Hardware/Software Transparency in Robotics Through Object Level Design[1]

by

T. N. Mudge, R. A. Volz, and D. E. Atkins
Center for Robotics and Integrated Manufacture
College of Engineering
University of Michigan
Ann Arbor, MI 48109
313/764-4343

ABSTRACT

This paper examines some computational issues in robotics. The concept of a robot-based manufacturing cell is outlined by viewing its behavior at three levels: machine level, cell integration level, and integration with higher level functions. Computational requirements associated with the first two of these three levels are discussed, and the need for very high computation rates at the machine level of a manufacturing cell, particularly for the real-time control of a robot arm and for robot vision is pointed out. The paper also explains the need to be able to manage multiple processes at the cell integration level. A unified solution to these computational needs which requires that the computation structure used to control and manage a manufacturing cell be designed as an object-based computer architecture is then proposed. The hardware/software boundary in the implementation of the objects comprising the overall architecture is transparent from a logical viewpoint but not from a timing viewpoint. Real-time constraints are met by implementing time-critical process objects directly in hardware. This "object level" design provides the hardware/software transparency necessary to formulate a unified system specification for these real-time embedded computer systems. Timing considerations can then be used determine the hardware/software boundary.

I. INTRODUCTION

The factories of the future, in fact the only factories with a future, are those which are moving to increase productivity through greater automation of design, processing, assembly, and management [1]. The University of Michigan has recently made a major commitment to the technological research and education necessary to achieve this goal. A Center for Robotics and Integrated Manufacturing (CRIM) has been established in the College of Engineering to coordinate and focus the research activities of 30 faculty and 50 staff and research assistants on problems of industrial automation [2]. Complex distributed computer systems are central to automated or integrated manufacturing and consequently the computer science and engineering communities at the University of Michigan and elsewhere are focusing more and more attention on applying state-of-the-art computer related technologies to, as well as developing new technologies for, advanced industrial automation. Indeed, results of this transfer of technology can already be seen in the area of industrial inspection, an important part of manufacturing, where inexpensive rugged CCD solid-state cameras coupled with microprocessors have made feasible a great in the level of automation [3]. In addition to benefiting from a flow of ideas from computer related technologies, the field of robotics provides many challenging problems which may provide the basis for fundamental breakthroughs in computer science/engineering. Notable among these problems are the management of multiple processes in a real-time environment and the implementation of these processes so that they meet the real-time constraints.

Of paramount importance in future manufacturing systems is the concept of a robot-based manufacturing cell composed of a number of different material processing machines together with material handling facilities. Computer control is central to the operation of these manufacturing cells. Less obvious, but also of significant importance, is a coupling of these cells to the outputs of computer aided engineering programs, e.g. geometric information in CAD databases. Computer control combined with "general purpose" robots gives the manufacturing cell a high degree of flexibility through the ability to reprogram the operation of the cell. This enables the cell to function in a cost effective way in manufacturing environments that do not fulfill the usual notion of mass production. An example of this is batch assembly which represents one third of all manufacturing in the United States [4].

While there exist a few examples of automated manufacturing cells [5], they are special purpose cells and limited in their flexibility. In this paper we sketch a proposal for developing the hardware/software structure needed to control a robot-based manufacturing cell. The aim is to develop a methodology which:

- Can be applied across a large class of manufacturing cells.
- Results in systems that are easily extensible if additional sensors, robots, and/or machines are added.
- Results in systems in which the hardware/software boundary is transparent.

[1] This work was supported in part by a grant from the Zimmer Foundation, Ann Arbor, MI 48103, and National Science Foundation grants ECS-8108954 and 8007298.

Proceedings of the Society of Photo-optical Instrumentation Engineers Technical Symposium West, August 1982.

The need for such a general methodology is enormous. As long as automated manufacturing cells are only produced on a one or two of a kind special purpose basis, there will be relatively few such cells in existence. The need for hardware/software transparency is particularly important during the design of a cell where, in order to meet real-time constraints, it may be necessary to move software implementations into special purpose hardware. If this can be done transparently it will facilitate system development, since the task of logical system development can occur without regard to the actual method of implementation. For similar reasons hardware/software transparency is important if extensions to the a system are to be easily accommodated--altered requirements may tighten certain timing constraints necessitating a migration of software into hardware.

In this paper we define computational issues in robotics with particular emphasis on the use of processors which are tailored to specific computational tasks. Our hypothesis is that "object-based computer systems" are excellent candidates for the framework to accommodate the evolution of the complex hardware and software required to meet these computational needs. In particular such systems provide a controlled means to enhance performance of an existing robotics control system by the introduction of special-purpose (probably VLSI) components in a manner transparent to the user. Hardware and software can be tied together in an evolutionary way.

This paper is organized as follows: The next section illustrates the concept of a robot-based manufacturing cell. Section III explains the notion of an object-based computer architecture and how it fits into the real-time control of a manufacturing cell. Section IV identifies possible time-critical computations that might arise in a robot-based manufacturing cell. Finally, Section V concludes by noting how the proposed design approach fits well with the programming language Ada, a language expressly designed for programming real-time embedded systems.

II. Robot-Based Manufacturing Cells

Robot-based manufacturing cells have many different configurations; however, their operation may be explained by viewing their behavior at three levels:

- Machine Level.
- Cell Integration Level.
- Intergration with Higher Level Functions.

Of critical importance in all three levels are the computer systems used to control the cell, their interconnection, and their programming.

At the machine level there will be a variety of processing and material handling machines. Typical machines will include numerically controlled lathes, drills, milling machines, presses, forges, and bending machines. Material handling systems will include robots, conveyors, mobile carts (with attached robots). Robots of course have a dual nature and may be considered processing machinery if they are used for assembly. Also at the machine level will be various types of sensors which provide information by which machine guidance and control may be determined. Each of these machines or sensors will require individual computer attention.

Computer based integration of these machines and their integration with higher level engineering functions offer a significant increase in manufacturing productivity. Cell control may be hierarchically organized. Each machine or sensor will have its own computer process controlling it. These processes will in turn communicate with a supervisor process responsible for overall coordination of cell activity. Computerization of the engineering functions associated with manufacturing is rapidly taking place. Computer aided design is already a one billion dollar a year business, and work is progressing on computer aided process planning, material requirements planning, and shop floor scheduling. Eventually, the manufacturing cell will be coupled to the output of these processes. It is already possible to automatically generate NC machines programs from a CAD database description of a part.

To illustrate the above concepts Figure 1 shows a hypothetical manufacturing cell. In this particular cell material handling is performed by the conveyor at the left, the inventory storage retrieval system at the top, the robots R1 and R2, and the conveyor at the bottom. Material processing is performed by the NC machine at the right and by R2 in the assembly area (R2 has a dual role). Parts flow in on a conveyor (the one at the left) where they are inspected by the camera C1. This sensor identifies each part so that the appropriate processing/handling sequence can be initiated for that part. Robot, R1, removes each part from the conveyor with the help of position information collected by C1. Depending on the type of the part and the state of the cell, the part may be temporarily stored in the inventory storage retrieval system to smooth the flow of material through the cell. For example, if machine failures require succeeding cells to be shut down for repairs R1 can still serve the incoming conveyor by simply storing each part. Alternatively, R1 may load the part into the NC machine for a sequence of machining operations. R1 is also responsible for changing tools in the NC machine--when not in use tools are stored in the tool magazine (M). After machining, the part is moved, by R1, to the assembly area where it is incorporated into a subassembly by the robot R2. The completed subassembly is placed on the outgoing conveyor (at the bottom) by R2 and transported to the next manufacturing cell. The assembly process, performed by R2, is achieved with the aid of vision information from camera C2 and touch and force information from pressure and force sensors in the grippers and the wrist of R2. In the case where parts arrive into the cell in an inappropriate sequence for assembly, they can be resequenced through the storage system. Additionally, if the flow of parts from preceding cells temporarily ceases, for whatever reason, the utilization rate of the machines in the cell can be maintained by retrieving parts from storage.

From the above machine level description of the hypothetical manufacturing cell it becomes clear that there are a number of interlocking processes to manage. This is done at the cell integration level. In Figure 1 a "cell management computer" is shown that performs this management function. The data rates required to provide local control to components at the machine level of the cell dictate the need for special purpose processors. These are shown in Figure 1 as attached processors (AP's) to the cell management computer. The cell management computer and its associated AP's should be viewed as a single computation structure. Indeed, our proposal suggests that the structure be defined as a single object-based architecture in which the hardware/software boundary is logically transparent. The boundary, and hence the placement of the AP's, would be defined by examining time-critical processes in the cell.

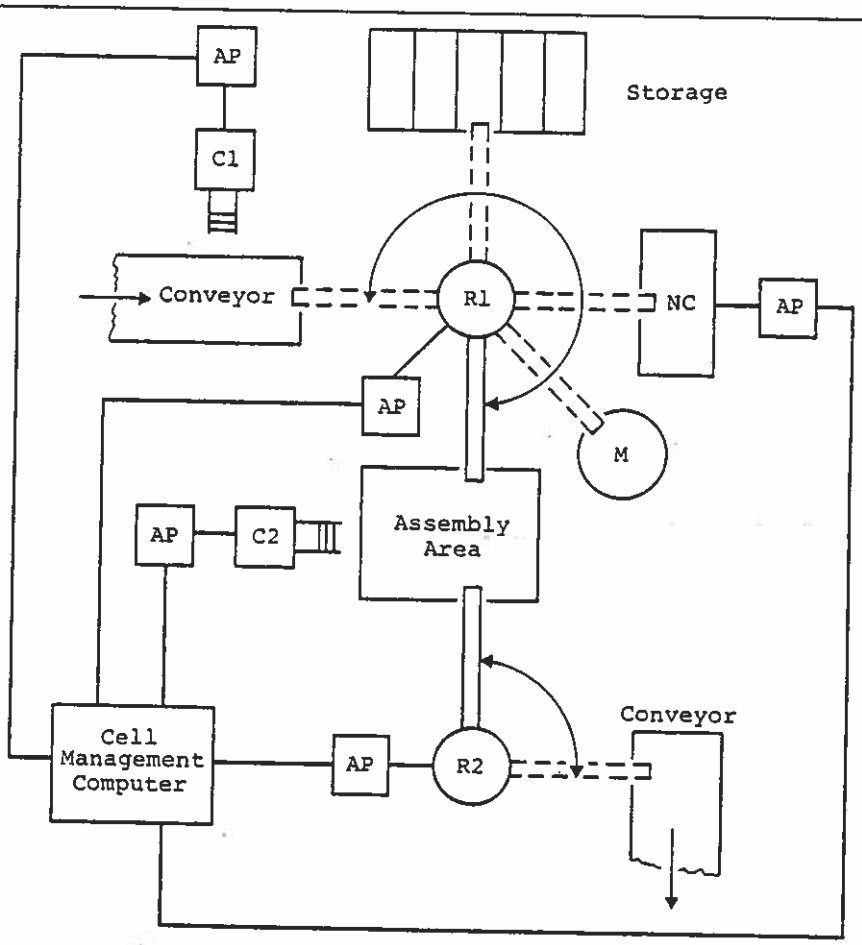

Figure 1. Robot-based Manufacturing Cell.

The cell management computer is also responsible for communicating with computers at the factory management level. The factory management level computers integrate with the CAD system database that describes the product which the various manufacturing cells are cooperating to produce.

III. Object-based Computer Architectures

Object-based computer architectures are a synthesis of two parallel developments in the fields of language and operating systems research. While language researchers were studying the problem of program decomposition, their counterparts in operating systems research were studying problems associated with data integrity and security. A consensus of opinion from language research directed at program decomposition emerged in favor of the use of information hiding (also referred to as data encapsulation or package-based decomposition) as the major criterion for the modular decomposition of computer programs [6]. This methodology employs the concept of a type manager to manage and effectively hide the implementation of abstract data types from other the modules in the system [7]. Meanwhile, in operating systems research, a consensus of opinion developed in favor of the concept of protection domains as a solution to the data integrity and security problem. This concept is analogous to the package based decomposition model [8].

In the type manager concept each package is divided into two major parts, a specification and a body (shown in Figure 2). The specification contains the explicit description of the portion of the package which is visible to other modules (packages). This usually consists of descriptions of procedures which operate on the "hidden" data abstractions, but can also include, in the case of external type managers (described later), type definitions which define the explicit data abstractions on which this particular type manager will operate. The body, on the other hand, contains the actual procedure code along with the descriptions of the "hidden" data abstractions which remain internal to the package. Other than granting procedure calls and data accesses to items described in the specification, the contents of the package body is inaccessible to all other modules (shown in Figure 3). External type managers are those type managers in which the actual copies of the data abstractions to be manipulated exist in other packages. The description of these data abstractions, however, is still given in the type manager's specification. Further, it is expected that all procedures which

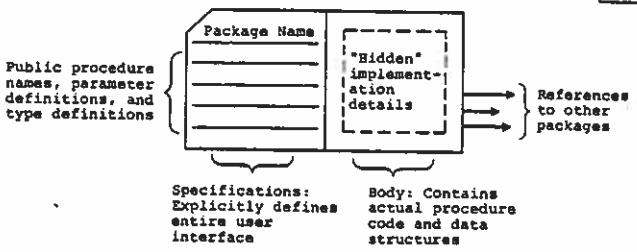

Figure 2. Package-based Decomposition.

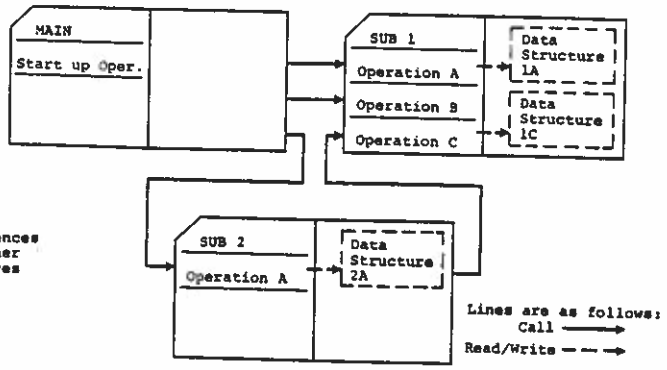

Figure 3. Interconnection of packages.

manipulate this data reside in the type manager package. This maintains the consistency with the notion that the implementation of an abstract data type need only be known to its specific type manager.

Protections domains are realized by grouping all of the data in the system into items called "objects" [9], and then employing a capability-based addressing scheme [10] to reference these objects. So, for example, there will be objects which represent the instruction code of procedures, activation records of executing processes, and even abstractions of physical processors. Hence the phrase "object-based computer architecture." Of course, the type managers and data abstractions mentioned above will also be represented by objects. To enforce these protection domains, all objects in the system are required to be referenced with capabilities.

To illustrate the concept of capabilities consider their implementation in Intel's iAPX 432, a commercially available object-based system (in Intel's terminology capabilities are commonly referred to as access descriptors). Access descriptors provide indexes into an object table directory and one of its corresponding object tables, they also specify the read/write privileges for a particular access (see Figure 4). The object table entries not only contain the physical memory addresses and swapped/not swapped bits pertaining to their respective objects, but also the specific type and length of the objects as well. All of this information is directly accessible by the hardware and is, in fact, used to dynamically check object types, access restrictions, and length bounds. This effectively allows the protection mechanism to

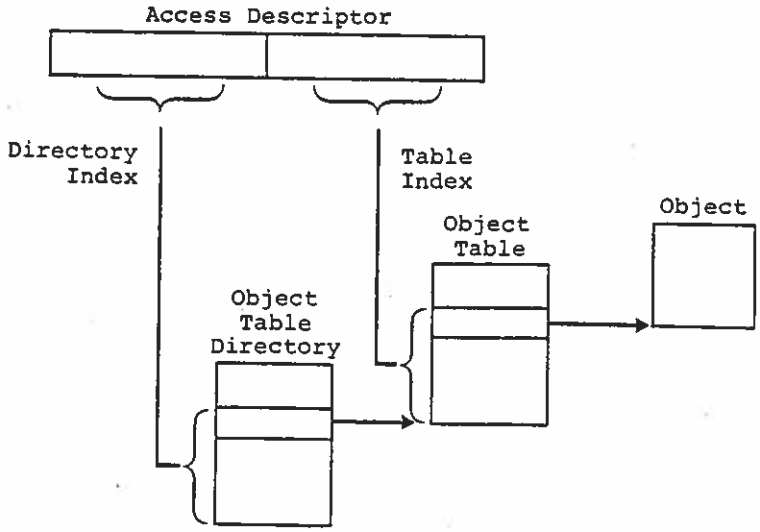

Figure 4. Intel's Capability Addressing Model.

extend down to objects which may represent items as small as single variables. Since there are no machine instructions which allow memory to be referenced by physical addresses, capability addressing is forced upon all users, including the operating system. This eliminates the need for separate privileged/user execution modes, hence all modules are given only the privileges necessary to do their jobs. This is in direct contrast to conventional systems where processes requiring more than the limited access available in user mode must be given full access to the entire system.

The iAPX 432 architecture is said to be object-based not only because it provides support for the concepts outlined above, but because it also incorporates those same philosophies and methodologies into its actual design. System objects which represent entities such as: processors, processes, activation records, dispatching queues, and free storage act as complex operands for very powerful machine instructions. The concept of system objects naturally extends to allow the incorporation of special purpose attached processors via their corresponding processor objects [11]. The object oriented machine instructions provide the mechanisms for implementing the object based concepts, while the operating system software provides for its policy and control. The packages which comprise the operating system are, literally, type managers for the system resources mechanisms which are provided by the hardware. This consistency of design philosophy between hardware (including special purpose AP's), operating system, and user software provides the transparency necessary for total system integration.

IV. Real-time Computing Requirements For Robotics

In this section we shall discuss some time-critical computations frequently required in the control of robot-based manufacturing cells. There are two major classes of computations that fall into the time-critical category: robot arm control computations and robot vision computations. Therefore, depending on the level of sophistication of the control strategy and the vision algorithms, it may be necessary to implement these processes directly in special purpose hardware (the AP's of Figure 1). Continuing developments in VLSI technology coupled with the promise of "silicon compliers" make "custom" special purpose processors attractive for the future; presently, however, these processors should be thought of as being implemented with off-the-shelf, possibly high-speed, components. In either case, an important concern is the integration of these processors into the overall cell management system. If the system is designed as an object-based architecture this integration is accomplished as a by-product of the resulting hardware/software transparency.

Robot Arm Control

Controlling a robot arm is a complex task that can be conveniently divided into the following four stages:

(1) Trajectory planning--in this stage the arm's path through space is determined. Depending on the application this may involve obstacle avoidance strategies.

(2) Resolved motion--in this stage the arm's trajectory is resolved into the component joint motions.

(3) Gross motion--in this stage torques and forces are derived that are required by the joint actuators to generate the joint motions computed in stage 2. Typically these torques and forces form the basis of a control law that incorporates some type of negative feedback.

(4) Fine motion--in this stage the torques and forces that are required to generate the incremental joint motions necessary once the arm is close to its goal are derived.

The purpose of control stages 2, 3, and 4 is to maintain the motion of the arm along the trajectory derived in stage 1 by applying corrective compensation through the actuators to adjust for any deviations of the arm from the desired arm trajectory. If a "perfect physical model" for the arm could be defined and if the model could be "solved" rapidly enough to output control signals at a rate compatible with the desired arm motion there would be no need for feedback in the control strategy. A "perfect physical model" would, however, have to be one which accounted for, among other things, gravitational and inertial loading, friction, link flex, not to mention all possible external perturbances. Clearly, such a model requires an impossible amount of computation. Indeed, in view of the need to include external perturbances, just defining the model is impossible.

Although a "perfect physical model" is clearly out of the question, in our opinion, one improvement to the present shortcomings of arm control is to include more of the physics of the arm in the model. The physics of the arm is well understood [12], and [13], [14]. A more accurate model in the control loop offers greater versatility than presently used control techniques. One can determine the arm's gravitational loading, inertial loading, friction, and can also model arm flex, allowing the use of much lighter and more efficient arm designs. But the technique goes further. Inputs from pressure and force sensors in the grippers and the wrist of the arm are easily incorporated into the model. This incorporation allows the model to adjust for the gravitational and inertial loading of the payload and to react to external forces and moments. As an offshoot of the preceding, during the fine motion stage of control, false or "pseudo-forces" and "pseudo-moments" can be artificially generated and summed into the sensor inputs from the wrist. The arm will exert forces or moments to compensate for these pseudo-force and -moment inputs. Thus, the arm can be made to exert any desired force and moment vectors by incorporating oppositely directed pseudo-force and -moment vectors into the hand inputs [14], [15].

If a more accurate model of the arm is used the amount of computation associated with controlling the arm can be a serious obstacle to meeting real-time constraints. However, we have shown that current VLSI technology will allow the fabrication of cost-effective special purpose processors that can overcome this difficulty. In particular, we have shown that if the control strategy is based on a model employing the Newton-Euler equations of motion real-time constraints can easily be met assuming control stage 1 is computed off-line [15]-[17].

In the context of our proposal for a computation structure to control manufacturing cells the computations associated with stages 2 through 4 would, typically, be handled by AP's and stage 1 would be handled by the cell management computer. Of course, the overall control strategy (all stages) would be developed as a single integrated unit and the assignment to hardware would only come after timing considerations have been taken into account. To illustrate the

potential for robot arm control to be a time-critical function consider the following: If the Newton-Euler based control strategy mentioned above is employed it has been shown that to control a PUMA 600 robot requires about 400 multiplicative operations and 450 additive operations to be performed in less than a millisecond [15].

Robot Vision

The special computational requirements of vision algorithms, particularly the high processing rates and the large data sets of predominately two dimensional arrays, can be best met by a computer architecture that has been tailored to those requirements. The objective in a manufacturing environment is to develop a vision subsystem capable of performing object segmentation based on a set of features relevant to a wide range of applications, supplying real-time sensory feedback information, and functioning effectively in the noisy environments encountered in industrial applications.

Past work in the computer vision area has resulted in several experimental systems for industrial robots as well as a few cost-effective commercial systems [18]-[20]. The major problems in the design of these systems have been segmentation, object location, object orientation and object recognition. A typical vision system consists of the following processing stages:

(1) Image acquisition of one or more views of the object area. These views are usually obtained in the visible light spectrum by a video sensor, such as a CCD device, operating at standard NTSC video rates. Other parts of the light spectrum have been used for computer vision such as the infrared region. In some cases laser ranging devices or structured lighting is also used for obtaining depth information.

(2) Image preprocessing to remove salt-and-pepper noise, filtering to correct uneven lighting, other forms of enhancement.

(3) Feature extraction, typically segmentation operations. The features used include gray-level edges, texture edges, shape descriptors such as a Fourier shape descriptors, area of objects, perimeter, depth from stereo views, depth from structured lighting, depth from ultrasonic or laser ranging, shape-from-shading, and many others. The features extracted are highly application dependent.

(4) Classification or feature evaluation with respect to object identification, position, and orientation. In the past most decision procedures were based on pattern recognition concepts using either statistical or syntactic pattern recognition. These techniques have limitations with regard to performance and ability to handle a large number of features. The classification operation is a higher level operation that has no real fixed format and in many ways this is the most difficult step in the vision operation.

There has been considerable interest in using artificial intelligence concepts to interpret or evaluate the features. This is the problem of "image analysis" that DARPA has been addressing for several years [21]. We believe that AI concepts combined with techniques capable of handling a large number of features efficiently will have to be used in solving this very complicated problem. The use of "expert systems" is now being considered as a strong candidate for the general scene interpretation problem.

A number of articles have appeared in recent years that identify the particular computational needs of computer vision and the related area of image processing [22]-[25]. An important point generally agreed upon is that computer vision algorithms require very high computation rates if they are to be done in a "reasonable" time. It has been estimated [25] that processor speeds on the order of 1 to 100 billion operations per second will be required to solve some of the current problems in computer vision. While the current trend towards "massively parallel" architectures for vision affords a solution, it raises the issue as to what algorithms can be implemented on such architectures. Algorithms such as correlation matching, histogramming, and edge thinning are highly parallel operations and have been implemented on various architectures [26]. Other algorithms, including many feature extraction algorithms, do not exhibit a high degree of parallelism.

In the context of our proposal for a computation structure to control manufacturing cells the highly parallel computations would, typically, be handled by AP's. Higher level functions which do not exhibit much parallelism or require large data sets, such as some of those found in feature extraction and classification algorithms, would be handled by the cell management computer. Of course, the vision algorithms would be developed as part of the overall manufacturing cell computation structure and the assignment to hardware would only come after timing consideration had been taken into account.

V. Conclusion

This paper has proposed a unified solution to the design of the computation structure used to control and manage a manufacturing. The solution proposes that the cell be designed as an object-based computer architecture. The hardware/software boundary in the implementation of the objects comprising the overall architecture is transparent from a logical viewpoint but not from a timing viewpoint. Real-time constraints are met by implementing time-critical process objects directly in hardware. This "object level" design provides the hardware/software transparency necessary to formulate a unified system specification for these real-time embedded computer systems. Timing considerations can then be used determine the hardware/software boundary.

The system design philosophy proposed in this paper is based on many of the concepts implicit in the design of the Intel 432 micromainframe. In addition, the object level design, as we have outlined it, is consistent with much of the underlying philosophy of the programming language Ada. It is interesting to note that Ada was developed on behalf of the Department of Defense as a standard for use in military applications requiring embedded real-time systems [27]. This development was motivated, in part, by a need to contain costs on large and complex software development projects.

It can be seen from the earlier characterization of a manufacturing cell that it is a typical example of a real-time embedded system requiring a large and complex software development effort. Therefore, it is our hypothesis that the use of the Ada language in conjunction with an object based hardware architecture will provide an environment

conducive to the development of the entire (hardware/software) cell control architecture. Current research underway at the University of Michigan is directed toward demonstrating this hypothesis.

REFERENCES

[1] J. A. Baker, "The factory with a future," *General Electric Executive Speech Reprint*, March 1982.

[2] *Robotics Research and Education at the University of Michigan*, Feb. 1982.

[3] J. F. Jarvis, "Visual inspection automation," *IEEE Computer*, vol. 13, pp. 32-38, May 1980.

[4] J. Evan, J. Albus, and A. Barbera, "Robot loading of an NC machine tool," *1977 Joint Automatic Control Conference*, 1977.

[5] L. Hissing, Vice-Pres. Sandvik Corp., Private Communication, Ann Arbor, MI 48109, April 7-th, 1982.

[6] B. Parhami, "A highly parallel computing system for information retrieval," *AFIPS Conf. Prod.*, pp. 681-690.

[7] M. Shaw, "The impact of abstraction concerns on modern programming languages," *Proc. of the IEEE*, vol. 68, no. 9, pp. 1119-1130, Sep. 1980.

[8] *Introduction to the iAPX 432 architecture*, Santa Clara, CA 95051: Intel Corp., 1981.

[9] G. J. Myers, *Advances in computer architecture*, New York: John Wiley & Sons, 1982.

[10] R. S. Fabry, "Capability-Based addressing," *Comm. of the ACM*, vol. 17, no. 7, pp. 403-412, July 1974.

[11] J. R. Rattner, "Functional extensibility: Making the world safe for VLSI," in *VLSI Systems and Computations, Eds: H. T. Kung et al.*, Pittsburgh: Computer Science Press, 1982.

[12] R. A. Lewis, "Autonomous manipulation on a robot: Summary of manipulation software functions," Technical Memo 33-679, JPL, March 1974.

[13] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *Trans. of the ASME: Jour. of Dynamic Systems, Measurement, and Control*, vol. 120, pp. 69-76, June 1980.

[14] J. L. Turney, T. N. Mudge, and C. S. G. Lee, "Connection between robot arm dynamic formulation with applications to simulation and control," Center for Research in Integrated Manufacturing Report RSD-TR-4-82, Univ. of Michigan, Ann Arbor, MI, Nov. 1981.

[15] T. N. Mudge and J. L. Turney, "Unifying Robot Arm Control," *Proc. of the 1982 Ann. Meeting of the Industry Applications Society*, Oct. 1982.

[16] C. S. Lee, T. N. Mudge, and J. L. Turney, "Hierarchical control structure using special purpose processors for the control of robot arms," *Proc. of the IEEE Computer Society Conf. on Pattern Recognition and Image Processing*, pp. 634-640, June 1982.

[17] J. L. Turney and T. N. Mudge, "VLSI implementation of a numerical processor for robotics," *Proc. of the 27th Int. Instrumentation Symp.*, pp. 169-175, April 1981, also presented at the Instrument Society of America Anaheim Conf., Oct. 1981.

[18] S. W. Holland, *et al.*, "A Vision Controlled Robot for Part Transfer," *Research Publication GMR-3120*, General Motors Research Laboratories, Warren, MI.

[19] G. J. Agin, "Computer vision systems for industrial inspection and assembly," *IEEE Computer*, vol. 13, no. 5, pp. 11-20, May 1980.

[20] E. J. Lerner, "Computers that see," *IEEE Spectrum*, vol. 17, no. 10, Oct. 1980.

[21] *Proc. of the DARPA Image Understanding Workshop*, 1977, 1978, 1979, 1980.

[22] A. P. Reeves, "Parallel computer architectures," *Proc. 1981 Conf. Parallel Processing*, pp. 199-208.

[23] E. J. Delp, T. N. Mudge, L. J. Siegel, and H. J. Siegel, "Parallel processing for computer vision," *Proc. of the Society of Photo-optical Instrumentation Engineers Technical Symposium East*, May 1982.

[24] T. N. Mudge and E. J. Delp, "Special purpose architectures for computer vision," *Proc. 15-th Hawaii Int. Conf. on Syst. Sci.*, Jan. 1982.

[25] D. R. Reddy and R. W. Hon, "Computer architectures for vision," in *Computer Vision and Sensor-Based Robots*, G. G. Dodd and L. Rossol, eds. Plenum Press, New York, 1979, pp. 169-186.

[26] H. J. Seigel, L. J. Seigel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. on Computers*, vol. C-30, Dec. 1981.

[27] J. G. P. Barnes, *Programming in Ada*. London, England: Addison-Wesley, 1982.