Design Lessons for Building Agile Manufacturing Systems

Wyatt S. Newman, *Member, IEEE*, Andy Podgurski, *Member, IEEE*, Roger D. Quinn, Frank L. Merat, *Member, IEEE*, Michael S. Branicky, *Member, IEEE*, Nick A. Barendt, *Member, IEEE*, Greg C. Causey, Erin L. Haaser, Yoohwan Kim, Jayendran Swaminathan, and Virgilio B. Velasco, Jr.

***Abstract*—This paper summarizes results of a five-year, multidisciplinary, university-industry collaborative effort investigating design issues in agile manufacturing. The focus of this project is specifically on light mechanical assembly, with the demand that new assembly tasks be implementable quickly, economically, and effectively. Key to achieving these goals is the ease of equipment and software reuse. Design choices for both hardware and software must strike a balance between the inflexibility of special-purpose designs and the impracticality of overly general designs. We review both our physical and software design choices and make recommendations for the design of agile manufacturing systems.**

***Index Terms*—Agile manufacturing, automated assembly, flexible automation, rapid-response manufacturing, robotic assembly, workcell.**

I. Introduction

THIS paper is based on results obtained and lessons learned over five years of recent multidisciplinary research in agile manufacturing at Case Western Reserve University (CWRU) [1]–[14], [34]. Our effort has included mechanical engineering, electrical engineering, and computer science in a collaboration between industry and academia with a common goal of achieving rapid implementation of automation for light mechanical assembly applications. This paper reviews our progress, particularly beyond that reported earlier in [3].

We note that the definition of agile manufacturing varies widely in the literature (see, e.g., discussion in [3]). In the present context, we refer to flexible manufacturing as the ability to reconfigure a system quickly and cheaply to assemble a varying part mix. Conventionally, flexibility is restricted to a suite of previously (possibly laboriously) developed applications. To incorporate agility, we invoke a design philosophy that promotes hardware and software reuse, enabling rapid redesign for entirely new applications. Agility of this type offers the promise of achieving responsive, economical automation of batch production or short life-cycle product manufacturing by reusing generic production software and equipment.

The CWRU experimental workcell is shown in Fig. 1. The work described here has been implemented on commercially-available components within this workcell to validate our assumptions and proposed concepts within an industrially applicable system.

Our experimental workcell includes four robotic workstations, a flexible material handling system, two machine vision systems, flexible parts feeders, eight cameras, and more than 200 digital input/output (I/O) signals. In our experience, this system is sufficiently complex to illuminate the challenges and evaluate the solutions in agile manufacturing system design. While the work described here is on-going research, the lessons learned have immediate applications.

Many philosophical choices in the design of an agile manufacturing system have implications for its usability. As the detail and complexity of the system can be large, it is important to choose the appropriate levels of abstraction for interaction and organize the system itself to encapsulate much of the complexity. Lessons learned from our research are condensed here as design guidelines for agile manufacturing systems and grouped into two categories. The first category is a collection of hardware design techniques that enable reuse and rapid reconfiguration. The second category concerns software design recommendations for an agile manufacturing control system that is maintainable, reusable, and extensible. While the hardware and the control software are inherently coupled with respect to global optimization, we have deliberately prescribed modularity and interfaces of the hardware and software to permit encapsulation of details, promoting decoupling of design constraints as much as possible. Optimization is thus with respect to ease of redeployment, rather than with respect to performance within a specific application.

Manuscript received September 17, 1998; revised April 16, 1999. This paper was recommended for publication by Associate Editor P. Banerjee and Editor P. Luh upon evaluation of the reviewers' comments. This work was supported by the Cleveland Advanced Manufacturing Program (CAMP, Inc.), the Center for Automation and Intelligent Systems Research, the Case School of Engineering, and their industrial sponsors for this work.

W. S. Newman, A. Podgurski, F. L. Merat, M. S. Branicky, and E. L. Haaser are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106 USA.

R. D. Quinn and G. C. Causey are with the Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland, OH 44106 USA.

N. A. Barendt was with the Center for Automation and Intelligent Systems Research at Case Western Reserve University, Cleveland, OH USA. He is now with the Nordson Corporation, Amherst, OH 44001 USA.

Y. Kim was with the Center for Automation and Intelligent Systems Research at Case Western Reserve University, Cleveland, OH USA. He is now with the Wireless Data Networking Laboratory, Lucent Technologies Bell Laboratories, Whippany, NJ 07981-0903 USA.

J. Swaminathan was with the Center for Automation and Intelligent Systems Research at Case Western Reserve University, Cleveland, OH USA. He is now with the US-Windows CE Platforms, Microsoft Corporation, Redmond, WA 98052-6399 USA.

V. B. Velasco was with the Center for Automation and Intelligent Systems Research at Case Western Reserve University, Cleveland, OH USA. He is now at 7019 S. Inglenook Cove, Apt. 1302, Midvale, UT 84047-5344 USA.

Publisher Item Identifier S 1042-296X(00)04695-4.

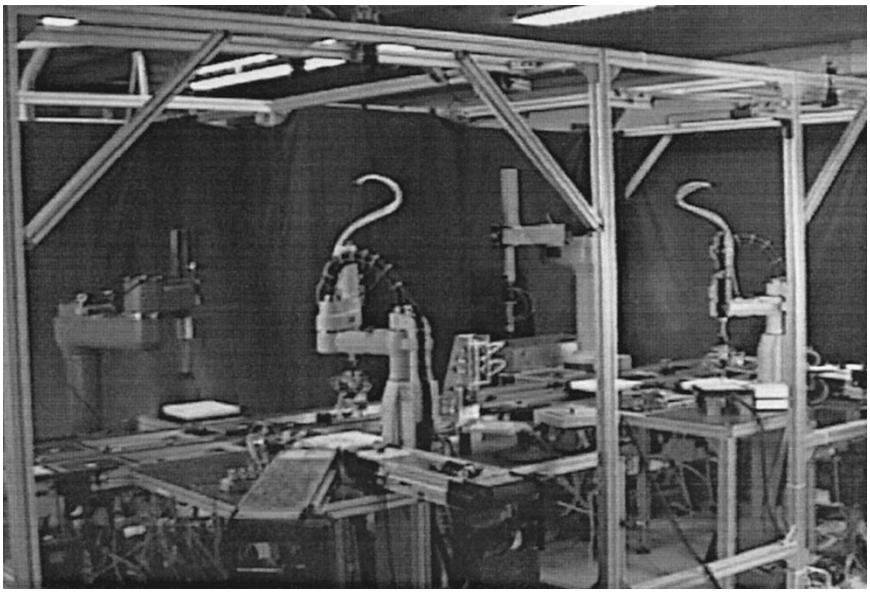

Fig. 1. CWRU agile manufacturing cell.

II. Hardware Design Techniques Enabling Rapid Reconfiguration

Robots are representative of an ideal element of agile manufacturing systems. They can perform a variety of manipulation functions with little or no hardware changes, and they offer the possibility of rapid reprogramming for new applications. These two features (equipment reuse and rapid redeployment) are consistent with our requirements for agile manufacturing. However, robots are just one component of an agile manufacturing system. In general, one also needs parts feeders, grippers and fixtures, intra-workcell transportation, sensors, and assembly-specific hardware.

In designing the automation elements to meet these requirements, there is an inevitable tradeoff. On the one hand, there is the appeal of designing clever, high-speed and/or low-cost components to handle specific parts. On the other hand, there is the desire to make each element of the agile system completely general and reusable. The former approach is used conventionally in design of high-volume automation systems, but it is not agile (i.e., it requires extensive change to the system when a new product is introduced). The latter approach can lead to impractically complex and expensive components. Our recommendations for reconciling these competing viewpoints are presented below.

A. Agile Parts Feeding

Parts feeding in industrial automation is typically performed with vibratory bowl feeders. Such feeders are custom designed for each new part to be fed, and the design effort invested in each bowl feeder can only be recovered over the life of the one part it was designed to present. In addition, the design of bowl feeders typically requires months, and this design process cannot begin until a sufficient quantity of the parts to be fed is available for experimentation. Thus, the lead time for design of bowl feeders cannot be accommodated in simultaneous product and process development.

At the other extreme in parts feeding is the generic bin-picking problem [15], [16]. In this very general approach, one uses three-dimensional (3-D) sensing, pattern recognition in cluttered scenes with overlapping parts, 6-DOF manipulation, and sophisticated approach and grasp planning. As a consequence, the result is expensive, difficult to program and maintain, and unreliable.

Recently, an agile alternative—a compromise between the specificity of bowl feeders and the generality of bin picking—has been promoted. “Flexible parts feeders” have been described in [17]–[33]. We advocate this approach for agile manufacturing. In our own system, we have constructed a form of flexible parts feeder that relaxes some of the restrictions of previous feeders (see, e.g., [3]). In this system, parts conveyors are used to present parts to an underlit surface. A vision system identifies parts that are easy to recognize and easy to grasp robotically, and the corresponding coordinates are communicated to a robot for acquisition. Parts that are hard to identify or hard to grasp (e.g., due to overlapping) are recycled to the conveyor system for repeated presentation trials.

Our system permits a larger feed volume and can accommodate larger parts than similar commercially-available systems (e.g., Adept Technology’s FlexFeeder [33]). In addition, our feeder (with variations currently in progress) can be adapted to accommodate rolling parts. Our feeder has been tested successfully on a wide variety of part shapes, including plastic disks, plastic snap-rings, cup-shaped objects, hex nuts, washers, caps, and plastic sockets. At present, the feeder speed (including time for robot acquisition) is up to 30 parts/min (depending on part size and geometry). We have validated the system in feeding trials of 750 000 parts, including a 400-h unattended continuous run, demonstrating high reliability. (Further detail on our flexible feeder design and performance is given in [6] and [34].)

Our flexible parts feeder is agile, since it can handle a wide variety of part shapes by merely reprogramming the part-recognition vision code. Based on our observations of reprogrammability, flexibility, reliability, and throughput, we conclude that

this approach to parts feeding is appropriate for agile manufacturing.

B. Agile Grippers and Fixtures

Another example of competition between dedicated solutions and general solutions in agile manufacturing is in the design of grippers and fixtures. The most common robot gripper in practice is the simple parallel-jaw, which is suitable for handling objects with parallel planar faces. More frequently, part shapes are marginally graspable between parallel planes, resulting in frequently dropped parts and imprecisely manipulated parts. In contrast, there exist very general anthropomorphic hands [35]. While these devices are capable of a much broader range of grasp alternatives, they are expensive, relatively fragile, and difficult to program and maintain.

To date, nearly all of the grippers used in our agile manufacturing system have been custom designed and machined, which does not satisfy our criteria for agility. (We do, however, use some conventional techniques, such as quick connectors and rotary wrists, and design some grippers to handle multiple parts.) To address the need for agility in grippers and fixtures, we have proposed and demonstrated a compromise approach that achieves high performance with low cost and quick implementation. Our concept is to utilize a standard parallel-jaw mechanism with custom-designed fingers, where the finger shapes are designed and fabricated automatically via CAD/CAM. (Note that this CAD/CAM process is decoupled from the rest of the system design, is performed off line, and results in a gripper that, with respect to the control software, is functionally indistinguishable from a parallel-jaw gripper.) Our approach, which is detailed in [9] and [10], is summarized as follows.

We presume the existence of a CAD description of the parts to be handled. We start with a default shape for the gripper fingers defined as solid blocks, represented in a compatible CAD description. The part to be handled is used to define the equivalent of a ram-EDM (electronic discharge machining) tool, and this postulated tool is advanced (computationally) into the faces of the default finger blocks, producing cavities comprising a shape complementary to the part to be grasped. This computational geometry operation produces a CAD description of fingertips that nearly envelop the desired part.

By construction, the resulting finger cavities are guaranteed to permit collision-free opening and closing of the gripper fingers with respect to the part (analogous to the parting line of a casting mold). Once the CAD description of the gripper fingers is prescribed, it can be fabricated automatically by any compatible CAM process. Since the computed finger cavities are guaranteed to have outward-facing surface normals, the cavities are guaranteed to be machinable with a 3-axis mill (with one setup). Our experimental approach to fabricating the custom gripper fingers has been to use laser cutting in a sheet-based rapid prototyping system, details of which can be found in [36] and [37].

We have shown that it is possible to define and fabricate multifunction grippers by repeating the computational geometry process multiple times on the same finger blocks. For a relatively small number of distinct parts (we typically feed no more than four part shapes at each robotic workstation), one can produce a composite finger cavity shape by treating each of the desired parts as a separate virtual EDM tool. If each virtual EDM operation (computationally) erodes part of the candidate multifunction gripper finger, and if each such erosion operation contributes significantly to the resultant shape of the finger cavity, then the resultant fingers are strong candidates for achieving form closure [38] (or at least frictional form closure [39]). Candidate shapes thus computed can be tested computationally to determine if they would be successful gripper fingers with respect to each of the parts to be handled, e.g., using the methods described in [40]. A simple, experimental example of this approach is shown in Fig. 2. Three parts (two sizes of hex nuts and a plastic socket), as shown, are all graspable in frictional form closure by our example computed and fabricated gripper fingers. Experiments showed that this gripper manipulated these three parts with greater precision than simple parallel-jaw fingers [10].

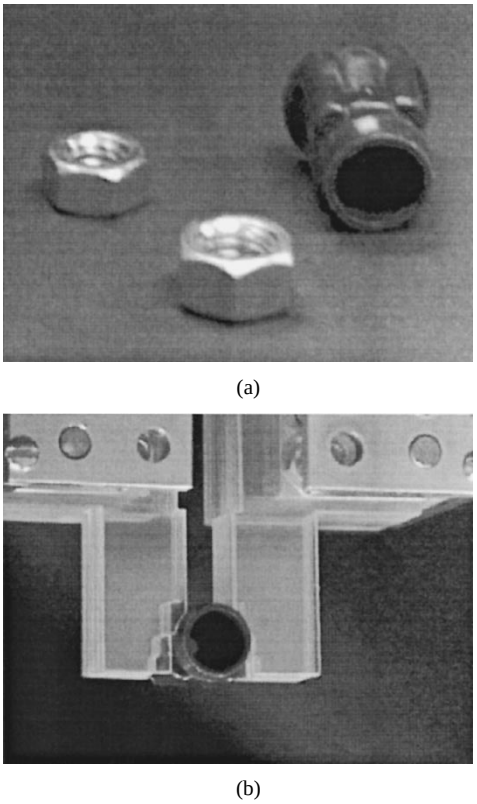

Fig. 2. Experimental multifunction finger design. (a) Workpieces used in evaluation. (b) Fabricated fingers grasping a plastic socket.

We note that our technique for automated gripper design and fabrication is also applicable to automated design and fabrication of fixtures. If a gripper actuator is mounted to ground rather than to a robot's wrist, then it can perform the function of a custom, actuated fixture.

In the above CAD/CAM automated process for gripper design and fabrication, we achieve a compromise between the restrictions of a simple parallel-jaw gripper and the expense and complexity of a more general, anthropomorphic gripper.

The gripper fingers we propose to fabricate are, in fact, special-purpose designs to a high degree; they are computed to handle a specific, small number of part shapes. Agility is achieved by computing and fabricating the fingertip shapes

automatically. This process offers the potential for high performance with low cost and quick response.

C. Agile Special-Purpose Tooling

While robots are capable of a wide range of tasks, some manufacturing operations are better performed by special-purpose equipment. For example, we have employed pneumatic presses to engage injection-molded parts in tight-fitting assemblies. The forces required for this operation exceed the capacity of our lightweight assembly robots. While the press operation could be performed by a stronger robot, substituting a larger robot would be more expensive, slower, and would consume more floorspace. Pneumatic presses for this operation are relatively inexpensive, compact and effective, making it seemingly irresistible to use them. However, the introduction of such special-purpose equipment clashes with the philosophy of agility. Such equipment would be dedicated to a single production operation, and it would not be reprogrammable for reuse in subsequent tasks.

Our approach to reconciling this conflict, introduced in [3], is to modularize and "encapsulate" the use of special-purpose equipment. This concept is borrowed from software engineering philosophy, as discussed further in Section III. By analogy, we emulate the "plug-and-play" paradigm of the computer industry to incorporate alternative special-purpose hardware into an otherwise generic system. A conventional example of this approach is the use of robot wrist quick connectors, which define a standard physical engagement interface, including power and signal interfacing, between custom grippers and a generic wrist plate.

For special-purpose equipment, we employ this philosophy by defining modular work tables. Work tables install in our workcell via a specified mechanical, signal, and power interface. The footprint of a work table is standardized, including fasteners and bolt pattern. Power (pneumatic and electric) and signal interfacing is defined in terms of standardized connector plugs. As a result, work tables can be swapped rapidly with no new machining, wiring, or plumbing. Upon installation, overhead cameras recognize calibration features on a work table, compute the precise installed position and orientation of the work table, and adjust robotic pick-and-place coordinates automatically to accommodate the work table's actual position.

Within the confines of our work-table interface requirements, a designer is otherwise unconstrained in the use of special-purpose equipment, and the resulting design is guaranteed to be compatible with the remainder of the workcell. By making the interface constraints clear, the designer can focus on the specialty equipment without having to simultaneously consider the myriad implications of interacting with the rest of a complex system.

D. Agile Material Handling

The conflict between generality and specificity occurs in material handling as well. The most general material transport system is an autonomous mobile robot (or, somewhat more restrictive, an autonomous guided vehicle). At the other extreme is an indexer with workstations at fixed points along the line. We considered the former approach to be too expensive, complex, and error-prone for our application domain (light mechanical assembly). The latter approach is relatively well developed, robust, and inexpensive, but it should be modified for greater agility.

In conventional conveyor systems, a belt moves constantly along a track and pallets are halted at workstations under computer control via pneumatic stops. (While a pallet is halted, the conveyor belt continues to move, resulting in frictional slip between the pallet and belt.) Sensors at each workstation detect the presence of a pallet (and, optionally, its identity). Interfacing between automation equipment and the material handling system typically consists of a simple binary handshake, including notification (by the conveyor) that a pallet is present and an eventual response (from the equipment at the workstation) that the pallet may be released. Also conventionally, coordination of sensors, actuators, and I/O between the conveyor and workstation equipment is performed in a low-level (e.g., ladder-logic) program on a programmable logic controller (PLC).

Such a conveyance approach is appropriate and successful in high-volume automation systems. For agility, several variations are desirable. First, agile systems should use conveyors that are constructed from modular components. An example of this is the Bosch modular flexible material handling system, which we have employed in our workcell design[1] [3]. Use of modular components permits extensibility in response to changing requirements while minimizing rework on the remainder of the system.

As described in [3], we advocate the use of modular "spurs" as additions to a flexible conveyor system. This addition relieves problems of blocking material flow at workstations and occluding significant reachable workspace of a manipulator by the conveyor. Rather than halt pallets at workstations along the track, we utilize lift/transfer modules to shunt pallets to workstations off the main track. Operations can then be performed by a robot with better utilization of workspace. When the work station operations are completed, the pallet is transferred back onto the main conveyor.

Fig. 3 shows a workstation within our cell exploiting use of a spur. The figure illustrates how the workspace surrounding the pallet on the spur is fully utilized (by work tables and flexible parts feeders), yet additional pallets may continue to pass this station while robotic operations are in progress.

E. Agile Sensing

The conflict between the desire for generality and the need for economy and robustness occurs in the area of sensing as well. To achieve robust, dependable operation, an automated manufacturing system typically requires a large number of sensors for automated error detection and correction. Common errors include attempted assembly of flawed parts, dropped parts, jammed parts, misfed parts, and accidental acquisition of nested parts. In each case, there are a variety of simple and effective sensors that can detect the identified type of error. Options include: optical thru-beam or optical reflection sensors, eddy-current sensors for conductive parts, magnetic sensors for ferrous

[1]Bosch Automation Technology, Racine, WA. [Online] Available: http://www.boschautomation.com

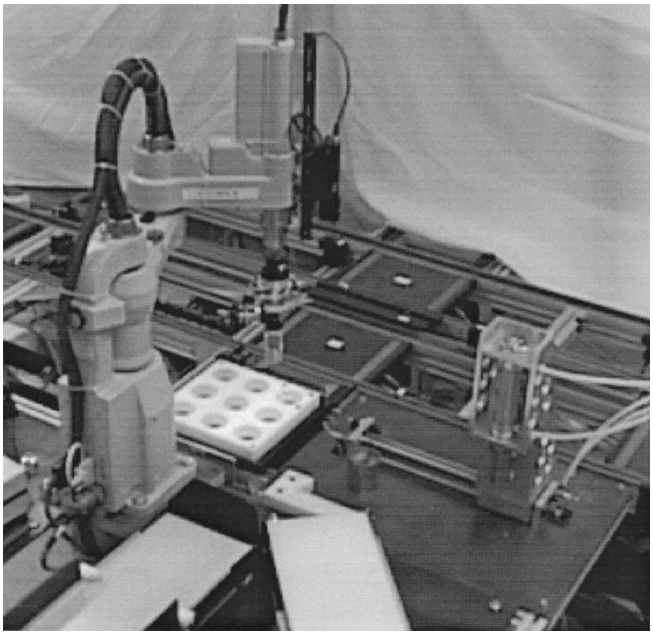

Fig. 3. Assembly station layout.

parts, contact sensors (e.g., microswitches), ultrasonic sensors, capacitive sensors, and others. For a specific error condition with specific parts, it is tempting to select a sensor that is relatively inexpensive and robust for detecting that error. However, such specific choices would not be adapted easily to new applications. An example of a general sensor for detecting part-handling errors is 3-D machine vision. However, 3-D vision is expensive, difficult to program, and error-prone.

Our approach to resolving the conflicting objectives for agile sensing is to use a simplified version of machine vision for a low-cost, relatively simple, general-purpose sensor. In this approach, we utilize a CCD image array and perform image subtraction with respect to a snapshot of a representative successful situation. This approach has been tested using conventional CCD cameras and frame grabbers. In continuing work, the camera sensor is being reduced to a low-cost system with local image subtraction. The resulting output is simply a binary good/bad signal based on the difference image. The otherwise relatively complex and expensive components in a machine vision system can be reduced to a compact sensor unit with a cost rivaling industrial special-purpose sensors. Programming the agile sensor consists of taking a snapshot of a successful state, taking snapshots of example error conditions, and adjusting a quality-of-fit parameter on the difference images consistent with flagging all of the errors. Note that the binary output of this sensor merely indicates success or failure. While it is tempting to analyze the images further to deduce what type of error has occurred, this would be inconsistent with agility. The expert vision programming time required to accomplish more sophisticated image processing would conflict with our goal of quick and easy reapplication of the agile system to new tasks. Restricting image processing to the operation of image subtraction (on region(s) of interest) makes the agile sensor more general than conventional dedicated sensors, yet less general than generic machine vision. We anticipate the use of constellations of inexpensive, vision-based agile sensors, enabling detection of a myriad of errors and permitting reuse of these sensors with relatively low effort and expertise.

III. Software Design Techniques Enabling Rapid Reconfiguration

In agile manufacturing applications, system reconfigurability, rather than speed of operation, is the metric of greatest interest to the system designer. The software architecture used to control an agile manufacturing system is what enables rapid reconfiguration. In part, reconfigurability can be achieved by simply following the practices of object-oriented design and programming, especially encapsulating design details within objects with well-defined interfaces (see, e.g., [41] and [42]). To enhance reconfigurability, it is necessary to go further by identifying the essential components of an agile manufacturing system and their interrelationships; that is, it is necessary to identify the essential *design patterns* [43], [44] of an agile manufacturing system. These components and relationships should be specified in such a way that the constraints on how components are actually implemented are minimized. In this way, an architecture is obtained that can be adapted to different applications and environments and that can be implemented by a variety of lower-level elements. It is the responsibility of the software architect to anticipate the range of requirements that might arise within a family of agile manufacturing applications and to devise an architecture that is general enough to accommodate them.

We note that, in this work, planning for purposes of reconfiguration is performed offline with human intervention. A natural extension is to incorporate increasingly sophisticated levels of self adaptation into the agile system. At this point, however, we have focused on the lower level of how to decouple hardware changes from software changes. This is accomplished by designing the control software with the presumption of change in the associated hardware, and to constrain the hardware changes to obey prescribed interface specifications. Both hardware changeovers and high-level replanning are performed offline with human intervention, but these tasks are made easier by the modular design of both the hardware and software.

Fig. 4 summarizes the architecture that has been adopted to control our workcell in terms of a simplified version of our class structure in Rumbaugh *et al.*'s object modeling technique (OMT) notation [47].

OMT is one of several standard graphical notations used to represent object-oriented software contructs and their interrelationships. Each class is represented by a small rectangle containing the class name, with various arrows indicating how classes are related to each other. The direction of the arrow is significant. The class doing the pointing has the relationship specified by the arrow type with the class being pointed to. For ease of discussion, we will call the pointing class 'A,' and the class being pointed to 'B.' The presence of a large dot at the tip of the arrowhead signifies that A may have the specified type of relationship with multiple instances of B. A dashed arrow indicates that A creates an instance of B. Per Fig. 4, the WorkcellManager creates one instance of the Transporter, and

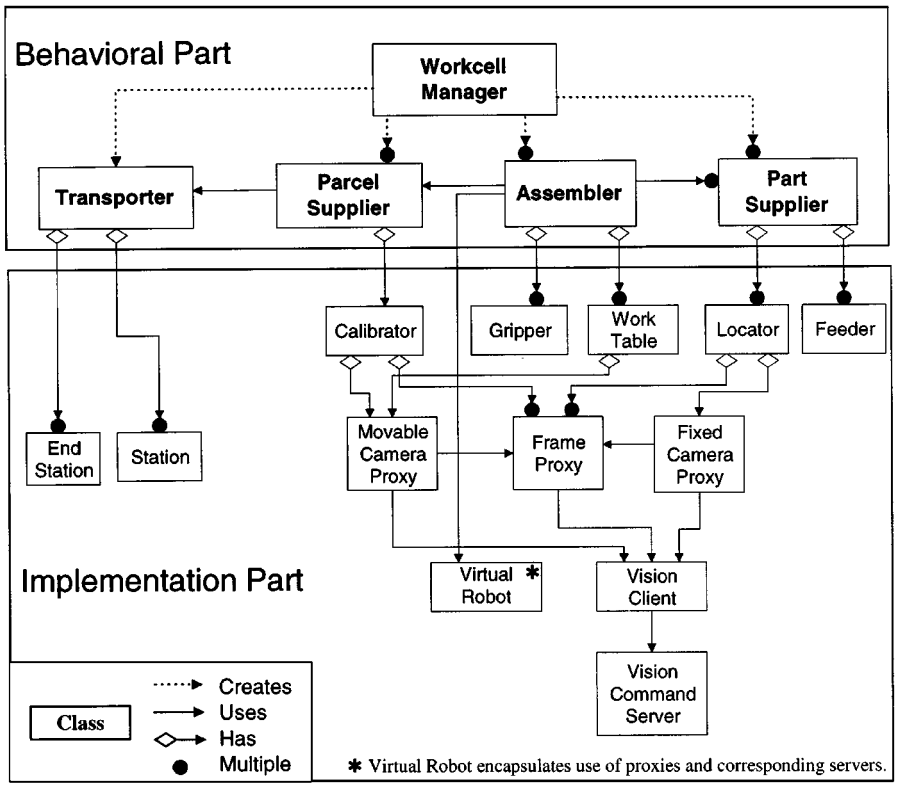

Fig. 4. System class diagram in OMT notation.

one or more instances of ParcelSuppliers, PartSuppliers, and Assemblers.

There is a significant difference between the *uses* and *has* relationships shown in Fig. 4. When A *uses* B, it is using an independently created object. When A *has* a B, it uses an object that it created for itself and that is an essential part of its definition and implementation—B is a part of A. A ParcelSupplier, for example, *uses* a Transporter to move pallets around the workcell, but it *has* its own Calibrator to assist in registering pallets when they arrive.

Fig. 4 shows the separation of software components into two categories, implementation components and behavioral components, which we describe in the following two sections.

A. Low-Level Hardware Control Objects

At the lowest levels, one has the greatest opportunity for encapsulating details that would otherwise inhibit code reuse. In some cases, taking advantage of modular hardware design is enough to ensure that higher-level software constructs can use various hardware components without requiring modification. In others, the implementation of agents which run as concurrent threads within a real-time operating system (VxWorks™ in our implementation) provide a buffer between those lowest-level objects that interact directly with the hardware and the rest of the system. In either case, the development of a sufficiently functional, yet generic, interface is the real challenge. If the interfaces of either the objects or the mediating agents are composed correctly, then the design of higher-level software can safely restrict attention to *what* tasks need to be performed, rather than *how*.

The implementation of the software controlling gripper activation is an excellent example of how modular hardware design can be used to facilitate the development of generic control software. Physical grippers are connected to a robot's flange using a generic connector with a fixed number of pneumatic channels that control the gripper's behavior. Manipulating digital signals trigger solenoids, which supply or deny air to the pneumatic channels. The software used to control gripper operation is written generically, depending on a specified set of constant values to perform the correct action. In our implementation, a physical gripper usually has several software representations, one for each tool presented during the assembly process. While this may seem repetitive, each tool presentation requires a different translation to ensure that the robot is aware of the respective gripper tip location relative to the end of its flange. Switching grippers is easy as far as the software is concerned; it switches several times during the assembly process while the same physical gripper is attached to the robot. At initialization time, the Assembler will create all of the grippers it needs, initializing a generic gripper object with the appropriate signal values as they are read from a file, loaded from a central repository, or gathered from a database.

Encapsulation of detail for robot motion commands is also necessary. To accomplish this, we have defined a robot motion server to abstract the services performed by robots. Ideally, one would program robots in terms of the desired behavior, e.g., that a part at a specified position and orientation should be relocated to another specified position and orientation. At this level of abstraction, details of a robot's programming language, operating system, and indeed its kinematics should be irrelevant to its clients.

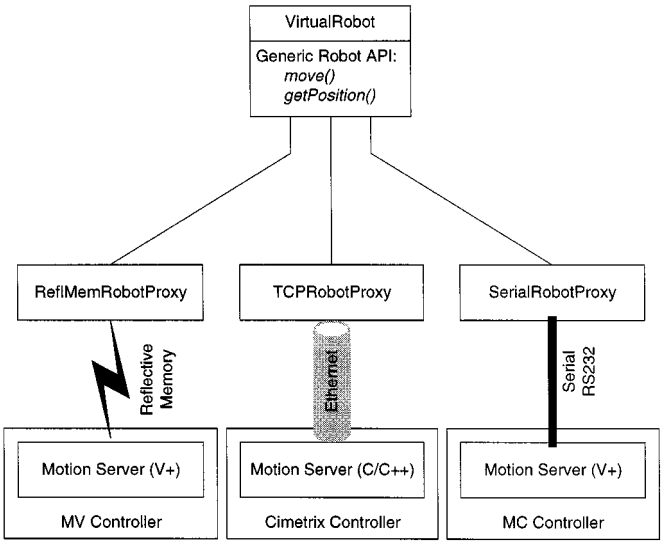

Fig. 5. Proxy abstraction of a generic (virtual) robot.

Preceding efforts at unifying robot control include the robot independent programming language (RIPL) from Sandia Laboratories [41] and the open architecture control specification from Cimetrix [45]. In our experimental system, we utilize three different robot controllers: an AdeptOne MC controller, a dual-arm Adept 550 MV controller, and a Cimetrix controller retrofit to an AdeptOne arm. We utilize a reduced command set—essentially a subset of the Cimetrix OAC specification—to command all robots with a common language, as per the philosophy of RIPL. Our reduced command set consists exclusively of task-space commands. By avoiding use of joint-space commands, the motion server can be independent of what type of robot is being used, and in this way the motion server hides information regarding implementation details, including robot kinematics.

Implementation of our motion server requires hardware-specific interfacing for each robot controller type, as well as installation-specific details of robot and workspace calibration. Our approach, illustrated in Fig. 5, is to write a simple program in the native language of each robot controller that communicates with higher levels via a defined physical and syntactical protocol. It receives motion requests from a higher level, invokes the specified (task-space) motion on the control platform, and reports status back to the higher level. For the AdeptOne/MC system, our local server program is written in V+ and communications is via a serial port. For the dual-arm Adept MV controller, two V+ server tasks control the two Adept 550 arms, and communications with higher levels is via a reflective memory network card installed on the MV's VME bus. The Cimetrix server program is written in "C," and it communicates with higher levels via Ethernet, TCP/IP, and Unix sockets. These details are hidden from higher levels, which interact with robot objects providing a generic interface, and by robot "proxies" [44], which encapsulate the details of communicating with a remote robot.

With our motion server approach, nonessential differences between robots are abstracted away. The only relevant behavior from the viewpoint of the clients is that the requested motions of parts take place. The fact that the requested service is performed with a particular robot is irrelevant to the client. We note that the effectiveness of this approach is coincident with the objectives and barriers of offline robot programming. Specifically, robot calibration (including gripper and tool calibration) is a critical requirement for success of programming in task-space coordinates. Since our system depends heavily on the use of vision-based manipulation, we have already accepted the burden of robot calibration. Having performed such calibration, we reap additional benefits in terms of practicality of task-space motion server capability as well as offline programming capability.

Our second major low-level object is the vision server. In our machine-vision software design, it is necessary to anticipate change (e.g., adding new image-processing hardware), to hide implementation details (e.g., vendor-specific languages and hardware details), and to enable resource-sharing while minimizing the corresponding complications imposed on the application programmer.

Our vision server addresses these needs using a client-server architecture to implement abstractions of machine-vision functionality. It can process images from multiple camera inputs, it can perform generic image-processing operations (e.g., threshold, segment, label, centroid computation, etc.), it can utilize available hardware resources (ranging from host-processor computations to image-processor-specific hardware acceleration capabilities), it can serve clients from multiple platforms (tested to date with clients on Unix, Linux, LynxOS, and VxWorks with Intel, Motorola, and SunMicrosystems processors), and it can service asynchronous requests from multiple clients.

From the viewpoint of the client, only the behavioral aspects of the vision server are important, independent of how they are implemented in hardware. In the abstract, a machine vision system contains three things: sensors (or cameras), frames (or images), and operations on frames. Cameras are used to acquire images into frames. These frames then understand how to perform operations on their images to extract information. By designing programs using these operations, developers can build useful machine vision tasks without being concerned about how the operations map onto a particular vendor's hardware. By interposing an idealized virtual machine between the implementation of the abstract-functionality model and the actual vision hardware, the task of porting the abstract model to a new hardware platform is radically simplified. Since such a change would be restricted to local modules, no client objects would require reprogramming, and the applications programmer would not need to learn changing vendor-specific hardware or software details. For further details on the vision-server design, see [11] and [46].

A third major component of our software architecture is an intermediate-level agent, the *transporter*. The transporter, which is responsible for control of the Bosch conveyor system,[2] is indicative of the agent design challenges. It is designed as a server, in stark contrast to the conventional approach of using a PLC with a dedicated ladder-logic program to control a material-handling system. The transporter accepts requests to move pallets of partial or completed assemblies between workstations. From the viewpoint of its clients, it is irrelevant how the transporter satisfies requests; it is only necessary to know which services

[2]Bosch Automation Technology, Racine, WA. [Online] Available: http://www.boschautomation.com

are available, how to request a service, and how to interpret the status information associated with a service that has been performed (e.g., the identity and coordinates of a pallet that has been delivered). The clients do not need to be aware of any of the details of the wiring, I/O address assignments, nor timing of potentially hundreds of sensor and actuator signals. Additionally, while the conveyor is a shared resource, details of how this resource responds to multiple, asynchronous service requests is hidden from the clients, thus decoupling the design constraints considered at the higher levels. Although our conveyor consists of a single conveyor loop, this intermediate-level agent could encapsulate significantly more complexity, including responsibility for any necessary parcel routing or storage between sources and destinations, as well as encapsulation of more sophisticated physical means of transport, such as AGV's. Such extensions would not affect the higher-level software.

Finally, our *part-supplier* is also an intermediate-level agent, conceptually similar to the transporter. A part-supplier interacts with two types of low-level objects: feeders, which command the servo controllers of the flexible-parts-feeder conveyors, and locators, which in turn encapsulate interactions with the machine vision system. From the viewpoint of clients, only the behavioral aspects of a part-supplier are important: what parts it is capable of supplying, how to request a service, and how to interpret the result (position and orientation of the requested part).

B. Higher-Level Control Objects

Having defined the low-level control objects, objects at the higher levels can interact with those at lower levels through interfaces describing their abstract behaviors, without regard to implementation details. Our higher-level classes interact as clients of the intermediate- and lower-level agents: Transporter, PartSupplier, MotionServer, and VisionServer.

The WorkcellManager is at the top of the control hierarchy, but its role is fairly limited. It is a supervisory agent that creates all of the Assemblers, Suppliers, and Transporters needed by the workcell, and allocates resources to them. The WorkcellManager starts and shuts down workcell operation, communicates with the operator, and orchestrates activities that require cooperation between agents in exceptional circumstances, such as error recovery.

If a registry, text file, database, or other resource is used to configure workcell components, the WorkcellManager would be responsible for performing the necessary steps to retrieve the required information from that resource, even if that means the instantiation and manipulation of another object. For the initialization of our system, the contents of a file containing configuration data in textual form is read and placed in a hierarchical registry that is queried when objects are instantiated.

The assembly of a product involves a particular sequence of steps. Some specialized equipment, such as robot grippers and modular work tables, is typically needed to execute this sequence. Some specialized software components, such as device drivers and part recognition strategies, are also needed. Since the assembly process is one of the most variable aspects of an agile manufacturing system, it is essential to encapsulate it. Therefore, the main control agent in our software control architecture is the Assembler.

An assembler object requests parts from a part supplier, requests and yields pallets from/to a parcel supplier (which uses the "transporter"), requests motions of one or more robot proxies, and (optionally) requests specialized services from work tables.

An assembler does not need to interact directly with the vision server; at its level of abstraction, the use of vision is an implementation means for obtaining coordinates of interest. A part supplier utilizes machine vision, but it informs its assembler client of a part's location/orientation coordinates. These coordinates are, in turn, specified by the assembler in its request to a robot proxy for motion service. Similarly, the parcel supplier invokes use of machine vision to obtain accurate coordinates of a delivered pallet, and it informs a client assembler of the presence and coordinates of the requested parcel. Thus, the assembler has no direct need for vision services.

An intermediate class is defined to assist in interactions between the part supplier and the vision server. The vision server accepts low-level image-processing commands. This level of interaction is necessary for programming recognition of specific parts, as the sequence and parameters of low-level image-processing operations must be identified for each new part. However, such programming detail is too implementation-specific relative to the level of abstraction of the suppliers. We thus introduce the "locator" class, which includes objects that are constructed to recognize specific parts. The part supplier can request the services of locator objects to obtain coordinates of named parts. This organization encapsulates the programming of part-specific image-processing routines within objects that insulate such details from affecting the rest of the control software.

The interactions described here among assemblers, transporters, and suppliers constitute our *assembler-transporter-supplier* design pattern, which we postulate is inherent in agile manufacturing applications in the context of light mechanical assembly.

C. Implementation Issues

Our current object-oriented control software is comprised of roughly 30 000 lines of C++ source code defining about 90 classes. It executes under the real-time operating system (RTOS) VxWorks™. The active agents of the system, the assemblers, suppliers, the transporter, and the vision and motion servers, operate concurrently as separate tasks. This concurrency is invisible to the clients (C++ does not directly support concurrency, but intertask communications can be implemented with details hidden within C++ class interface modules). Clients need not make RTOS system calls to create, schedule, synchronize, or communicate between tasks; this is done by the class implementations. Presently, 20 tasks run concurrently in round-robin priority with 20-ms time slices, which is well within our timing requirements.

We have experimented with rapid response to change, both in terms of introducing new assembly tasks on new products and in terms of adding hardware to the system. Our class definitions succeeded in promoting reusability of code. In our experience, we have observed better than 90% code reuse in programming new assembly applications. Further, we can add control

of an extra robotic workstation by merely composing the corresponding workstation objects and editing details within the transporter object to accommodate the new spur. (If the new robot uses a new controller type, then it will also need a motion server task composed in its native language, and a corresponding robot proxy must be built.) Such additions require no revisions to our overall software architecture, and none of the remaining functioning control code would require changes.

IV. Conclusions and Future Work

In this overview of our agile manufacturing research, we have offered some simple lessons for the design of agile manufacturing systems.

Our hardware recommendations are as follows:

- use vision-based flexible part feeders;
- use rapid, automated CAD/CAM fabrication of custom grippers and fixtures;
- use encapsulation of special-purpose equipment on work tables with standard interfaces;
- use lift/transfer units and spurs for interfacing workstations to conveyor systems;
- use low-cost vision sensors to emulate simple, dedicated sensors.

Our software recommendations are as follows:

- use object-oriented programming;
- run concurrent tasks within a real-time operating system;
- use encapsulation or information hiding (particularly with respect to low-level implementation details);
- make a clean separation between behavior and implementation;
- utilize client/server interactions for generic vision, motion, transportation, and part-feeding services;
- restrict attention to an appropriate target range of flexibility (e.g., light mechanical assembly);
- identify recurring design patterns applicable to the chosen context (e.g., the assembler-transporter-supplier design pattern).

Detailed results of our work can be found in [1]–[14] and [34]. In future work, we recognize the need for extending investigations in the following areas:

- flexible parts feeding—faster and cheaper designs, and extensions to handle rolling parts;
- automated gripper design—extensions to optimization with respect to sliding contacts guiding parts into precise alignment;
- work-table development—interface extensions for automatic plug-and-play mechanical, electronic, and software configuration;
- agile sensors—reduction to practice of low-cost, specialized vision sensors;
- vision programming—software tools for reducing the required technical expertise for programming new vision applications;
- error detection and recovery—identification of appropriate modularity, encapsulation, and design patterns for automated error detection and correction;
- assembly server—extensions of motion server to respond to sensed forces and moments of interaction to guide parts mating.

Although considerable work is called for in continuing to uncover principles, lessons, and techniques for agile manufacturing, results to date are already useful for immediate application to design of practical agile manufacturing systems.

Acknowledgment

The authors would like to thank the following current or former faculty and students who contributed substantially to the agile manufacturing project: L. Sterling, S. Kim, S. Ameduri, R. Bachman, M. Birch, E. Blanchard, C. Chamis, S. Chhatpar, L. Huang, J.-Y. Jo, Y. Karagoz, S. Ramasubramanian, D. Sargent, T. Wei, H. Zhang, W. Zhang, and G. Zhao.

References

[1] R. D. Quinn *et al.*, "Design of an agile manufacturing workcell for light mechanical applications," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, 1996, pp. 858–863.
[2] ——, "Design of an agile manufacturing workcell for light mechanical applications," in *Video Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, 1996.
[3] ——, "An agile manufacturing workcell design," *IIE Trans.*, vol. 29, pp. 901–909, 1997.
[4] ——, "Advances in agile manufacturing," in *Video Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.
[5] F. L. Merat *et al.*, "Advances in agile manufacturing," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997, pp. 1216–1222.
[6] G. C. Causey *et al.*, "Design of a flexible parts feeding system," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997, pp. 1235–1240.
[7] Y. Kim *et al.*, "A flexible software architecture for agile manufacturing," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997, pp. 3043–3047.
[8] J. Y. Jo *et al.*, "Virtual testing of agile manufacturing software using 3D graphical simulation," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997, pp. 1223–1228.
[9] V. B. Velasco, Jr. *et al.*, "Computer-assisted gripper and fixture customization via rapid prototyping," in *Proc. Int. Conf. on Robotics and Manufacturing*, 1997, pp. 115–120.
[10] ——, "Computer-assisted gripper and fixture customization using rapid prototyping technology," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Leuven, Belgium, 1998, pp. 3658–3664.
[11] N. A. Barendt *et al.*, "A distributed, object-oriented architecture for platform-independent machine vision," in *Proc. Int. Conf. on Robotics and Manufacturing*, 1998, pp. 50–55.
[12] W. S. Newman *et al.*, "Technologies for robust agile manufacturing," in *Proc. Int. Conf. on Robotics and Manufacturing*, 1998, pp. 56–61.
[13] G. C. Causey and R. D. Quinn, "CWRU flexible parts feeding system," in *Video Proc. IEEE Int. Conf. on Robotics and Automation*, Leuven, Belgium, 1998.
[14] W. S. Newman *et al.*, "Technologies for robust agile manufacturing," in *Video Proc. IEEE Int. Conf. on Robotics and Automation*, Leuven, Belgium, 1998.
[15] B. Horn, *Robot Vision*. New York: McGraw-Hill, 1986, ch. 18.
[16] R. B. Kelly *et al.*, "A robot system which acquires cylindrical workpieces from bins," *IEEE Trans. Syst., Man, Cybern.*, vol. 12, no. 2, pp. 204–213, Feb. 1982.
[17] W. Schmidt, "Multiple part orientation using a CVIM system," presented at the RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop, 1995.
[18] C. E. Bailey, "Strategies for flexible parts feeding," presented at the RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop, 1995.
[19] *MRW Robo-Pot*, Comtech, Farmington Hills, MI.
[20] W. F. Davis, "Centrifugal feeders the flexible approach," presented at the 1994 RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop, 1994.

[21] T. Nonaka and K. Otsuka, "Photoelectric Control System for Parts Orientation," U.S. Patent 4 333 558, June 8, 1982.
[22] J. L. Goodrich and W. L. Devlin, "Programmable Parts Feeder," U.S. Patent 4 608 646, Aug. 26, 1986.
[23] P. E. Kane, "Part Positioning Apparatus and Method," U.S. Patent 4 712 974, Dec. 15, 1987.
[24] A. L. Dean *et al.*, "Programmable Parts Feeder," U.S. Patent 4 619 356, Oct. 28, 1986.
[25] *Tape and Reel Technology for Automatic Packaging and Feeding of Connectors*, GPAX International, Inc., Columbus, OH.
[26] P. E. Kane, "Technical Presentation," presented at the RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop, 1994.
[27] *Advanced Parts Orientating System, T-APOS*, SonyFA, Rangeburg, NY.
[28] K. Goldberg , M. Mason , and M. Erdman *et al.*, "Generating stochastic plans for a programmable parts feeder," in *Proc. 1991 Int. Conf. on Robotics and Automation*, Sacramento, CA, Apr. 1991.
[29] M. Brokowski *et al.*, "Curved Fences for Part Alignment," in *Proc. 1993 Int. Conf. on Robotics and Automation*, 1993.
[30] K. Yoneda, "Bottle unscrambler," in *Proc. RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop*, Cincinnati, OH, Oct. 1994.
[31] K. Tsuruyama *et al.*, "Apparatus for aligning vessels," U.S. Patent 5 370 216, Dec. 6, 1994.
[32] D. Boehlke, Smart design for flexible feeding machine design, Dec. 1994.
[33] *Adept Robotics, Adept FlexFeeder 250,* , San Jose, CA, Sept. 1995.
[34] G. C. Causey, "Elements of agility in manufacturing," Ph.D. dissertation, Dept. Mech. Aerosp. Eng., Case Western Reserve Univ., Cleveland, OH, Jan. 1999.
[35] M. T. Mason and J. K. Salisbury, *Robot Hands and the Mechanics of Manipulation*. Cambridge, MA: MIT Press, 1985.
[36] J. D. Cawley *et al.*, "Al203 ceramics made by CAM-LEM (computer-aided manufacturing of laminated engineering materials) technology," in *Proc. Solid Freeform Fabrication Symp.*, Austin, TX, Aug. 1995, pp. 9–16.
[37] B. B. Mathewson *et al.*, "Automated fabrication of ceramic components from tape-cast ceramic," in *Proc. Solid Freeform Fabrication Symp.* Austin, TX, Aug. 1995, pp. 253–260.
[38] F. Reuleaux, *The Kinematics of Machinery*. New York: Dover, 1963.
[39] J. C. Trinkle, "A quantitative test for form closure grasps," in *IEEE Int. Conf. on Intelligent Robots and Systems*, July 1992, pp. 1245–1251.
[40] R. G. Brown and R. C. Brost, "A 3-D modular gripper design tool," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1997, pp. 2332–2339.
[41] D. J. Miller and R. C. Lennox, "An object-oriented environment for robot system architectures," *IEEE Contr. Syst.*, vol. 112, pp. 14–23, Feb. 1991.
[42] S. Adiga and P. Cogez, "Toward an object-oriented architecture for CIM systems," in *Object-Oriented Software for Manufacturing Systems*. London, U.K.: Chapman & Hall, 1993.
[43] F. Buschmann and R. Meunier, "Building a software system," *Electron. Design*, vol. 43, p. 132, Feb. 1995.
[44] E. Gamma *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.
[45] R. L. Anderson, "Open architecture controller solution for custom machine systems," *Proc. SPIE*, vol. 2912, pp. 113–127, 1996.
[46] N. A. Barendt, "A distributed, object-oriented software architecture for platform-independent machine vision," M.S. thesis, Dept. Elect. Eng., Case Western Reserve Univ., Cleveland, OH, May 1998.
[47] J. Rumbaugh *et al.*, *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

Wyatt S. Newman (M'87) received the S.B. degree in engineering science from Harvard College, the M.S.E.E. in control theory from Columbia University, and the M.S.M.E. and Ph.D. degrees in fluid and thermal sciences and mechanical engineering, design, and control, respectively, from Massachusetts Institute of Technology.

He is currently a Professor of Electrical Engineering and Computer Science at Case Western Reserve University (CWRU), Cleveland, OH. Prior to joining CWRU, he was a Senior Member of Research Staff at Philips Laboratories, Briarcliff Manor, NY. He has also been a Visiting Research Scientist at: Philips Natuurkundig Laboratorium, Eindhoven, The Netherlands; NASA Lewis Research Center, Cleveland, OH; and Sandia National Laboratories, Albuquerque, NM. His research interests are in mechatronics and robotics.

Andy Podgurski (S'85–M'90) received the Ph.D. degree in computer science from the University of Massachusetts at Amherst in 1989.

In 1989, he joined Case Western Reserve University, Cleveland, OH, where he is currently an Associate Professor in the Department of Electrical Engineering and Computer Science. His research interests include software engineering, software architecture, distributed systems, software testing and analysis, and software for advanced manufacturing systems.

Dr. Podgurski is a member of the ACM.

Roger D. Quinn received the B.S.M.E. and M.S.M.E. degrees from the University of Akron, Akron, OH, in 1980 and 1984, respectively, and the Ph.D. degree in engineering science and mechanics from Virginia Polytechnic Institute and State University, Blacksburg, in 1985.

He is currently a Professor on the faculty of the Mechanical and Aerospace Engineering Department at Case Western Reserve University (CWRU), Cleveland, OH. From 1994 to 1998, he was the PI for the CWRU Agile Manufacturing Project (http://dora.eeap.cwru.edu/agile/). He has also directed the Biorobotics Laboratory (http://biorobots.cwru.edu) at CWRU since its inception in 1990, engaging in design and control of insect-inspired robots. His research is devoted to biorobotics and robotics for manufacturing.

Frank L. Merat (S'73–M'77) received the B.S degree in electrical engineering from Case Institute of Technology, in 1972, and the M.S. and Ph.D. degrees in electrical engineering from Case Western Reserve University (CWRU), Cleveland, OH, in 1975 and 1979, respectively.

He is presently Associate Professor of Electrical Engineering and Computer Science and Interim Department Chair at CWRU. His technical interests include three-dimensional computer vision, wireless communications, digital image processing and industrial inspection, and micro-optomechanical systems.

Dr. Merat is a member of SPIE, ACM, and SME.

Michael S. Branicky (S'87–M'95) received the B.S. and M.S. degrees in electrical engineering and applied physics from Case Western Reserve University (CWRU), Cleveland, OH, in 1987 and 1990, respectively. He received the Sc.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1995.

In 1997, he rejoined CWRU as Nord Assistant Professor of Engineering. He has held research positions at MIT's AI Lab, Wright-Patterson AFB, NASA Ames, Siemens Corporate Research (Munich), and Lund Institute of Technology's Department of Automatic Control. Research interests include hybrid systems, intelligent control, and learning with applications to robotics, flexible manufacturing, and networks.

Nick A. Barendt (S'96–M'97) received the B.S. and M.S. degrees in electrical engineering from Case Western Reserve University, Cleveland, OH, in 1995 and 1998, respectively.

He currently designs software to control automation for the electronics industry for Nordson Corporation, Cleveland, OH. His interests are in the fields of machine vision, robotics, and embedded/real-time systems.

Greg C. Causey received the B.S., M.S., and Ph.D. degrees in mechanical engineering from Case Western Reserve University (CWRU), Cleveland, OH, in 1991, 1993, and 1999, respectively.

His dissertation work dealt with flexible parts feeding, robotic offsets which enable modular workcells, gripper design, and component design for flexible manufacturing systems. He is currently a Postdoctoral Research Associate in the Center for Automation and Intelligent Systems Research at CWRU. His research interests lie in the areas of the design, construction, control, and simulation of flexible parts feeding systems. He is also interested in flexible manufacturing systems.

Erin L. Haaser is working toward the M.S. degree in computer science at Case Western Reserve University (CWRU), Cleveland, OH, focusing on software architecture design in the agile manufacturing domain.

Yoohwan Kim received the BA degree in economics from Seoul National University, Korea, in 1989, and the M.S. degree in computer science from Case Western Reserve University (CWRU), Cleveland, OH, in 1994. He is currently working toward the Ph.D degree in computer science at CWRU.

Since 1997, he has been working as a Member of Technical Staff at Lucent Technologies, Bell Laboratories, Whippany, NJ, where he develops the base-station software for third-generation WCDMA wireless networks. His research interests include software architecture, object-oriented software design, software testing, real-time embedded systems, and telecommunication network software.

Jayendran Swaminathan received the B.E degree in computer science from the University of Pune, India, in 1996. He received the M.S. degreee in computer science from Case Western Reserve University, Cleveland, OH, in 1999.

His Master's thesis involved design and implementation of a software framework to enable and automate robotic peg-in-maze assemblies. He is currently a Member of the Microsoft Windows CE OS team, which focuses on embedded and real-time systems targeting devices and markets such as cellular phones, webtv, and industrial automation. His research interests include object-oriented design, real-time, and embedded systems design.

Virgilio B. Velasco, Jr. received the B.S. degrees in physics and computer engineering from Ateneo de Manila University in the Philippines in 1989 and 1990. He then received the M.S. and Ph.D. degrees in electrical engineering from Case Western Reserve University, Cleveland, OH, in 1993 and 1997.

He currently works for Philips Digital Video Systems, West Valley City, UT. His research interests have included robotic control and object manipulation, as well as automated manufacturing philosophies.