

Design Lessons for Building Agile Manufacturing Systems

Wyatt S. Newman¹, Andy Podgurski¹, Roger D. Quinn², Frank L. Merat¹, Michael S. Branicky¹,

Nick A. Barendt¹, Greg C. Causey², Erin L. Haaser¹,

Yoohwan Kim¹, Jayendran Swaminathan¹, Virgilio B. Velasco¹

Center for Automation and Intelligent Systems Research

Case Western Reserve University

Cleveland, OH 44106

ABSTRACT

This presentation 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.

¹ Electrical Engineering and Computer Science Dept.

² Mechanical and Aerospace Engineering Dept.

1 INTRODUCTION

This paper is based on results obtained and lessons learned over 5 years of recent multi-disciplinary research in agile manufacturing at Case Western Reserve University [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. Agility of this type offers the promise of achieving economical automation of batch production or short life-cycle product manufacturing by reusing generic production equipment across many applications and by reducing the lead time and implementation cost for each new application.

The CWRU experimental workcell is shown in Figure 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 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 useful implications.

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.

2 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 tension. 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.

2.1 AGILE PARTS FEEDING

Parts feeding in industrial automation is typically performed with vibratory bowl feeders. A bowl feeder takes randomly oriented parts in bulk and presents those parts one at a time at a fixed position and orientation. This creation of order establishes boundary conditions that greatly simplify the remainder of the workcell design. Unfortunately, bowl feeders do not satisfy the requirements for agile manufacturing. These components 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 use of bowl feeders does not support rapid response. 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 3-D sensing, pattern recognition in cluttered scenes with overlapping parts, 6-dof manipulation, and sophisticated approach and grasp planning. Such complexity is required to recognize and manipulate randomly-oriented parts in a bin. 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. Our flexible parts feeder is illustrated schematically in Figure 2, and our implementation is shown in Figure 3. The system consists of three conveyors that work together to present parts to a robot. The first conveyor, under servo control, is mounted at an inclined angle and is used to lift parts from a bulk hopper. Parts slide down a ramp at the end of the inclined conveyor and onto the horizontal conveyor. The horizontal conveyor terminates within the work envelope of a robot, where an underlit window provides a clean silhouette of the part for location and classification by a vision system. Based upon information from the vision system, a robot is used to acquire the parts for assembly. Parts that are overlapping or are not in useful orientations for assembly are dropped from the end of the horizontal conveyor onto a return conveyor. The return conveyor transports the parts back to the bulk hopper for re-feeding. As an example, the picture on the left in Figure 3 shows LEGO blocks drawn from the hopper and ascending the inclined conveyor. The return conveyor to the right is shown returning parts to the bulk hopper. The picture on the right in Figure 3 shows sockets spilling from the inclined conveyor onto the horizontal conveyor. Parts near the top of this figure are within the underlit vision window of the horizontal conveyor; parts within this region would be identified and located by the vision system then grasped by the robot.

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/minute (depending on part size

and geometry). We have validated the system in feeding trials of 150,000 parts, including a 60-hour unattended continuous run, demonstrating high reliability. (Further detail on our flexible feeder design and performance is given in [6, 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. Unlike the bowl-feeder approach, it is not required that each part be presented in a fixed position and orientation. Also, the flexible feeder does not attempt to handle all parts, as opposed to the general bin-picking problem. Instead, parts which are difficult to recognize (e.g., due to overlapping) or are difficult to grasp (e.g., due to closely surrounding parts) are ignored, returned to the hopper, and eventually re-presented with another random opportunity to achieve a more favorable (recognizable and graspable) state. Based on our observations of reprogrammability, flexibility, reliability, and throughput, we conclude that this approach to parts feeding is appropriate for agile manufacturing.

2.2 AGILE GRIPPERS AND FIXTURES

Anotegrams of the (e)2, siob4(et)w(h)-2(vedfeedi)c2(at)4(eso(cl)4(h)-tal)4(itione and(general)4so(cl)4(h)-tali in agile manufac
ur4ffieste.T(thdem)13(h)-s4(t)4(oram)13(m)1[(o(prAnot)og(ur4)hopp1(

designed fingers, where the finger shapes are designed and fabricated automatically via CAD/CAM. Our approach, which is detailed in [9, 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 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 envelope the desired part, as illustrated in Figure 4.

Figure 4 shows the result of the virtual EDM operation, using the part geometry as a virtual tool. 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 set-up). 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, 37].

We have shown that it is possible to define and fabricate multi-function 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 multi-function 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

Figure 5. Three parts (2 sizes of hex nuts and a plastic socket), shown in the left figure, are all graspable in frictional form closure by the example computed and fabricated gripper fingers shown in the right figure. Experiments showed that this gripper manipulated these three parts with greater precision than simple parallel-jaw fingers [10].

In on-going research, gripper-design enhancements are being incorporated within our automated design and fabrication process. Our default finger block shapes will include sculpting of the non-grasping (external) surfaces to minimize the “footprint” of the gripper, including tapering the fingers to help singulate nearly-touching parts. Additionally, we are working on sculpting the contacting (internal) surfaces to promote sliding contact between a part and the gripper fingers in the act of gripper closure to guide the grasped part into precise location.

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.

2.3 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 light-weight 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 is to “encapsulate” the use of special-purpose equipment. This concept is borrowed from software engineering philosophy, as discussed further in section 3. A common example of encapsulation is the plug-and-play concept for personal computers. In the plug-and-play philosophy, alternative special-purpose hardware can be installed into an otherwise generic computer platform, and the computer recognizes the new hardware, associates it with device-driver software, and acquires new functionality quickly and easily. Similarly, the robot wrist quick connectors mentioned in section 2.2 define a standard physical engagement interface, which may also include 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, as illustrated in Figure 6. The top figure shows an installation of specialty equipment, the middle view shows the same site with the previous work table removed, and the lower figure shows installation of a different work table. 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 worktable, compute the precise installed position and orientation of the work table, and adjust robotic pick-and-place coordinates automatically to accommodate the worktable’s actual position.

To be agile, it is not only necessary that work tables can be exchanged rapidly, it is also necessary that any new specialty equipment can be introduced rapidly to adapt the system to new requirements. Our interface specification for worktables helps address this need. The designer is free to utilize any specialty equipment desired, but design is constrained to satisfy the mechanical baseplate and envelope constraints. Power and signal I/O must also conform to the defined interface specifications via standardized worktable plugs. Within the confines of this

“encapsulation,” the designer is otherwise unconstrained, 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. Through encapsulation applied to work tables, we resolve the tradeoff between the appeal of specificity and the desire for generality in the use of special-purpose equipment.

2.4 AGILE MATERIAL HANDLING

The tension 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 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. Coordination of sensors, actuators and I/O between the conveyor and workstation equipment is typically performed in a low-level (e.g., ladder-logic) program on a Programmable Logic Controller (PLC).

The conveyor approach is appropriate and successful in high-volume automationobust and in1 Tci~4(a)3(e wor rThe Cona lr)0(

As noted, the standard conveyor methodology involves halting pallets at stations along the path of the belt. With this approach, a pallet halted at a workstation blocks the transfer of all pallets upstream of that workstation. In addition, if a robot is to work on a pallet halted on the conveyor, then a significant fraction of the robot's workspace overlaps the conveyor track upstream and downstream of the halted pallet. Since the conveyor track must be kept open for pallet transit, the overlapping regions are unusable for work tables or parts feeders, and thus a significant fraction of the robot's reachable workspace is unusable. Since the fastest and least expensive robots have small workspaces, it is important to utilize the robot's workspace as fully as possible, and this is in conflict with workstations located on the conveyor.

Our solution to the above problems (blocking material flow and occluding useful workspace) is to introduce "spurs" in the material handling system. 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. Figure 7 illustrates the spur concept, as analyzed in the design phase under graphical simulation, as well as the implementation of the conveyor spur, surrounded by work tables and parts feeders. The figures illustrate how the workspace surrounding the pallet on the spur is fully utilized, and additional pallets may continue to pass this station while robotic operations are in progress.

With the use of modular conveyor components and the introduction of lift/transfer units and workstations located on spurs, simple conveyor systems can be sufficiently flexible to support the requirements of agile manufacturing. The design approach to the control software for such conveyors is equally important, but this topic is deferred until section 3.

2.5 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, mis-fed 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 parts, contact sensors (e.g., micro-switches), 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, three-dimensional 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 re-application 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 re-use of these sensors with relatively low effort and expertise.

3 SOFTWARE DESIGN TECHNIQUES ENABLING RAPID RECONFIGURATION:

An appropriate software design approach for agile manufacturing also requires resolution of the conflicting appeals and restrictions of generality vs. specialization. While the same conceptual issues must be addressed, the options and compromises are not as intuitive to visualize as in the case of hardware. Humans can naturally interpret robots in the context of arms, grippers in the context of hands, sensors in the context of human vision, and material transport in the context of locomotion. Technological analogies are common and familiar. For software design, the human analogy is the brain—but this analogy does not help the designer in terms of emulating a known solution nor evaluating design approaches and choosing an appropriate level of generality. Instead, we must rely on software engineering design principles to guide our choices. Such principles are motivated by objectives consistent with agile manufacturing: ease of extensibility, maintainability, and code re-use for new applications. The current software engineering perspective is also comfortably in a compromise position between the special-purpose approach of dedicated, hard-coded solutions and general artificial-intelligence techniques such as neural networks, evolutionary programming or attempts at constructing a general problem solver. The software design presented here does not attempt learning or self adaptation (although we are optimistic that such capabilities will become practicalities for agile manufacturing in the future).

Object-oriented programming is the leading approach to developing maintainable software. Object orientation is based on the observation that the fundamental objects of a system are typically more stable than its features or functions. The common properties of a set of related objects are characterized by defining a *class*, which is a software module that provides a set of services (operations) to other classes called *clients*, but which encapsulates the implementation of these services. Thus, a class has a public interface and a private implementation. These features promote the capability for encapsulation and information hiding, which enables code re-use.

The application of object-oriented design to manufacturing systems is not, in itself, new. Miller and Lennox [42] describe layered software in which physical objects are organized into class hierarchies. Adiga and Coge [43] define a hierarchical software architecture for conventional manufacturing. Buschmann and Meunier [44] apply the *model-view-controller* design pattern in the design of a material handling system. While this prior work contributes

to understanding applications of object orientation to manufacturing, specific design choices within an application domain are crucial to success.

Beyond encapsulation of detail for code re-use, it is also important to be able to re-use a design intent—the structure and interactions among higher-level software. A useful approach towards achieving this goal is *design patterns*, which are “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context”[45]. Design patterns capture the important relationships between objects and classes in an object-oriented design. Both classes and design patterns can be reused in new designs. This description appeals to our desire for an appropriate level of generality, and we have pursued this philosophy in our agile software design. Still, in achieving an appropriate resolution between generality and specificity, one must commit to a scope of applications. The more narrow the range of applications to be considered, the more detailed one can make the software specification, yet re-apply it to new applications within the narrow context. In contrast, there is little that can be said meaningfully about a general-purpose manufacturing system. In this section, we present our decisions for appropriate classifications within our context of agile manufacturing for light mechanical assembly. (for additional detail, see [7]).

3.1 LOW-LEVEL HARDWARE CONTROL OBJECTS

At the lowest levels, one has the greatest opportunity for encapsulating details that would otherwise inhibit code re-use. The implementation of agents which run as concurrent threads within a real-time operating system (VxWorks™) provide a buffer between these lowest-level objects that interact directly with hardware and the rest of the system. In the development of these agents, it is particularly important to create the transition from *implementation* to *behavior*. If the agents are composed correctly, then the design of higher-level software can restrict attention to *what* tasks need to be performed, not with *how* the tasks are carried out. Such a distinction enables the applications programmer to focus on the task to be accomplished and not be distracted by the overwhelming detail of a complex and mutating physical system.

The *transporter*, which is responsible for control of the Bosch conveyor system [41], 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 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 loop, in more complex cases the transporter could be responsible for any necessary parcel routing or storage between sources and destinations, and this additional complexity would be hidden from the clients as well. In fact, since the transporter interface hides the actual transportation mechanism, alternative physical means of transport (e.g., AGV's) could be used in place of the conveyor without affecting higher-level software.

Our *part-supplier* objects are 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).

We have also defined a robot motion server to abstract the services performed by robots. Robots are good examples of agile-manufacturing modules in the context of manipulation. They are capable of performing new tasks with only software changes. However, conventional robot programming interfaces are restrictive in the context of agile manufacturing. A notable limitation is that robot controllers use vendor-specific programming languages and operating systems. Further, robot manufacturers do not yet support object-oriented programming nor modern software-development tools. 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.

Preceding efforts at unifying robot control include the Robot Independent Programming Language (RIPL) from Sandia Laboratories [42] and the Open Architecture Control specification from Cimetrix [46]. 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 Figure 8, 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; 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 with the introduction of robot "proxies" [45]. Clients can communicate with our proxy abstraction of a generic virtual robot without concern for the communications medium (serial, reflective memory network or Ethernet, in our system), without concern for what type of robot is being used (AdeptOne or Adept 550, in our present system), and without knowledge of or concern for the robot's native control language (V+ or Cimetrix API's, at present).

With our motion server approach, a robot is an implementation detail. 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 robot is irrelevant to the client. We note that the effectiveness of this approach is coincident with the objectives and barriers of off-line 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 off-line programming capability.

Our fourth major low-level object is the vision server. Like robots, machine vision systems vary widely in their hardware details, their programming languages and their programming environments. Such details continue to change, as image-processing technology is advancing rapidly. Further, while the cost of machine vision is falling, it is still economically necessary to share the resources of a machine vision system among multiple cameras. These conditions impose constraints on 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. To address these needs, we have designed a vision server.

Our vision server uses 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. Our abstract-functionality model reduces machine vision systems to their core functionality, disregarding 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 vendor-specific hardware or software details of the new vision system installed. (For further details on the vision-server design, see [11,47]).

3.2 HIGHER-LEVEL CONTROL OBJECTS

Having defined the low-level control objects, objects at the higher levels can interface to the lower levels in terms of their abstract behaviors, without regard for implementation details. Our higher-level classes interact as clients of our low-level servers: transport, part-feeder, motion and vision. Our main control agent is the "assembler." An assembler object requests parts from a part supplier, requests parcels and yields parcels 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.

Our highest level entity is the “workcell manager,” which is a supervisory agent that creates assemblers, suppliers and the transporter and allocates necessary resources to them. It starts or shuts down workcell operation and communicates with the operator. It also orchestrates all the other activities that require cooperation between other agents, such as error recovery.

Figure 9 summarizes our software architecture in terms of a simplified version of our class structure in Rumbaugh *et al*'s OMT notation [48]. Our agents are designed with as few assumptions about the overall workcell structure as possible, so they are not sensitive to changes in that structure. The interaction among assemblers, transporters and suppliers defines an important pattern of behavior that we postulate is inherent in agile manufacturing applications in the context of light mechanical assembly. We call this *the assembler-transporter-supplier* design pattern.

3.3 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 inter-task 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 20ms 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. While our experience is limited to anecdotes at this point, we have observed better than 90% code re-use 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.

4 CONCLUSIONS AND FUTURE WORK:

In this summary 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)

In future work, we recognize the need for extending investigations in the following areas:

Flexible parts feeders need to be able to handle rolling parts, which is currently problematic. Further, economic viability of flexible parts feeders may require achieving higher throughput at lower cost.

Our automated gripper design process should be extended to include consideration of sliding contact between parts and fingers during gripper closure. It should be possible to sculpt the contacting surfaces to help guide parts into precise, known grasp poses. Such capability would actually improve the precision of manipulation through the act of grasping. In addition, the non-contacting gripper finger surfaces should be automatically shaped (e.g., tapered) to minimize the gripper profile and to help singulate parts to be grasped during gripper approach.

Our work table concept should be further developed such that the control software can interrogate the work table, recognize its identity, associate the corresponding driver software, and warn the operator of incorrect work table installations.

Our emulation of agile sensors should be reduced to practice. At present, we use a conventional CCD camera, a frame grabber, and image-processing software to emulate the functions of our proposed agile sensor. Realization of the agile sensor would consist of a low-cost image sensor integrated with local RAM, a local processor, and a simple go/no-go binary output signal.

Approaches to network communications within the system should also be examined. At present, our system incorporates considerable dedicated wiring, which can be overwhelming when adding new hardware or diagnosing problems. Our wiring should be dramatically simplified with the use of network communications, but an appropriate choice should be made with consideration of timing requirements, ease of physical extensibility, and ease of software extensibility.

A software area that needs further work is vision programming. Our vision server helps hide vendor-specific details of image processing. Nonetheless, for each new part a corresponding “locator” object must be written, which involves expert programming with low-level image-processing functions. Additional work is needed to ease the task of programming locator objects.

Our vision server approach should be further generalizable to accommodate multiple vision processors. Ideally, the vision server would easily incorporate additional hardware while hiding such details from its clients. The server should choose how to exploit parallelism while making such scheduling transparent to the clients.

Our greatest software research need at present is the identification of appropriate modularity, encapsulation and design patterns for automated error detection and correction. As system complexity grows, automated error detection (as well as human-interactive diagnostic guidance) is increasingly necessary. (Early steps in graphical guidance for error diagnostics have been initiated, as described in [8]).

Finally, we are currently attempting to extend the generality of our motion server to an assembly server. In the broader context, some assemblies are performed successfully through precision manipulation, but other assemblies require response to sensed forces and moments of interaction to guide parts mating. Extensions of assembly to include effective and reliable use of force sensing is an ambitious undertaking, and we have only begun to address this need.

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.

5 ACKNOWLEDGMENTS:

We gratefully acknowledge financial support from the Cleveland Advanced Manufacturing Program (CAMP, Inc.), the Center for Automation and Intelligent Systems Research, the Case School of Engineering, and our industrial sponsors for this work. In addition to the authors listed, the following current or former faculty and students have contributed substantially to our agile-manufacturing project: Leon Sterling, Seungwoo Kim, Scott Ameduri, Rich Bachman, Matt Birch, Ed Blanchard, Chrysanthie Chamis, Siddharth Chhatparr, Ling Huang, Ju-Yeon Jo, Yalcin Karagoz, Sreeram Ramasubramanian, David Sargent, Terrance Wei, Hao Zhang, Wei Zhang, and Gang Zhao.

6 REFERENCES:

- [1] Quinn, R.D., *et al.* *Design of an Agile Manufacturing Workcell for Light Mechanical Applications*. In *IEEE International Conference on Robotics and Automation*. 1996. Minneapolis, MN: IEEE. pp. 858-863.
- [2] Quinn, R.D., *et al.*, *Design of an Agile Manufacturing Workcell for Light Mechanical Applications*, 1996, Video Proceedings of the *IEEE International Conference on Robotics and Automation*: Minneapolis, MN.
- [3] Quinn, R.D., *et al.*, *An Agile Manufacturing Workcell Design*. IIE Transactions, 1997. : pp. 901-909.
- [4] Quinn, R.D., *et al.*, *Advances in Agile Manufacturing*, 1997, Video Proceedings of the *IEEE International Conference on Robotics and Automation*: Albuquerque, NM.
- [5] Merat, F.L., *et al.* *Advances in Agile Manufacturing*. In *IEEE International Conference on Robotics and Automation*. 1997. Albuquerque, NM: IEEE. pp. 1216-1222.
- [6] Causey, G.C., *et al.* *Design of a Flexible Parts Feeding System*. In *IEEE International Conference on Robotics and Automation*. 1997. Albuquerque, NM: IEEE. pp. 1235-1240.
- [7] Kim, Y., *et al.* *A Flexible Software Architecture for Agile Manufacturing*. In *IEEE International Conference on Robotics and Automation*. 1997. Albuquerque, NM: IEEE. pp. 3043-3047.
- [8] Jo, J.Y., *et al.* *Virtual Testing of Agile Manufacturing Software Using 3D Graphical Simulation*. In *IEEE International Conference on Robotics and Automation*. 1997. Albuquerque, NM: IEEE. pp. 1223-1228.
- [9] Velasco, V.B.Jr. *et al.* *Computer-Assisted Gripper and Fixture Customization via Rapid Prototyping*. In *IASTED International Conference on Robotics and Manufacturing*. 1997: IASTED. pp. 115-120.

- [10] Velasco, V.B.Jr. *et al.* *Computer-Assisted Gripper and Fixture Customization using Rapid Prototyping Technology*. In *IEEE International Conference on Robotics and Automation*. 1998. Leuven, Belgium: IEEE. pp. 3658-3664.
- [11] Barendt, N.A., *et al.* *A Distributed, Object-Oriented Architecture for Platform-Independent Machine Vision*. In *IASTED International Conference on Robotics and Manufacturing*. 1998: IASTED. pp. 50-55.
- [12] Newman, W.S., *et al.* *Technologies for Robust Agile Manufacturing*. In *IASTED International Conference on Robotics and Manufacturing*. 1998: IASTED. pp. 56-61
- [13] Causey, G.C and Quinn, R. D. *CWRU Flexible Parts Feeding System*, 1998, Video Proceedings of the *IEEE International Conference on Robotics and Automation*: Leuven, Belgium.
- [14]] Newman, W.S., *et al.* *Technologies for Robust Agile Manufacturing*., 1998, Video Proceedings of the *IEEE International Conference on Robotics and Automation*: Leuven, Belgium.
- [15] Horn, B. *Robot Vision*, McGraw-Hill, 1986: Chapter 18
- [16] Kelly, R.B., *et al.* *A Robot System which Acquires Cylindrical Workpieces from Bins*. In *IEEE Transactions on Systems, Man and Cybernetics*. Vol. 12, No. 2, pp. 204-213.
- [17] Schmidt, W. *Multiple Part Orientation Using a CVIM System*. Rockwell Automation - Allen Bradley. Presented at the *RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop*, 1995
- [18] Bailey, C.E. *Strategies for Flexible Parts Feeding*. Delco Electronics Corporation. Presented at the *RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop*, 1995
- [19] Comtech. MRW Robo-Pot. Company Product Literature. Farmington Hills, MI 48333
- [20] Davis, W. F. *Centrifugal Feeders The Flexible Approach*. Custom Systems Integration Co. Presented at the 1994 RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop, 1994
- [21] US Patent #4333558. Nonaka, Takeyoshi and Otsuka, Kazuo. *Photoelectric Control System for Parts Orientation*. June 8, 1982
- [22] US Patent #4608646. Goodrich, Jerry L. and Devlin, W. L. *Programmable Parts Feeder*. August 26, 1986
- [23] US Patent #4712974. Kane, Peter E. *Part Positioning Apparatus and Method*. December 15, 1987
- [24] US Patent #4619356. Dean, Arthur L., et.al. *Programmable Parts Feeder*. October 28, 1986
- [25] Mahoney and Marshall, *Tape and Reel Technology for Automatic Packaging and Feeding of Connectors*. GPAX International, Inc. Company Product Literature. Columbus, Ohio.
- [26] Technical Presentation, Sankyo Seiki Mtg. Co., LTD. Presented at the *RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop*, 1994
- [27] SonyFA, *Advanced Parts Orientating System, T-APOS*. Company product Literature. 560 Route 303 Rangeburg, New York 10962.
- [28] Goldberg, Mason, and Erdmann. *Generating Stochastic Plans for a Programmable Parts Feeder*. *Proceeding of the 1991 International Conference on Robotics and Automation*. Sacramento, California, April 1991

- [29] Brokowski, Peshkin, and Goldberg. *Curved Fences for Part Alignment. Proceeding of the 1993 International Conference on Robotics and Automation.*, 1993
- [30] Yoneda, Kenji. *Bottle Unscrambler. Proceedings of the RIA Flexible Parts Feeding for Automated Handling and Assembly Workshop.* Cincinnati, Ohio, October 1994 .
- [31] US Patent #5370216. Tsuruyama, Katsuji., *et.al. Apparatus for Aligning Vessels.* December 6, 1994
- [32] Boehlke, D. *Smart Design for Flexible Feeding. Machine Design,* December 12, 1994.
- [33] Adept Robotics, Adept FlexFeeder 250, Company Product Literature, 150 Rose Orchard Way, San Jose, CA, Sept. 1995.
- [34] Causey, G.C. *Elements of Agility in Manufacturing.* Ph.D. Dissertation. Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland Ohio, Jan 1999.
- [35] Mason, M. T., and Salisbury, J. K., *Robot Hands and the Mechanics of Manipulation,* The MIT Press, Cambridge, Massachusetts, 1985.
- [36] Cawley, J. D. *et al. Al2O3 ceramics made by CAM-LEM (computer-aided manufacturing of laminated engineering materials) technology. Solid Freeform Fabrication Symposium Proceedings,* pp. 9–16, Austin, TX, August 1995.
- [37] Mathewson, B. B. *et al. Automated fabrication of ceramic components from tape-cast ceramic. Solid Freeform Fabrication Symposium Proceedings,* pp. 253–260, Austin, TX, August 1995.
- [38] Reuleaux, F. *The Kinematics of Machinery.* New York: Macmillan, 1876; reprint, New York: Dover, 1963.
- [39] Trinkle, J. C. *A Quantitative Test for Form Closure Grasps. IEEE Int. Conf. on Intelligent Robots and Systems,* pp. 1245–1251, July 1992.
- [40] Brown, R. G. and Brost, R. C. *A 3-D Modular Gripper Design Tool. Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 2332–2339, 1997.
- [41] Bosch Automation Technology, 7505 Durand Avenue, Racine, WI 53406 . URL: <http://www.boschautomation.com>
- [42] Miller, D. J. and Lennox, R. C. *An Object-Oriented Environment for Robot System Architectures. In IEEE Control Systems,* Feb 1991.
- [43] Adiga, S. and Coge, P. *Towards an Object-Oriented Architecture for CIM Systems. Object-Oriented Software for Manufacturing Systems,* Chapman & Hall 1993.
- [44] Buschmann, F. and Meunier, R. *Building a Software System. In Electronic Design,* Feb 20, 1995, pp.132-144
- [45] Gamma, E., *et al., Design Patterns: Elements of Reusable Object-Oriented Software.* 1994, Reading, Massachusetts: Addison-Wesley.
- [46] Anderson, R.L., *Open Architecture Controller Solution for Custom Machine, Systems. Proceedings of SPIE,* 1996: pp. 113-127.
- [47] Barendt, N. A. *a Distributed, Object-Oriented Software Architecture for Platform-Independent Machine Vision.* MS Thesis, Department of Electrical Engineering, Case Western Reserve University, Cleveland, OH, May 1998
- [48] Rumbaugh, J., *et al., Object-Oriented Modeling and Design,* Prentice Hall, Englewood Cliffs, NJ, 1991.



Figure 1: CWRU Agile Manufacturing cell

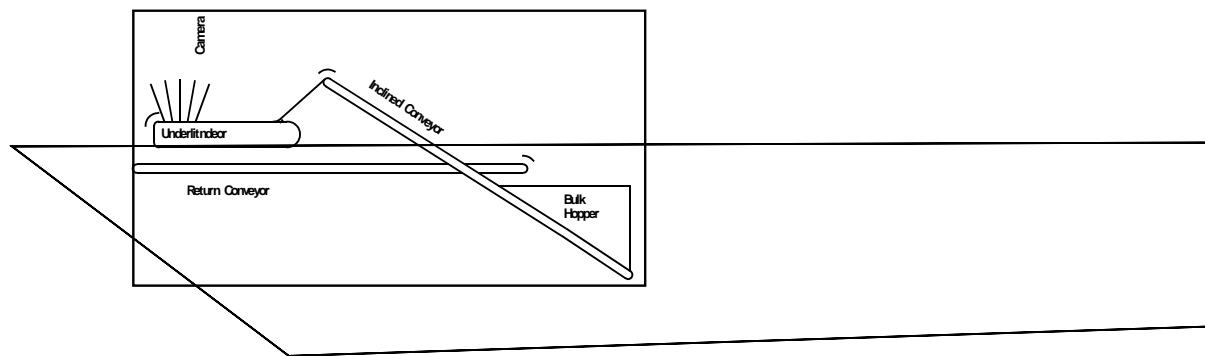




Figure 3: left: LEGO blocks drawn from the hopper and ascending the inclined conveyor
right: Sockets spilling from the inclined conveyor onto the horizontal conveyor

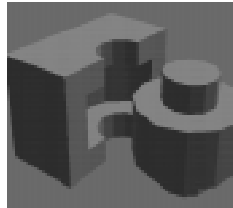


Figure 4: Illustration of Finger shape
(Cavity complimentary to part shape)



Figure 5: left: workpieces used in developing the experimental multi-function finger design;
right: fabricated fingers grasping a plastic socket

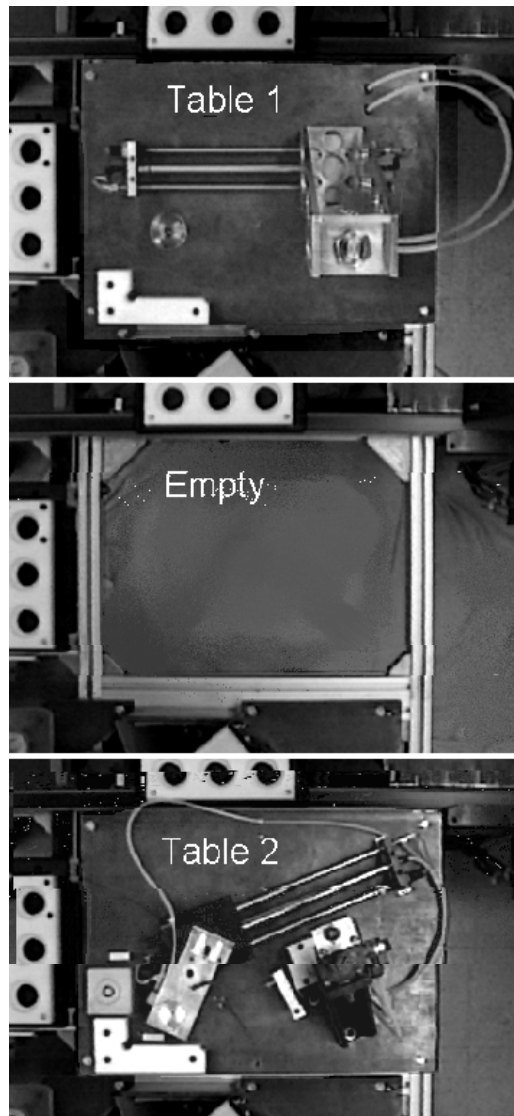


Figure 6: Worktable Changeout Sequence

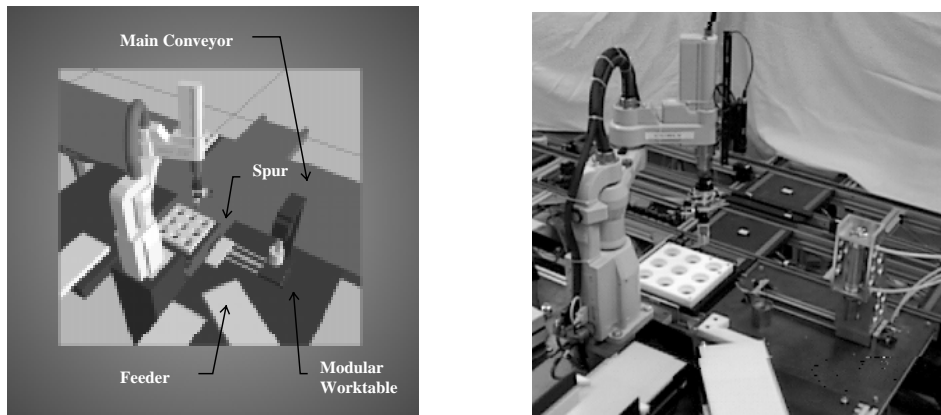


Figure 7: Assembly Station Layout left: Simulation ; right: implementation

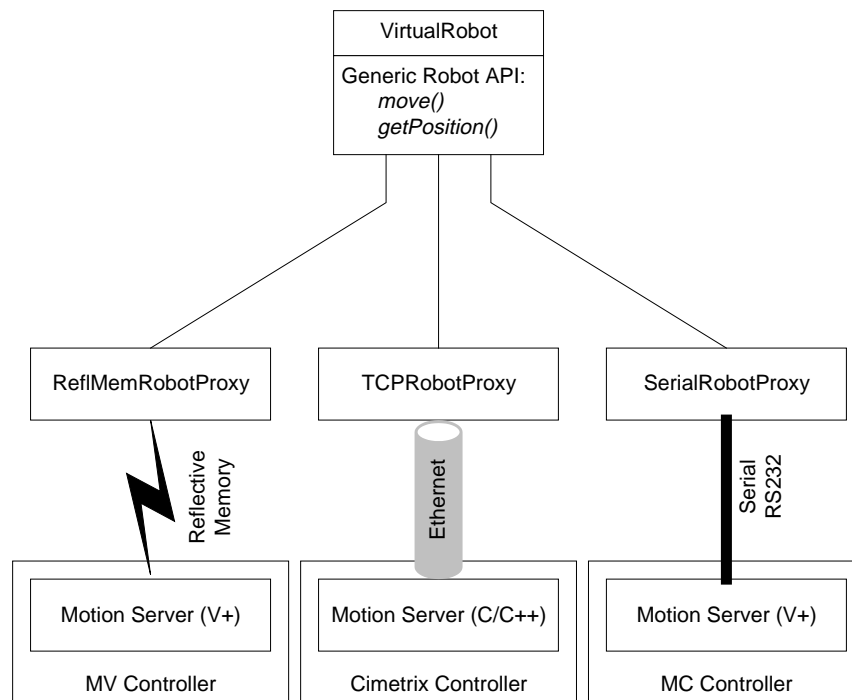


Figure 8: Proxy Abstraction of a Generic (Virtual) Robot

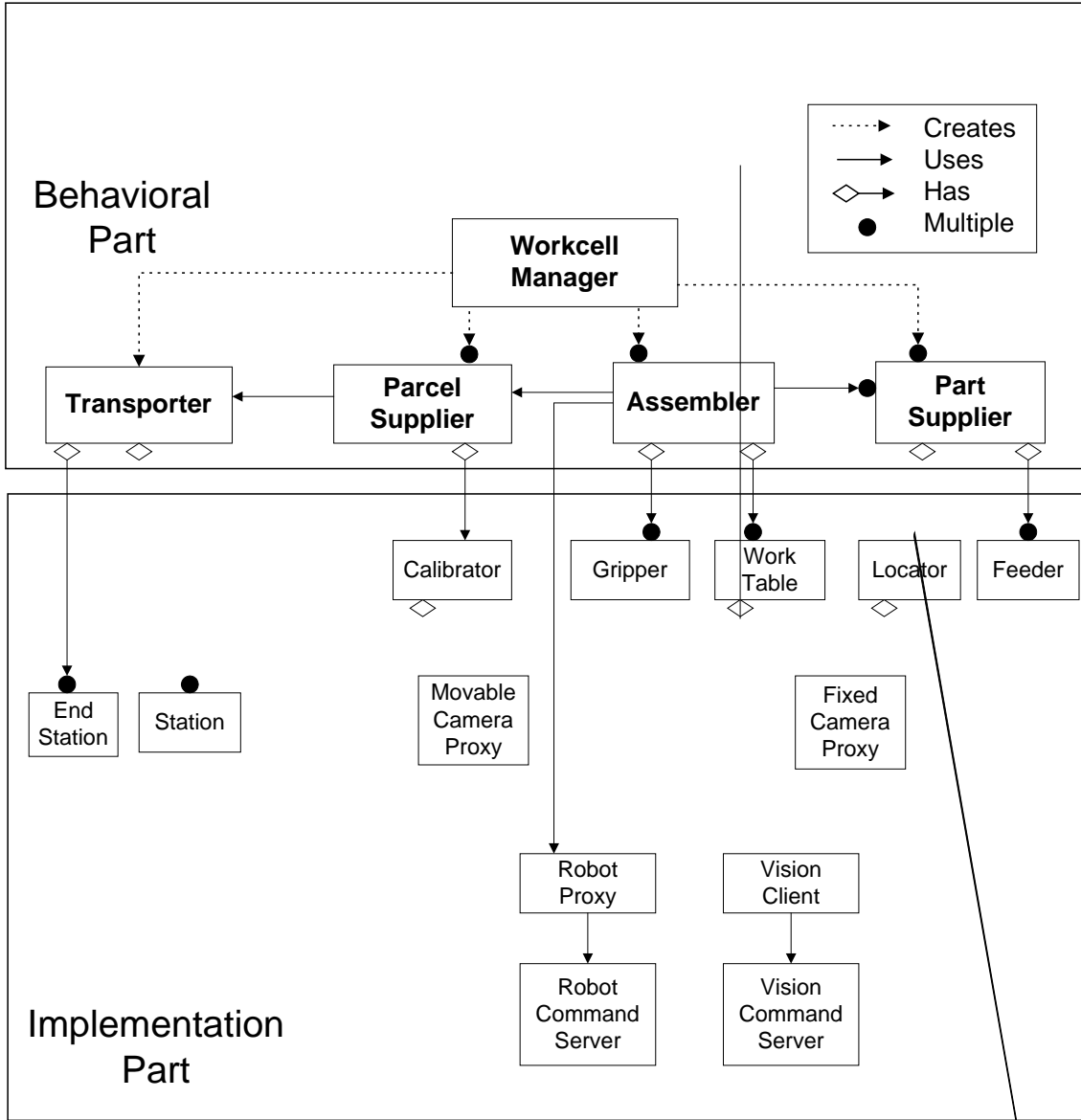


Figure 9: System Class Diagram

