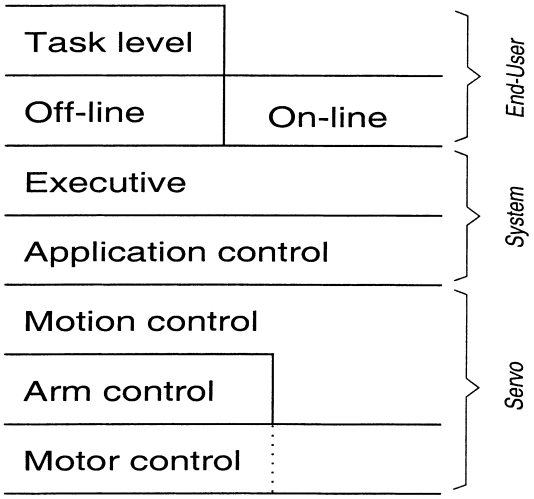

Fig. 4. The Open Robot Control (ORC) architecture.

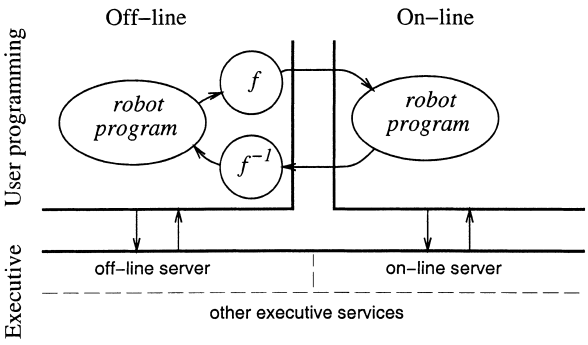

Fig. 5. Integration of on-line and off-line programming required transformation of the robot programs involved. These transformations are denoted f and f^{-1} here.

engineering competence, whereas the former (application support) does not. It is therefore reasonable and appropriate to define two different layers for these two types of programming, see Fig. 4. The lower layer for application-specific motion control is called the *application layer*, and the upper layer for tailoring of the programming interface is called the *executive* layer [50].

3.2.3. Servo-level programming

On the level of servo programming, a persistent feature is that software and hardware modules are characterized by their very close relationship. Hardware and software, at least for the motor control, are designed as integrated activities to facilitate price-performance optimization. Thus, the software naturally takes on a structure that reflects that of the hardware, whereas programming is performed by the implementors of the system. This implies that the user views of the architecture map well onto an object-oriented design of the motion control system, which in turn reflects the structure of the physical objects.

3.3. Implementation

Each software layer initially exposes a basic functionality in two directions; an interface to the next higher level of control and an interface to the engineering/programming/operation tools of the user. Utilizing the tools interface, the user or engineer may develop, install or use software supporting that particular type of user's needs. This may, of course, be done in multiple stages at increasing levels of abstraction. Typically, only the first stage is actually installed within the embedded system. See for example the Matlab Real-Time Workshop interface, designated SimulinkIO in Fig. 6, which then can be used from other Matlab tool-boxes. The high-level operations (or abstractions) added within a set of user tools should not be confused with functionality providing system abstractions. However, the control system interfaces that we actually need (or dare) to expose will be almost the same; on a specific level it does not matter if a planner operating within a higher software layer or an externally connected human operator abuses a too open system. On the other hand, the user of one view may hide some of the default functionality exposed to higher levels of control. That is necessary in some safety-critical applications.

If we can formalize the creation of high-level operations originally performed by the human operator, we can also achieve it within the ORC architecture (even if the AI system is handled from another user view), possibly using a meta-protocol like OpenC++ for compiled parts of the system [15]. Thus, the equivalence of the internal system and the external operator interface is attractive from an AI point of view, though in this paper we only describe the openness of the layers from a user point of view.

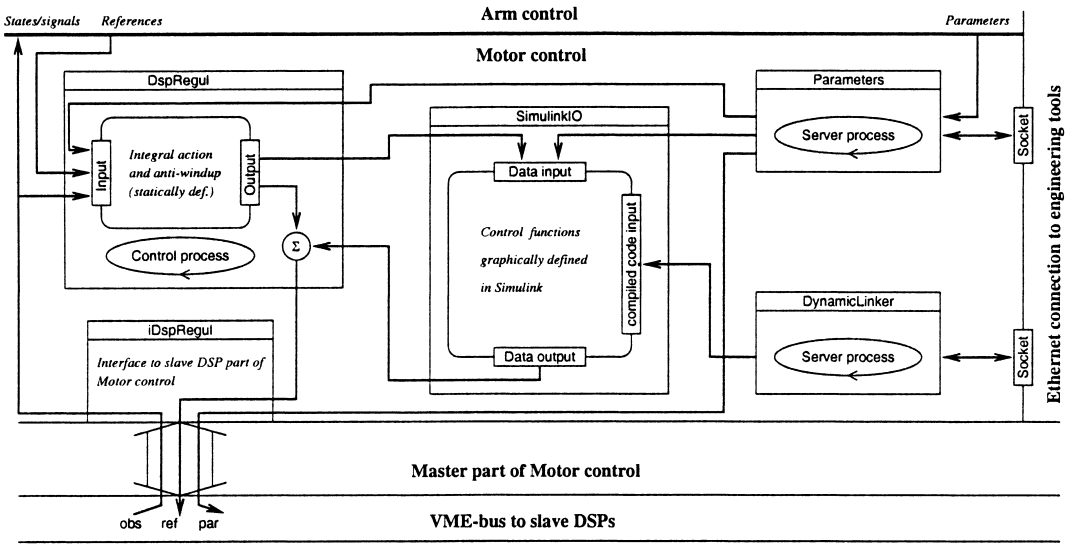

Fig. 6. Master part of the motor control implementation.

3.3.1. Combining flexibility and efficiency

Even though the internal implementation was not our primary concern for the definition of our architecture, the software slots of a certain user-view provide the desired flexibility, and the fixed/closed parts of the software provides safety which also can be made more extensive by additional modules created by the user. Nonetheless, in order to accomplish the user-views which do not map to any specific single-CPU software, we need to pass software pieces over CPU-boundaries. Software techniques normally used for multi-processor systems (such as message servers and remote procedure calls) would cause undesirably high data-flows at run-time. One alternative would then be to use symbolic or interpreted languages, though that would decrease efficiency and safety. For the non-real-time parts (if any) and for connection of user-interfaces, the web-technologies of recent years provide solutions. They do, however, not extend to the hard real-time control parts of the architecture. Therefore, taking demands on efficiency and predictability into consideration, we decided to use compiled functions that are dynamically bound at run-time to software already running in the embedded system.

The problem we are facing can be described in terms of a client and a server. The server is the embedded software already implemented, and possibly running. The basic service is first of all not only to perform the control, but also to serve requests to change the control behavior. It is the latter aspect that is treated here. The client acts at a higher hierarchical level, which means that it contains more knowledge about the application. Depending on what is appropriate to the particular application, the client requests the lower-level control (i.e., the server) to use different new control functions. We then need some means of expressing these functions, and to dynamically let the server evaluate them.

In this paper, the term *action* will be used for such a compiled function that is to be dynamically bound in the embedded system and we distinguish between two types of action implementations:

1. The simplest version is so-called *function-based* actions. Such actions are functions that are compiled into position-independent code which is then called with pointers to the target context. This type of action can be achieved by ‘pointers to functions’ in the special case of a single CPU within one address space. The main advantage is that a minimum of run-time support is needed on the target CPU. We use it mainly when the high levels of control need to perform control actions at the lowest level of feedback/real-time control. For example, a simple ‘impact signature catcher’ was

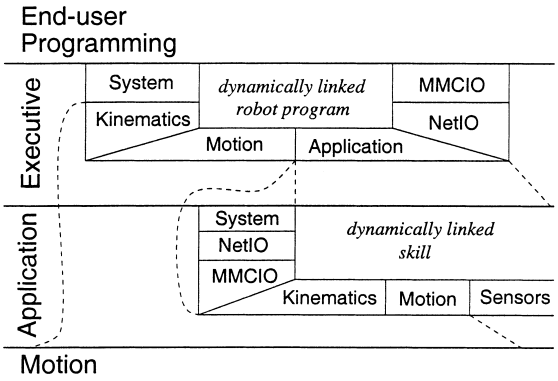

Fig. 7. The *dynamically linked robot program* and the *dynamically linked skill* are the slots where actions are installed to extend the capabilities of the robot.

passed at run-time via a master M68040 CPU into on-chip memory of a DSP (used by the motor control), requiring only 212 bytes of memory and 3.4 μs of execution time in the target DSP (worst case, including overhead).

2. The more general type of actions are the so-called *dynamically linked actions* — e.g., actions consisting of complete objects, or even complete programs — which have proven useful at intermediate levels of control. For example, robot capabilities were extended (at run-time) by introducing additional feedback control loops within the Application layer. Such an action may be installed from a high-level planner, and it may in turn contain other actions that are passed to the underlying motion control. The application layer slot was defined in such a way that installed actions (designated *skill* in Fig. 7) extended the available primitives of the Executive (language) layer. Research concerning on-line incremental extensions of robot programming languages (grammar and semantics) is in progress.

When using a type safe language, both types of actions are type safe. The use of actions as plug-ins is shown in Fig. 2 with further examples on application in [49].

To ensure that real-time deadlines in the system are met, the timing of individual actions is of paramount interest. The reader may recall the Spring Kernel [61] and the Synchronous approach [58] as relevant examples. If the system is statically scheduled, the schedule has to take into account the extra time that actions are allowed to take. The timing constraints then constitute a property of the action slot, and need to be checked when actions are installed. In dynamically scheduled systems, rescheduling has to be done when actions are loaded. In both cases, the execution time of the action has to be supplied from the client and/or determined or checked by the server.

3.3.2. Prototypes

The work done on the implementation of all layers includes the following:

- A task-level interface based on IGRIP has been developed by other members of our project [11]. Connecting high-level planning and expert systems to the world-model database of available off-line systems appears to be a practical way of creating task-level interfaces.
- A programming interface to application-specific motion control was developed at our laboratory [7]. To the user, it exposes a set of module interfaces encapsulating the available features of the application layer as well as the available internal contexts of the underlying motion control system (Section 3.3).
- A control engineering interface for the motor control layer was created based on Simulink Real-time Workshop interfaces [39] and some additional Matlab-based graphical interfaces to DSP filters [25]. The user could conveniently tune the feedback loops using available tools (on the host computer) for control design. Alternatively, using an intermediate text-based interface, such changes of the feedback control can also be performed from a high level planner (for example, to introduce a high-frequency limit cycle to avoid stiction during certain robot operations).

In some respects, motor control is the simplest part to design because its structure is not that complex. On the other hand, the real-time requirements are strict and the hardware must be efficiently utilized despite the interactive access we need.

3.3.3. Motor control example

The motor control was mapped to the hardware in such a way that digital signal processors (DSPs) were utilized for the high-frequency properties of the control, and a master M68040 CPU was used for the low-frequency part. The reason is that the DSPs execute pure filter algorithms very efficiently, but inte-

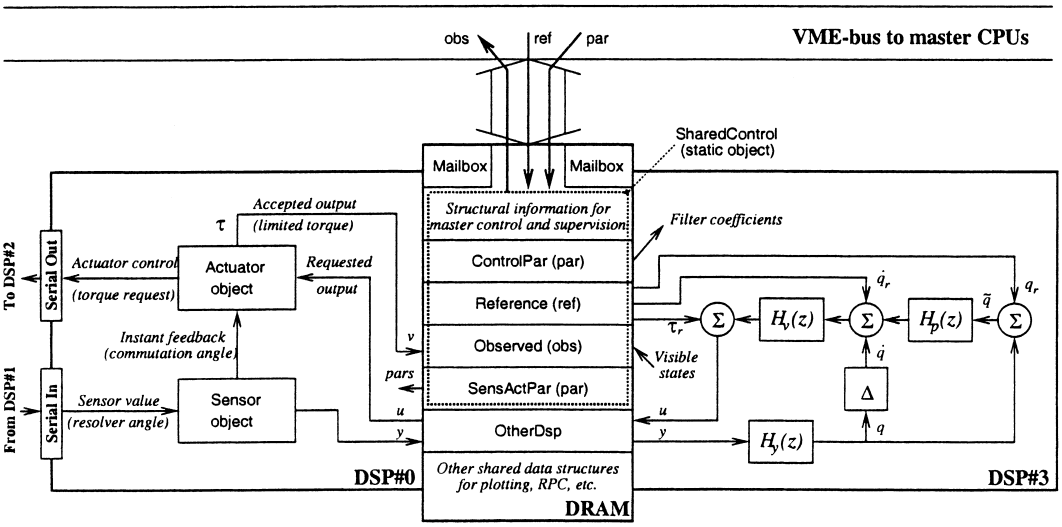

Fig. 8. Motor control implementation on the DSP hardware level. Only the two main DSPs DSP#0 and DSP#3 are shown. Other DSPs handle sensor input, actuator output, and AC-motor torque control (for an ABB IRB-2000 robot in this case).

grating action and the necessary wind-up protection contains logic that breaks the floating point pipeline in a very undesirable way. The M68040 CPU, on the other hand, is better suited for general control tasks [13]. Sampling frequencies up to 1 kHz were used on the master CPU, whereas frequencies up to 24 kHz was used on the DSP part. (The hardware interface to ABB robots, designed and built at our laboratory, allows sampling frequencies of up to 48 kHz [48].)

Fig. 8 shows the data flow for the DSP part of the control. The DRAM memory is a central feature for the multi-processor implementation. Control parameters and references are written by the master processor to the DRAM, the two DSPs shown in the figure exchange control data via the DRAM, observed states are updated by the DSPs and it is then used both by the master part of the control and by another M68030 that protects the equipment from illegal and possibly damaging control actions. The programs were written in C++ [47]. However, the filters $H_y(z)$, $H_v(z)$, and $H_p(z)$ were available as highly optimized assembly code from the application library for the DSPs.

The master control part running on the M68040 board is shown in Fig. 6. It contains a fixed implementation of an I regulator which can integrate speed or position error according to parameters set at run time. The I-part deals with the low frequency behavior of the control, whereas the HF-control governs the high-frequency part. Advanced motor control such as active damping typically takes place in a mid-frequency range. The maximum sampling frequency of 1 kHz for the master control is enough for this, given the HF-control part. For the rapid prototyping of new control principles, code generation from block diagram descriptions on the host computer forms a valuable tool, as mentioned above. To load and unload the Simulink descriptions at run-time, we use the action mechanism handled by the DynamicLinker module shown in Fig. 6. Special software solutions were developed to handle the differences in address spaces, real-time primitives, floating point representation, and byte numbering, as described in [48]. For clarity, data logging services have been omitted.

4. Applications and evaluation

The proposed architecture has been confronted with a number of application problems ranging from special low-level control for spot-welding applications to improved autonomous operation for robots performing deburring of castings [48]. As an example of typical multi-layered requirements, we shall now briefly describe some aspects of arc-welding robots.

4.1. Arc-welding control and programming

Of the several needs for intermediate-level programming within arc-welding applications, the following situations have been studied:

- *Basic functionality.* A basic application package for arc welding includes control and programming features for waving motions and control of the welding equipment and interfaces to common types of welding equipment. The ArcWare package [1] developed within ABB Robotics is an example of well-designed software for this purpose.
- *Path tracking.* Available industrial systems usually also support weld-seam-tracking using a laser-scanner sensor. The sensor is then integrated with the motion control system, but this can so far only be done by the robot manufacturer (because available systems do not provide an open application layer). The path tracking control problem includes estimation of the weld seam based on (often very noisy) data from the laser-scanner, and control of robot motion in such a way that the seam is tracked. The stochastic nature of the problem, and the desire to perform high-speed welding of thin-sheet metal without losing track of the seam necessitates a stochastic control approach.
- *Welding-specific functionality.* Arc-welding is a quite complex electrical, mechanical, and chemical process, which can be influenced by voltages, currents, and weld-tool motions. For special materials, or for welding with special demands on quality or productivity, special welding principles have been developed [2]. Sensing and control (of voltages, currents, weld-joint geometry, etc.), and close interaction with the robot motion control are of key importance.
- *Task-level programming.* Even if on-line programming is claimed to be superior concerning operator adjustments, it should be borne in mind that the initial programming of, for instance, advanced arc-welding programs may often be done better off-line. That is because CAD data for the work-pieces can be used for the definition of welding paths and end-effector orientations. The off-line programming system also provides a platform for task-level programming using knowledge-based techniques. So far this does not imply any need for intermediate-level control, but our desire to use 'hand-crafted agents' does. Special welding techniques according to the previous item should be possible to load and refer to from the task-level system.

A thorough review indicated that the proposed control system design should suit the basic functionality and path tracking needed, but implementation of those parts has been done within our project. As industrially available systems can handle such items, these aspects are omitted here. Instead, let us focus on the desired 'welding-specific functionality' and on the 'task-level programming'.

Thus, we want a system admitting incorporation of welding control at an intermediate level (here, in the application layer), and an on-line connection to a task-level programming system. The welding control should be possible to define and install from the task-level system, and it should be possible for motion commands from the task-level system to utilize the loaded welding skill.

4.2. Implementation

Efforts to support advanced arc-welding were inspired by the welding experts at the next-door laboratory at the Department of Production and Materials Engineering (DPME). Implementations were done in close collaboration with them. This activity is still in progress. At the time of writing, the situation is the following.

Techniques for control of special welding have been developed and experimentally verified at the DPME [2]. Because available systems are not open enough, such research entails the addition of controllers external to the robot controller. The solutions have been studied to obtain requirements on open robot control systems. The feedback control nature of the welding control is visualized in Fig. 9. Clearly, implementation of the welding control at the end-user level of the system would be very inefficient (e.g., many additional function calls and data transfers for each sample). Instead, we want to implement this within the application layer. The observers may require additional hardware to be plugged in (on the VME-bus in our case), which is a standard procedure, but principles for the application layer programming would be similar to what has been described above for the

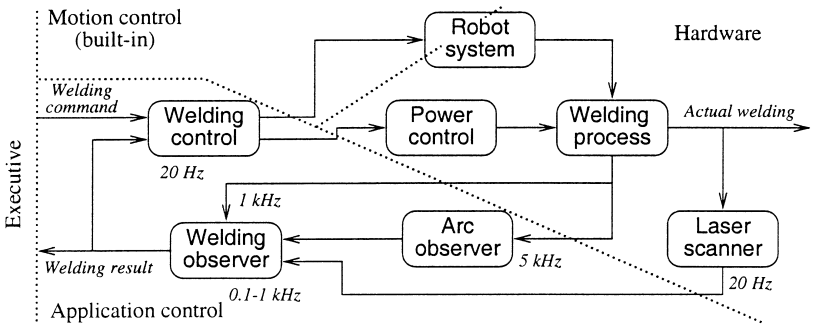

Fig. 9. Block scheme interpretation of welding control according to [2].

deburring control. Installation of a special welding controller will return an identification number to the host.

Implementation of task-level programming for arc-welding was made based on the IGRIP [17] off-line programming system; results are reported in [9]. An on-line connection between the task-level system and the robot was needed. Using the features of the ORC architecture and our experimental robot control system, this was accomplished as follows:

- The shared library in IGRIP was extended with functions for teleoperation that during robot simulation transmitted the robot joint trajectories to the embedded controller which was located over 100 m away in another laboratory. The IGRIP part of the interface is based on UNIX sockets connected to Internet.
- A server running on a workstation in our laboratory was developed. This server connects the off-line system to the embedded control system whenever these clients are available and responding properly.
- On-line control engineering was supported by our Matlab-based software. Tuning of a few control parameters is shown to the upper right in Fig. 10; for brevity, tools for on-line system identification are not shown.
- An IGRIP server for the embedded controller was developed. This server is installed at run-time into the executive layer. The server accepts trajectories from IGRIP as robot commands. Call of embedded system primitives from the IGRIP system is accomplished by escape codes (negative numbers) in the time column of a supplied trajectory. One such code is used to refer to robot skills loaded into the application layer. (The identity of the skill is supplied as the second number of the trajectory sample.)
- Video cameras, a frame grabber in the Sun workstation, and SunVideo software was used to remotely observe the robot motions.

Though Internet-based teleoperation of the robot is possible, it is currently hard to rely on dynamic feedback information transmitted through the available network, due to the often quite limited data rate (bandwidth). In our case, to speed up the video interface, a dedicated 10 Mbit/s connection was installed.

4.3. Experiences and safety considerations

Based on the combined need for appropriate user views and the flexibility needed for new unforeseen applications, we use the proposed concept of actions as a solution. Both actions and built-in parts of the system can provide configuration/tuning support by allowing parameter changes. By restricting the values of the parameters, the system can keep its tuning operational and safe. In the motor control example above, there was not only a built-in part with filters that could be adjusted from higher levels of control, but also an action slot where additional control strategies can be inserted. Previous industrial experience shows that such special controllers are highly desirable in some cases, such as for optimized short motions in spot welding [48].

Considering the specific example of arc welding, our system provides a control engineering interface which in its unrestricted version is well suited for the work normally done by the robot manufacturer. Fol-

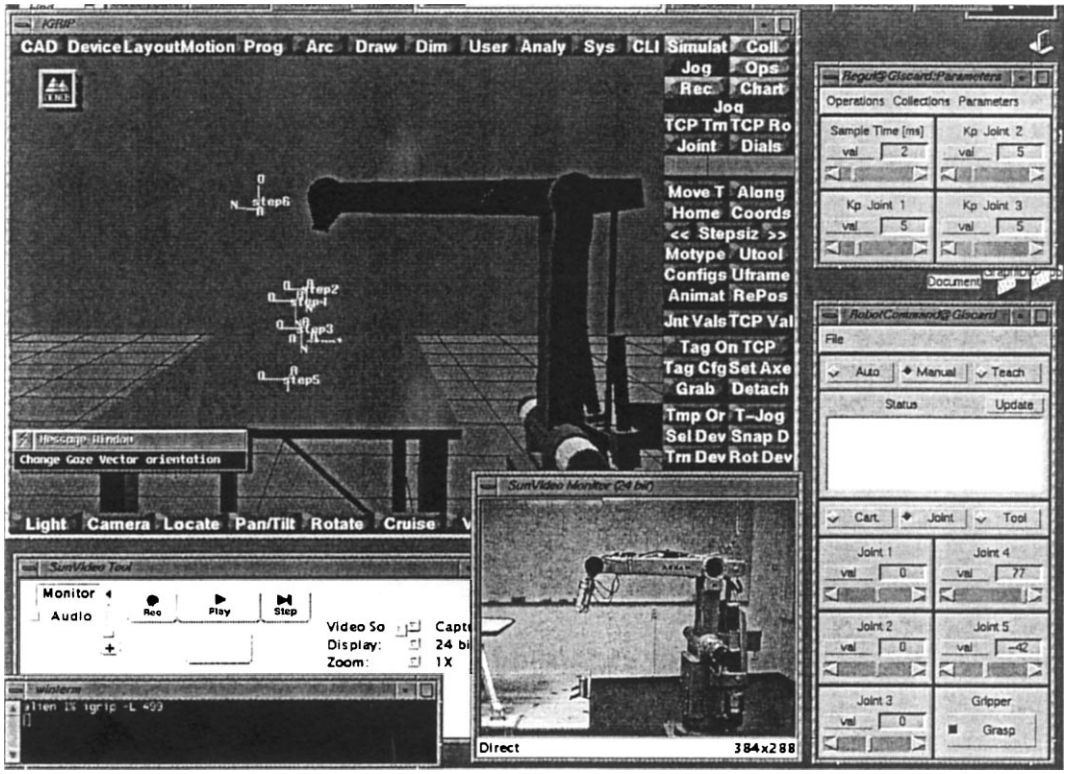

Fig. 10. User interface to off-line robot simulation (at DPME) with on-line connection to the physical robot (at the Department of Automatic Control). The robot (located 100 m away) is observed via a video interface (available on a Sun close to the robot and connected back to the X-server on the SGI running IGRIP). The robot can also be manually controlled and tuned using our Matlab-based interface shown to the right.

lowing ORC, a new part of that control design is to decide what functionality that should be exposed for future tailoring of the control. Findings on our experiments suggest a reasonable approach appears to be to permit arbitrary algorithms and additional states as actions may generally contain, but also to impose restrictions by:

- Exposing only a restricted set of control signals (some read-only), though others can be changed.
- Limitation of the magnitude of the changes that instantly affect motions (but still maintaining short response times allowing corrections done at, e.g., kHz rates).
- Checking added high-priority CPU-load, using software timers and supervision threads.
- Maintaining original low-performance control for possible activation by the built-in supervision system.

Note that keeping the safety functions adds another lever of protection. Therefore, plugging in arc-welding actions providing the feedback control described above works safely with small high-frequency corrections. But if needed, how can larger corrections be accomplished?

The answer is *cascade control*; larger corrections are made at higher levels of control at a slower rate. This affects the low-level set-points which can then be further corrected, but task-planners, operator interfaces, or restrictions defined at the end-user level still work properly.

For weld quality control, small but rapid corrections of waving, wire-feed, and voltages are sufficient. But for high-speed welding of a partly unknown seam, seam-tracking motions to locations far outside the accepted range for high-frequency corrections may be necessary. To describe the benefits of the ORC archi-

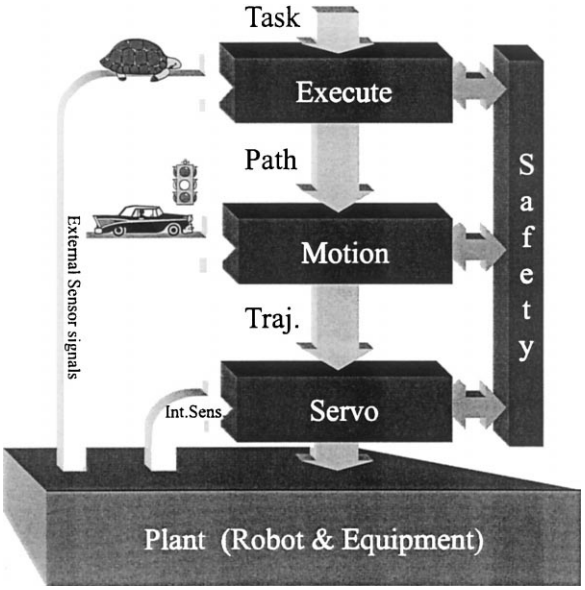

Fig. 11. Industrial controllers have many safety functions working on all levels of the system. Almost any sensor can be incorporated on a user-program level, but only some specific sensors are allowed (decided by robot manufacturer) on a feedback-control level.

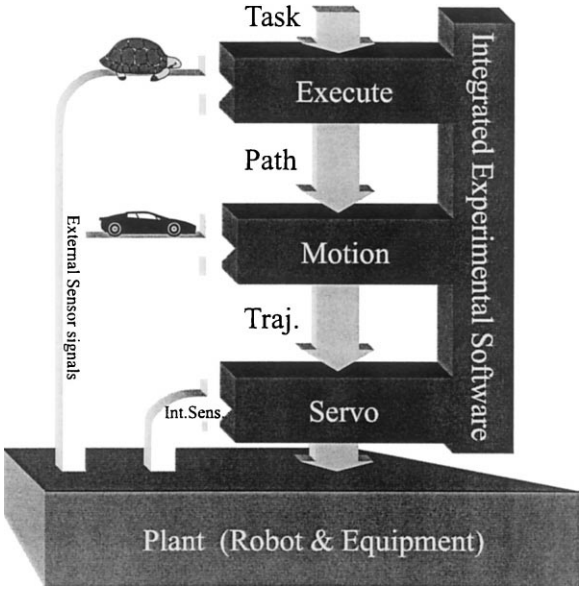

Fig. 12. Research-type controllers form fully open systems, and feedback loops can thereby be freely accomplished. But this type of system does not comprise any industrially useful safety net (which major robot manufacturers put years of experience and work in).

tecture in this case, we need to compare this approach with others.

- *Industrial systems.* If the desired control happens to be supported by the controller, e.g., by setting parameters and activating the control, it is unusually fortuitous. Otherwise, it takes a big customer (such as a major car manufacturer) to get the required functions implemented. In other words, as denoted by the traffic light in Fig. 11, it is decided by the system whether fast feedback is available or not.
- *Research systems.* To circumvent the restrictions of available systems, research-type systems are fully open, perhaps even with source code for the entire system. The researcher is then free to introduce any type of control, but proper system behavior in an abnormal production situation (perhaps involving human operators and a sensor failure) cannot be guaranteed. This is depicted in Fig. 12 by full-speed input of external sensor signals but with the safety block shown in Fig. 11 replaced by an experimental software block.
- *Our approach.* One solution is to have the above restrictions built into the system, and to provide slots where actions can be plugged in to extend embedded functionality. This is depicted in Fig. 13, where the action slots are denoted 'Plug-in'. Note that fast response to external sensors is always possible to introduce, and large corrections can be achieved by slower feedback via higher levels of control where multiple built-in safety restrictions are working. For instance, the maximum path deviation between requested and actual motion will be kept small.

Figs. 11–13 illustrates how tight application-specific feedback is accomplished in ORC. As a front end to those levels of control, user views are defined as depicted in Fig. 2 to aid in managing system features. One such user view is the application layer that handles the issue of tight application-specific feedback control.

5. Discussion

Industrial manipulators are characterized by intensive interplay between user-level commands, which often appear robot independent, a good (but not perfect) world model, motion control services, and exter-

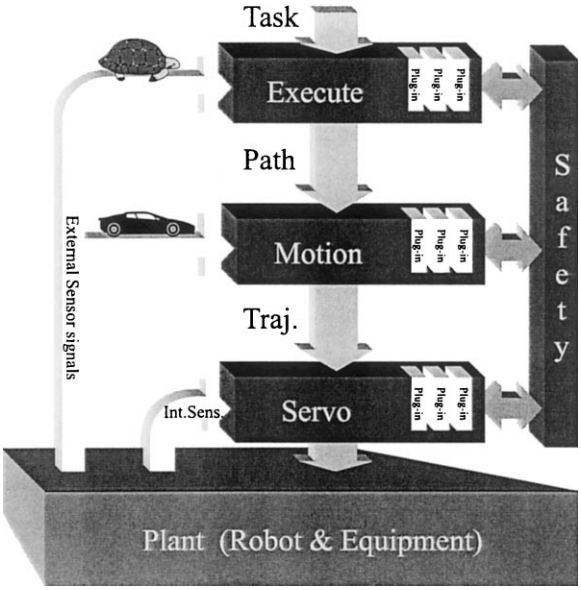

Fig. 13. Our hybrid approach includes the (possibly revised) industrial safety functions, but permits additional control functions to be plugged into the system. The plug-in slots are defined by in the embedded controller in such a way that both safety and application-specific performance can be achieved.

nal signals from, say, a welding or grinding tool. This interplay is crucial not only to obtain flexibility and performance, but also to avoid the cost of otherwise necessary external sensors.

The motion control system provides a set of robot functions. Seen from the outside it consists of data and procedures. Programmers often regard it as a set of device drivers. One might consider the possibility of trying to find a complete set of well-defined procedures — i.e., some form of generic set of robot functions. These functions could then form a hard shell (no reason to enter) library, where the internal implementation is hidden and optimized. We can then consider the robot as an abstract data type [66], and it can even be conveniently incorporated in an object-oriented framework [43]. There are proprietary and safety reasons for having a closed system, and they are also easier to implement than open systems are. This explains why several of the currently available robot control systems seem to have such a closed structure. This is a major reason for the difficulties in slightly modifying motion control to include a new sensor, etc. This leads to a situation when it is no longer possible for the robot manufacturer to do the required modifications. Thus, the flexibility or performance becomes unnecessarily limited. It is therefore important for the purposes of the approach outlined in this paper that know-how about the application can be added on top of the built-in motion control as shown in Fig. 14. Note that it is only in simple cases that application-specific motion control can be achieved by changing available control parameters. More often, new control strategies need to be added which puts special demands on the implementation.

The term ‘Plug-in’, used in Fig. 13, is nowadays also used for code loading over the Internet, e.g., to extend the functions of a web browser, but such a plug-in is requested and run at the client side. In ORC it works the opposite way; an action is defined and issued from the client side, and its operation is performed by (and in the context of) the server/controller. Additionally, the actions are subject to dedicated slot-dependent (and sometimes application dependent) access restrictions for system security reasons. The ability to extend the available set of functions in one high-level slot from actions plugged into a low-level slot, as shown in Fig. 7, makes it possible to maintain the user views of the extended system.

An architecture should be long-lasting and stable, but yet flexible and extensible. It should be simple and understandable, otherwise it will not be used. A

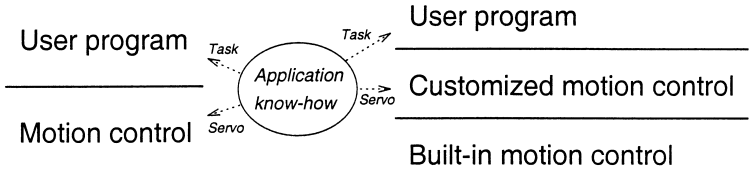

Fig. 14. Application know-how gets expressed in robot programs. Presently (left) however, it also affects the built-in motion control. The proposed system (right) contains a software layer for application-specific (customized) motion control.

remaining problem with architectural development is how to know what architecture is the best or 'optimal' (in some sense). There are two approaches to deal with this problem; formal methods and experimental verification. As formal methods are difficult to apply to heterogeneous systems, we are left with experimental verification which has been done. Having experienced the properties of our architecture, work with additional arc-welding control is under way.

5.1. Misconceptions

Tackling ill-posed problems can create confusion. Depending on the reader's background and previous experience (e.g., from some existing system), different parts of the solutions presented can give rise to some confusion concerning approach, importance, novelty, etc. Based on reactions to viewpoints presented in technical discussions and earlier work [45], the following remarks are made in order to pin-point some standard misconceptions.

The purpose of software architecture

Software architectures have received much attention within robotics research [3,14,40]. One reason is that when robot controllers (e.g., for space applications) become more and more complex, abstractions and software structures are introduced to cope with the complexity. In other words, the purpose of the architecture is to make the implementation of the system feasible.

In industrial robotics the software is complex and various functions must be tightly coupled to achieve efficiency. However, the implementation complexity is not worse than that which can be handled by proper software engineering methods such as an object-oriented design. However, the variety of user interactions in flexible manufacturing systems indicates that user views of the system should constitute the basis for the architecture. This is a completely different approach that should not be confused with implementation architectures.

"We can do that in our system"

When suggesting a new embedded control system, there will always be alternative means of implementation. Take, for instance, some advanced process control systems. First, such a system can of course be used to control a robot, but will the desired performance, cost-effectiveness, programmability, and flexibility be achieved? Secondly, when designing servo control using process control systems, will use of a specific system and its special language etc. be appropriate for interfacing to stand-alone servos? In conclusion, almost anything feasible can be implemented in any system, but specific application demands as considered in this paper are typically not taken into consideration. Although the desired application software structure may sometimes be accomplished, performance specifications are not met simultaneously.

"Layered systems are not useful without a detailed specification"

Layered systems are perhaps most common within computer communication. Within computer and telecommunication applications, it is crucial that the specification of the layers is complete in all its details. It must be possible to integrate components and layers from different vendors, and the layers reflect the implementation.

In this work, the layers reflect *user views* of the system. Detailed internal interfaces could of course also be developed, but that needs to be done in collaboration with major vendors and/or standardization organizations. Otherwise, the industrial impact would be too small. On the other hand, we claim that earlier and current standardization of robot interfaces on high [41], intermediate [26,65], and low levels [27] are inappropriate. Because this work is not devoted to some new programming language, it does not depend on a detailed standard. It is usage of the principles proposed that yields the benefits. Standards come with maturity!

"A robot controller is just another PLC block"

Process control systems and PLCs (programmable logic controllers) often control motions usually via dedicated servo controllers containing the drive electronics and the low-level feedback control. Process controllers are typically programmed by combining and connecting PLC blocks into a block diagram defining the control program. A servo controller can then be encapsulated in such a block. What then is a robot controller? In simple and less demanding cases, a robot control system is simply a multi-axis programmable servo controller. For the reader whose experience mainly derives from such applications, it may be hard to understand why robot control should

be such a big issue; it is just another PLC-block. However, investigating demanding use of industrial robots, the needs of motion descriptions, operator interactions, non-linear and variable structure control, and computing efficiency clearly show that robot control requires its own control techniques.

"We already have an open system"

A system that is *open* allows the user or system manager to change or add certain internal components of the system. In practice, systems are a mixture of open and closed parts [19]. The open parts can also be open in many ways. For example, consider a robot control system with a replaceable trajectory generator. Such a system can be claimed to be open. However, the interfaces to the software component (the trajectory generator in this case) could be so rigid that only the algorithm can be replaced. In other words, structural changes involving, for example, new types of interaction with the servo control are often not possible. Therefore, an open system should be reviewed concerning the type of changes possible, and what degree of flexibility that implies. However, it should be borne in mind that there may be safety and proprietary reasons for keeping parts of the system closed.

"Our customers have not required that feature"

Industrial development has to focus on customer requirements. Sometimes, however, new features that let customers explore new possibilities simply have to be offered. Proposal of new features for application or customer support engineers, sometimes results in a comment that "our customers have not required that feature".

This research is inspired by real industrial problems, but specific solutions can very well be questioned in the light of short term requirements. The reader should in those cases, however, not forget the fundamental long-term benefits.

"A new type of motion is just another procedure"

Principles for the incorporation of application features have not received the same attention as other software aspects in robotics, such as high-level planning or low-level explicit joint control. This aspect has been dismissed with such statements as "just implement a procedure" or "implement another robot function". On the other hand, robot manufacturers exspend major resources in designing and implementing such robot functions. Even so, it is well known that it may be difficult or impossible to slightly modify a function, to change an application feature, or to include a new type of sensor in existing systems. The reason is of course that the software is complex with many coupled functions that are based on mutual primitives, include timing and so on. The seemingly innocuous task of including a new robot function may represent a major effort. Applications mature over time, and it is natural that more and more special features need to be implemented. If this can be done efficiently using the principles proposed in this paper, then the implications for production speed and efficiency are obvious.

6. Conclusions

Efficient design and industrial applications of advanced robot control systems require support for installation of application software for improved intelligent behavior and of tight application-specific control loops. Thus, robot control systems need to be *open*. We have achieved that objective by using functional operators as parameters which can be changed from the next higher level. Such operators consist of pieces of cross-compiled executable code. A unique combination of efficiency and flexibility was thereby achieved.

We also identified the need for software architectures to cope with the internal complexity of autonomous robot operations as well as with the multiple types of user interaction. Whereas architectures usually deal with the internal complexity of the system, we have proposed an architecture primarily intended for support of the user or programmer of the system, which is manifested by the new architectural concept of *user views*. Within each such user view, interfaces and tools constitute a uniform and integrated environment which also forms a software layer for the design of system architectures.

The proposed Open Robot Control (ORC) architecture defines multiple software layers for different types of users and programming situations. Use of the functional operators turned out to be a key mechanism for implementation of the ORC architecture. It made the user-view concept applicable, for instance, by

allowing tight control loops to be conceptually defined within one (high-level) view and then efficiently carried out by the run-time services of another (low-level) view. This proved to work even down to the lowest level of (motor) control, and even when industrial demands in terms of efficiency were considered. Nevertheless, ORC also supports high-level abstractions, and several other architectures can be achieved as special cases.

We have experimentally verified that the ORC architecture efficiently responds to combined needs such as performance, flexibility, task-level operation, and incorporation of intelligent sensors.

Acknowledgements

This research has been supported by NUTEK (the Swedish Board of Technical and Industrial Development) We thank Lars Nielsen for useful discussions on the topic of application-specific control. Thanks are also extended to our colleagues at DPME for cooperation on IGRIP and the arc-welding application. We thank Marcel Schoppers for discussions and access to some unpublished material. Friends and former colleagues (of KN) at ABB Robotics are also acknowledged for fruitful discussions on robot control and applications.

References

[1] ABB Flexible Automation, S-721 68 Västerås, Sweden. ArcWare User's Guide, Article no. 3HAB 0011-13, 1995.

[2] B. Ågren, Sensor integration for robotic arc welding, Ph.D. Thesis, TMMV-1023, Department of Production and Materials Engineering, Lund Institute of Technology, Lund, Sweden, 1995.

[3] S.A. Albus, H.G. McCain, R. Lumia, NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), Technical Report, Technical Note 1235, US National Bureau of Standards, 1987.

[4] R.J. Anderson, Smart: A modular architecture for robotics and teleoperation, in: Proceedings of the IEEE Conference on Robotics and Automation, Atlanta, GA, May 1993, pp. 416–421.

[5] J. Andersson, Ett öppet system för programmering och styrning av robotar (An open system for robot programming and control), Master's Thesis, ISRN LUTFD2/TFRT-5557-SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, April 1996.

[6] R.L. Andersson, Computer architectures for robot control: A comparison and a new processor delivering 20 real Mflops, in: Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989.

[7] R.L. Andersson, System design for robot control with a scalar supercomputer, in: Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, May 1990, pp. 1210–1215.

[8] A. Benveniste, G. Berry, The synchronous approach to reactive and real-time systems, Proceedings of the IEEE 79 (9) (1991) 1270–1282.

[9] K. Brink, Event-based control of industrial robot systems, Lic Tech Thesis, Department of Production and Materials Engineering, Lund Institute of Technology, Lund, Sweden, 1996.

[10] K. Brink, M. Olsson, G. Bolmsjö, Event based robot control, in: Proceedings of the First ECPD International Conference on Advanced Robotics and Intelligent Automation, September 1995.

[11] K. Brink, M. Olsson, G. Bolmsjö, Increased autonomy in industral robotic systems: A framework, Journal of Intelligent and Robotic Systems, 19 (1997) 357–373.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, New York, 1996, ISBN 0-471-95869-7.

[13] S. Caselli, E. Faldella, F. Zanichelli, Performance evaluation of processor architectures for robotics, in: Proceedings of the Compeuro, IEEE Press, Piscataway, NJ, 1991, pp. 667–671.

[14] R. Chatila, S.Y. Harmon (Eds.), Proceedings of the Workshop on Architectures for Intelligent Control Systems, IEEE International Conference on Robotics and Automation, Nice, France, 1992.

[15] S. Chiba, A metaobject protocol for C++, in: Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications, vol. 30, OOPSLA, ACM Sigplan Notices, October 1995, pp. 285–299.

[16] J.J. Craig, Introduction to Robotics: Mechanics and Control, 2nd ed., Addison-Wesley, Reading, MA, 1989.

[17] Deneb Robotics, Detroit, USA. IGRIP Users Manual, 1995, IGRIP is a registred trademark of Deneb Robotics, Inc.

[18] V.J. Eng, Modal segments for a peg insertion task, Technical Report, Harvard Robotics Laboratory, 1988.

[19] W.E. Ford, What is an open architecture controller, in: Proceedings of the Ninth International Symposium on Intelligent Control, 1994.

[20] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, ISBN 0-201-63361-2.

[21] N. Halbwachs, Synchronous Programming of Reactive Systems, Kluwer Academic Publishers, Dordrecht, 1993.

[22] J. Hallam, Autonomous robots: From dream to reality, in: Proceedings from 'Robotikdagar', Linköping, Sweden, May 1991.

[23] S.Y. Harmon, Architectures: Designers vs. implementors, in: Proceedings of the Workshop on Architectures for Intelligent Control Systems, IEEE International Conference on Robotics and Automation, Nice, France, 1992.

[24] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, M. Balabanovic, A domain-specific software architecture for adaptive intelligent systems, IEEE Transactions on Software Engineering, March 1995, pp. 288–301.
[25] B.H.J. Hendriks, Implementation of industrial robot control, Master's Thesis, ISRN LUTFD2/TFRT-5555-SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, March 1996.
[26] The International Organization for Standardization (ISO), Technical Committee TC184/SC2, ISO/TR 10562:1995 http://www.iso.ch/, Manipulating industrial robots — Intermediate Code for Robots (ICR), 1995.
[27] IPK Berlin, Realistic robot simulation interface specification, Dr. R. Bernhardt, Pascalstr. 8-9, 10 587 Berlin, Germany, 1994.
[28] J. Ish-Shalom, Task level specification of a robot control, in: Proceedings of the IEEE International Symposium on Intelligent Control, Philadelphia, PA, 1987.
[29] IVF-KTH, Montering, idag och bortom år 2000 (Assembly, today and beyond year 2000), Monteringsenheten, 100 44 Stockholm, Sweden, September 1995.
[30] I. Jacobson, Object-Oriented Software Engineering, A Use Case Driven Approach, ACM, New York/Addison-Wesley, Reading, MA, 1992, ISBN 0-201-54435-0.
[31] R. Johansson, M. Spong, Quadratic optimization of impedance control, in: Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, May 1994, pp. 616–621.
[32] A.J. Koivo, Fundamentals for Control of Robotic Manipulators, Wiley, New York, 1989.
[33] K. Kosuge, K. Furuta, T. Yokoyama, A control architecture for intelligent mechanical system: Virtual internal model following control, in: Proceedings of the IEEE International Symposium on Intelligent Control, Philadelphia, PA, 1987.
[34] C.S.G. Lee, R.C. Gonzalez, K.S. Fu (Eds.), Tutorial on Robotics, IEEE Computer Society Press, Silver Spring, MD, 1986, ISBN 0-8186-0658-4.
[35] C.S.G. Lee, T.N. Mudge, J.L. Turney, Hierachical control structure using special purpose processors for the control of robot arms, in: C.S.G. Lee, R.C. Gonzalez, K.S. Fu (Eds.), Tutorial on Robotics, IEEE Computer Society Press, Silver Spring, MD, 1986, ISBN 0-8186-0658-4.
[36] L.I. Lieberman, M.A. Wesley, AUTOPASS: An automatic programming system for computer controlled mechanical assembly, in: C.S.G. Lee, R.C. Gonzalez, K.S. Fu (Eds.), Tutorial on Robotics, IEEE Computer Society Press, Silver Spring, MD, 1986, ISBN 0-8186-0658-4.
[37] P. Loborg, A. Törne, A layered architecture for real-time applications, in: Proceedings of the Seventh Euromicro Conference on Real-Time Systems, Odense, Denmark, IEEE Press, Piscataway, NJ, June 1995.
[38] C. Malcolm, T. Smithers, Programming assembly robots in terms of task achieving behavioural modules: First experimental result, Technical Report, DAI Research Paper no. 410, Department of Artificial Intelligence, University of Edinburgh, 1988.
[39] The MathWorks, Inc., Natick, MA, Real-time Workshop Users Guide, 1994.
[40] A. Maystel, Architectures for intelligent control systems, Discussion List: aics-l@ubvm.cc.buffalo.edu, 1992–1996.
[41] G. Messina, G. Tricomi, Standard programming tools for robots, in: Proceedings of the Third International Symposium on Measurement and Control in Robotics, IMECO Technical Committee on Robotics (TC 17), Torino, Italy, September 1993.
[42] D.J. Miller, Standards and guidelines for intelligent robotic architectures, Technical Report, Sandia National Laboratories, 1993.
[43] D.J. Miller, R.C. Lennox, An object oriented environment for robot system architectures, in: Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, 1990.
[44] F.N. Nagy, A. Siegler, Engineering Foundations of Robotics, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[45] K. Nilsson, Application Oriented Programming and Control of Industrial Robots, Lic Tech Thesis, ISRN LUTFD2/TFRT-3212-SE, Department of Automatic Control, Lund Institute of Technology, June 1992.
[46] K. Nilsson, Object oriented DSP programming, in: Proceedings of the Fourth International Conference on Signal Processing Applications and Technology, vol. 1, DSP Associates, Santa Clara, CA, September/October 1993, pp. 561–570.
[47] K. Nilsson, Software for embedded DSPs, in: Proceedings of the American Control Conference, Baltimore, MD, June–July 1994.
[48] K. Nilsson, Industrial robot programming, Ph.D. Thesis, TFRT-1046, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, May 1996.
[49] K. Nilsson, A. Blomdell, O. Laurin, Open embedded control, Real-Time Systems — The International Journal of Time Critical Computing 14 (1998) 325–343.
[50] K. Nilsson, L. Nielsen, An architecture for application oriented robot programming, in: Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 1115–1121.
[51] U. Rasper, R. Angerbauer, Heterogeneous distributed real-time achitecture for numerical and robot control, in: Proceedings of the Third International Symposium on Measurement and Control in Robotics, 1995, pp. 493–498.
[52] E. Sandewall, J. Hultman, E. Tengvald, Software architecture and programming paradigms for robotic applications, Technical Report LiTH-IDA-R-88-12, Department of Computer and Information Science, Linköping University, Linköping, Sweden, 1988.
[53] M. Schoppers, Robotic whole-body dexterity and a software architecture for robotic task performance in uncontrolled environments, Technical Report, Robotics Research Harvesting, 1993. Phase 1 SBIR Final Report under NASA contract 9-18861.
[54] M. Schoppers, The use of dynamics in an intelligent controller for a space faring rescue robot, Artificial Intelligence 73 (1995) 175–230.

[55] M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, G. Zelesnik, Abstractions for software architecture and tools to support them, IEEE Transactions on Software Engineering, March 1995, pp. 314–335.
[56] B. Shimano, C. Geschke, R. Goldman, C. Spalding, D. Scarborough, AIM: A task level control system for assembly, Robotics and Autonomous Systems 11 (1987) 45–62.
[57] D. Simon, B. Espiau, E. Castillo, K. Kapellos, Computer-aided design of a generic robot controller handling reactivity and real-time control issues, Technical Report 1801, INRIA, Sophia-Antipolis, France, November 1992.
[58] D. Simon, B. Espiau, E. Castillo, K. Kapellos, Computer-aided design of a generic robot controller handling reactivity and real-time control issues, IEEE Transactions on Control Systems Technology, December 1993, pp. 213–229.
[59] M.G. Smith, An environment for more easily programming a robot, in: Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 10–16.
[60] M.W. Spong, M. Vidyasagar, Robot Dynamics and Control, Wiley, New York, 1989.
[61] J.A. Stankovic, K. Ramamritham, Hard Real-Time Systems, IEEE Computer Society Press, Silver Spring, MD, 1988.
[62] A. Topper, B. Eng, A computing architecture for a multiple robot controller, Technical Report TR-CIM-91-5, McGill Research Centre for Intelligent Machines, McGill University, Montreal, Quebec, June 1991.
[63] T.L. Turner, J.J. Craig, W.A. Gruver, A microprocessor architecture for advanced robot control, in: Proceedings of the ISIR'84, 1984.
[64] S.G. Tzafestas, Intelligent Robotic Systems, Marcel Dekker, New York, 1991, ISBN 0-8247-8135-X.
[65] VDI-Verlag, Düsseldorf, IRDATA, VDI-Richtlinie 2863, Blatt 1, 1986.
[66] R.A. Voltz, T.N. Mudge, Robots are (nothing more than) abstract data types, in: Proceedings of the Robotics Research: The Next Five Years and Beyond, 1984.
[67] S.H. Zweben, S.H. Edwards, B.W. Weide, J.E. Hollingsworth, The effects of layering and encapsulation on software development cost and quality, IEEE Transactions on Software Engineering, March 1995, pp. 200–208.

Klas Nilsson received the M.S. degree in Mechanical Engineering from Lund Institute of Technology, Lund University, Sweden, in 1982. He then worked six years at ABB Robotics Products in Västerås, Sweden, where he developed motion control software and algorithms. Having experienced the contradiction between open/flexible systems and optimized/low-cost-products, he went back to Lund and started his Ph.D. studies at the Department of Automatic Control in Lund. His thesis, entitled "Industrial Robot Programming" (1996), pays careful attention to the interaction between programming and control of robots. During parts of 1996/97 Dr. Nilsson returned to Västerås and the work at ABB Robotics, but being settled in Lund he returned to Lund University, where he now is an Assistant Professor at the Department of Computer Science. Both within teaching and research, his main interests are software techniques for real-time systems and factory automation.

Rolf Johansson received the M.S. degree in Technical Physics in 1977, the B.M. degree in 1980, and the doctorate in 1983 (Control Theory). He was appointed Docent in 1985 and received the M.D. degree from Lund University in 1986. Since 1986 he has been with the Lund Institute of Technology, Lund University, where he is Professor in Control Theory with side appointments at the Faculty of Medicine. Currently he is Coordinating Director of robotics research with participants from several departments of Lund University. He has had visiting appointments at Laboratoire d'Automatique de Grenoble, France and CalTech, Pasadena, CA and Rice University, Houston, TX. In his scientific work he has been involved in research in adaptive system theory, mathematical modeling, system identification, signal processing theory and robotics. In 1995 Rolf Johansson was awarded the biomedical engineering prize (the Ebeling Prize) of the Swedish Society and Medicine. In 1993 he published the book *System Modeling and Identification* (Prentice Hall, Englewood Cliffs, NJ.)