Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

A Flexible Software Architecture for Agile Manufacturing

Yoohwan Kim[†], Ju-Yeon Jo[†], Virgilio B. Velasco Jr.[‡], Nicholas A. Barendt[‡],
Andy Podgurski[†], Gultekin Ozsoyoglu[†], and Frank L. Merat[‡]
[†]Computer Engineering & Science Department
[‡]Electrical Engineering & Applied Physics Department
Center for Automation and Intelligent Systems Research
Case Western Reserve University
Cleveland, Ohio 44106

Abstract

The flexibility required of an agile manufacturing system must be achieved largely through computer software. The system's control software must be adaptable to new products and to new system components without becoming unreliable or difficult to maintain. This requires designing the software specifically to facilitate future changes. As part of the Agile Manufacturing Project at Case Western Reserve University, we have developed a software architecture for control of an agile manufacturing workcell, and we have demonstrated its flexibility with rapid changeover and introduction of new products. In this paper, we describe the requirements for agile manufacturing software and how our software architecture addresses them.

1 Introduction

For our purposes, *agile* or *flexible* manufacturing facility is one which supports the following capabilities:

- rapid changeover between products
- rapid introduction of new products
- unattended operation

To reduce costs and delays, an agile manufacturing system should be designed so that hardware changes are minimized. Agile manufacturing requires the use of technologies that can adapt to a variety of products without extensive retooling. These technologies include computers, general-purpose robots, machine vision systems, sensors, conveyors, and flexible parts-feeders. Integration and overall control of an agile manufacturing system are embodied in computer software, and this software provides a large measure of the system's flexibility. To the extent possible, product changeover is reduced to loading appropriate software, and introduction of new products is accomplished by modifying software. Thus, the control software for an agile manufacturing system must undergo frequent modification. Although software is typically more malleable than hardware, changes can gradually degrade its structure, so that it becomes unreliable and difficult to maintain. To avoid this, software must be designed specifically to facilitate maintenance.

As part of the Agile Manufacturing Project at Case Western Reserve University (CWRU) [1, 2], we have developed a flexible and robust software architecture for workcell control. *Object-oriented* design techniques were employed to facilitate rapid changeover and rapid introduction of new products. The architecture identifies fundamental objects and operations of agile manufacturing, and it describes their relationships in terms of *design patterns* [3]. Hardware and software dependencies have been isolated as much as possible. To support unattended operation, a hierarchical framework has been developed for error handling. The architecture has been implemented using the C^{++} programming language and a commercial real-time operating system (VxWorks) conforming to the POSIX standard. The flexibility of the architecture has been demonstrated by rapid changeover between current products and by introduction of new products.

2 CWRU Workcell Overview

2.1 Workcell Hardware

A testbed implementation of an agile manufacturing workcell has been developed at CWRU for light mechanical assembly applications. It includes multiple Adept SCARA robots mounted near a central conveyor system. These are augmented with flexible parts feeders, a vision system, sensors, and limited special purpose hardware. All of these are controlled by the workcell software. Each robot does partial assembly work, and subassemblies are placed in fixtures on conveyor pallets for transportation to other robots. Parts are fed to robots on belt conveyors which terminate at

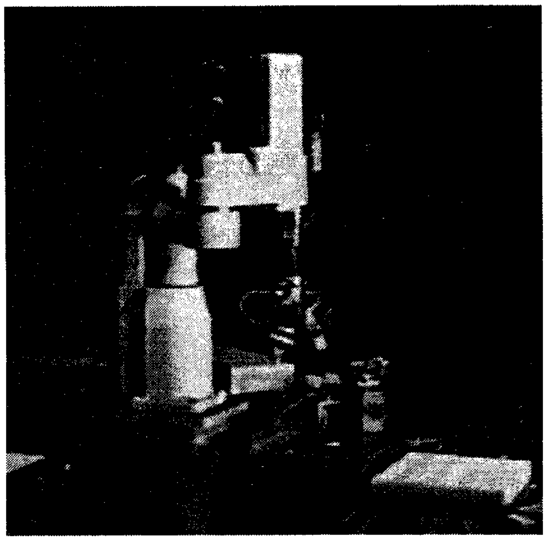
Figure 1: Workcell Hardware

an underlit window. Part position and orientation are identified by the vision system. Rapid changeover is achieved through the use of multipurpose grippers, modular wrist tool-change connectors, and modular worktables. Figure 1 shows the overview of the agile workcell.

2.2 Controller

The entire workcell is controlled using a dual VMEbus architecture, where robot motion control and vision processing are assigned to an Adept MV controller and all other operations, such as assembly sequencing and I/O, are performed by third-party boards in a non-Adept VMEbus. This innovative, open architecture allows us to exploit the MV controller's advanced vision and robot control capabilities without adopting Adept's proprietary V^+ programming language and operating system as our principal software development platform. The workcell is controlled primarily by a C^{++} program running under the VxWorks real-time operating system (RTOS) on the non-Adept VME bus. The two VMEbuses are connected by a *reflective memory network*, in which changes made to a memory module on one bus are automatically reflected on the other bus. Requests from the primary control program for vision and robot motion services are made to a vision server and command server, respectively, that run on the non-Adept bus. These are local proxies that service requests by communicating them to the Adept system via reflective memory. Their function is to isolate dependencies on the Adept system. We are currently moving responsibility for vision processing to the non-Adept side as well.

3 Design Goals

- **Design for change:** The software must accommodate future changes and its reconfiguration process must be quick. The scenarios of possible changes must be studied and necessary modification procedures should be well planned and documented.
- **Design for lower complexity:** We expect our software will grow continuously as new products are added and structural changes are applied to the workcell. Because complex systems are less maintainable, controlling complexity was another goal in our design.
- **Design for reuse:** Another factor in reducing software cost is reuse. It is easier and faster to use existing software components rather than developing them from the scratch. Well-defined software components are also less error-prone because they usually undergo thorough unit testing and are field-tested by users.

4 Software Engineering Methodology

Developing maintainable software entails anticipating changes in requirements and *encapsulating* design decisions within software modules, so that modifications can be localized. Encapsulation or *information hiding* is a fundamental principle of *object-oriented* software design and programming. 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. Solutions to important design problems can be expressed as *design patterns*, which are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context" [3]. 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.

5 System Structure

The primary source of change in an agile manufacturing system is the introduction of new products. Another important source of change is the introduction of new workcell components such as robots, conveyors, and vision processors. Our software architecture encapsulates these sources of change within classes. The architecture defines a hierarchy of servers, ranging from high-level agents that control

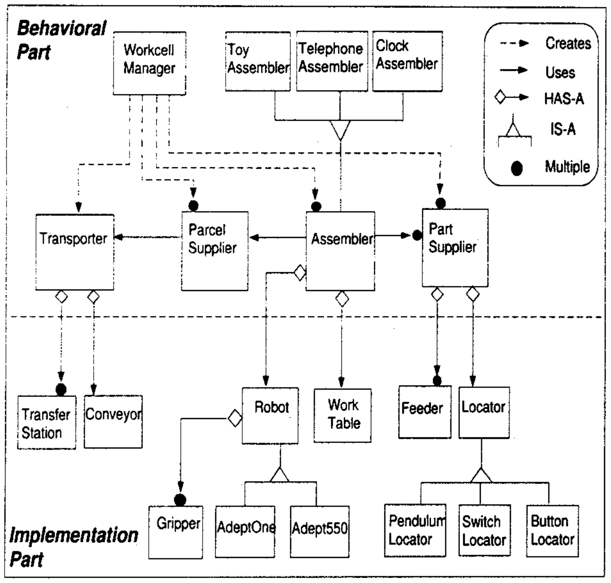

Figure 2: Simplified Class Diagram

entire subsystems down to simple objects that provide an interface to physical devices. At the top of this hierarchy is the Workcell Manager, which initiates operation of the workcell by instantiating its objects. Immediately below it are the agents which control the principal subsystems of the workcell. These agents are Assemblers, Suppliers, and the Transporter. Their interactions define the workcell's high-level operation. We now describe each of these objects in more detail:

- The Workcell Manager 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.
- An Assembler is responsible for making partial or complete assemblies. Partial assemblies made by one Assembler are completed by another. An Assembler requests parts and subassemblies from Suppliers. It encapsulates an assembly sequence and directly or indirectly employs a robot, vision system, parts feeders, and possibly special purpose hardware.
- A Transporter provides general purpose transportation of parts, assemblies, or other material around the workcell. It services requests from Suppliers. In the current system, there is one Transporter, which uses a Bosch conveyor to move pallets of partial or complete assemblies between work stations. However, the Transporter interface hides the actual transportation mechanism, so that different ones may be used without affecting the rest of the system. A Transporter may need to route or to store parcels between their source and destination, although this is unnecessary with the current workcell, whose conveyor forms a simple loop.
- A Supplier services request for parts or assemblies. There are two basic kinds of Suppliers: Part Suppliers and Parcel Suppliers. A Part Supplier provides the coordinates of a part or subassembly to an Assembler. Some Part Suppliers invoke a flexible parts feeder and the vision system to locate a part. Others request a parcel of subassemblies from a Parcel Supplier, which obtains the subassemblies from an Assembler and invokes the Transporter to move them to where they are needed.

These 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 between Assemblers, Transporters, and Suppliers defines an important pattern of behavior that is present in most agile manufacturing applications. We call this the *Assembler-Transporter-Supplier* design pattern.

Figure 2 shows a simplified version of class structure in Rumbaugh *et al*'s OMT notation [4].

Some notable auxiliary objects include Robots, Feeders, and Part Locators:

- A Robot provides an interface to a physical robot. The Robot class interface contains operations common to most robots, such as *move to*, *open gripper*, and *close gripper*. Subclasses of Robot are defined for each type robot used in the system, e.g., AdeptOne and Adept550. Robots are employed by Assemblers.
- A Feeder controls a flexible parts feeder to bring parts under the view of a camera. Feeders are employed by Part Suppliers.
- A Part Locator determines the location and orientation of a part, using the vision system. Part Locators are employed by Part Suppliers.

6 Error Handling

To achieve the goal of unattended operation, an agile manufacturing system must incorporate robust error handling. This can add considerable complexity to the control software. It is not unusual for error handling code to make up 80% of conventional control programs. To simplify error handling in the CWRU agile workcell, we have developed a hierarchical framework for error handling. In this framework, errors are handled locally if possible. If they cannot be handled locally, then responsibility for handling them is passed up to higher-level agents. For example, if a part cannot be found by a Part Locator, it can attempt to adjust parameters of the vision system to obtain a better image. If this does not work, it reports failure to the Part Supplier that invoked it. If that Supplier can obtain the part elsewhere, e.g., from another Feeder, it will do so. Otherwise, it must report failure to the Assembler that invoked it. If another Supplier exists for the part, the Assembler will try it; otherwise the Assembler will report failure to the Workcell Manager, which could bypass the Assembler or notify plant personnel.

We are currently working to enhance the error handling capabilities of the CWRU agile workcell. This involves incorporating additional sensors and redundant subsystems.

7 Implementation Issues

7.1 Software Development Platform

The control software for an initial prototype of the agile workcell was implemented with Adept's V^+ programming language and operating system. A number of problems were revealed during this effort. Being proprietary, V^+ does not lend itself to an open system. The V^+ programming language and software development environment are rather primitive, and they do not support object-oriented programming. Finally, the V^+ operating system does not offer the task-management facilities of a full-fledged RTOS. It was concluded that V^+ was unsuitable for our purposes, which are not typical of current robotic applications. Instead, the bulk of our control system was implemented in the ANSI C^{++} programming language, using the POSIX-compliant RTOS VxWorks. By choosing a standardized programming language and operating system interface, we obtained a significant measure of platform independence. Moreover, advanced software development tools are available for use with C^{++} and VxWorks.

The current software was developed on Sun workstations under the UNIX operating system. The Revision Control System (RCS) was used for configuration management. The software is currently composed of about 40 classes and its size is roughly 8,000 lines of C^{++} source code.

7.2 Concurrent Programming

The active agents of the system, such as Assemblers, Suppliers, and the Transporter, operate concurrently and are implemented as RTOS tasks. However, this is invisible to clients, which interact with the agents through their C^{++} class interfaces (C^{++} does not directly support concurrency). Clients need not make RTOS system calls to create, schedule, synchronize, or communicate between tasks; this is done by the class implementation. Priority scheduling has proven unnecessary for the current system, because mechanical operations introduce considerable slack time. Presently, 18 tasks run concurrently in round-robin fashion with 20 *ms* time slots.

7.3 Data Logging

System status is logged using commands provided by VxWorks. Logged items include: commands sent to the robot motion servers; the assembly status for each robot; transportation status, etc. Data is logged with time-stamps, then analyzed in real-time by the Agile Database Server [5] for performance measurements or to answer remote queries.

8 Maintainability

To illustrate the maintainability of our software architecture, we now consider how it is affected by the introduction of a new product or a new hardware component.

- **New product**
  One or more new subclasses of Assembler must be defined to assemble the product. If multiple subclasses are used, each is responsible for part of the assembly. New subclasses of Gripper and Work Table may have to be defined to interface with new robot grippers and modular worktables. If the product contains new parts, new subclasses of Locator must be defined to find them using the vision system. This can usually be done with standard vision routines. Finally, a new Workcell Manager is necessary to instantiate new classes and start workcell operation. In our experience, only about 600 lines of code must be written to accommodate a new product; this can be done in a day or two, although more time is necessary for testing.

- **New hardware component**
  We consider two examples of new hardware:

  1. **New robot:** This entails implementing a new subclass of Robot. This subclass provides the same operations as other Robot subclasses like AdeptOne and Adept 550, but its implementation is different.
  2. **New transportation mechanism:** If the backbone conveyor is replaced with a radically different system, e.g., mobile robots, new classes must be developed for controlling the new hardware, and the implementation part of Transporter must

be changed to use these classes. However, the interface of Transporter is unchanged, so clients are not affected. Transporter and its associated classes currently comprise about 1,000 lines of code.

9 Related Work

Several applications of object-oriented design to manufacturing systems have been described previously in the literature: Miller and Lennox [6] describe layered software in which physical objects are organized into class hierarchies; Adiga and Cogez [7] define a hierarchical software architecture for conventional manufacturing; Buschmann and Meunier [8] apply the *Model-View-Controller* design pattern in the design of a material handling system. However, these papers do not address the issues of agile manufacturing.

10 Conclusions

Quickly reconfigurable software is crucial for the success of agile manufacturing. At CWRU, we have developed an extensible software architecture for controlling an agile manufacturing workcell and we have demonstrated its flexibility. This architecture is intended to support light mechanical assembly, but it should be applicable to other agile manufacturing applications as well. We hope that it will contribute to the development of one or more standard architectures for this important family of applications.

11 Acknowledgements

This work was supported by the Cleveland Advance Manufacturing Program (CAMP) through Center of Automation and Intelligent Systems Research (CAISR) and the Case School of Engineering.

References

[1] R.D. Quinn, G.C. Causey, F.L. Merat, D.M. Sargent, N.A. Barendt, W.S. Newman, V.B. Velasco Jr., A. Podgurski, J.Y. Jo, L.S. Sterling, Y.H. Kim, "Design of an Agile Manufacturing Workcell for Light Mechanical Applications," *Proc. of IEEE International Conference on Robotics and Automation*, 1996, pp.858-863.

[2] R.D. Quinn, G.C. Causey, F.L. Merat, N.A. Barendt, W.S. Newman, V.B. Velasco Jr., A. Podgurski, Y.H. Kim, G. Ozsoyoglu, J.Y. Jo, "Advances in Agile Manufacturing," *Proc. of IEEE International Conference on Robotics and Automation*, 1997.

[3] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.

[4] J. Rumbaugh, M. Blaha Gamma, W. Premerlani, F. Eddy, and W. Lorenson, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[5] Sungkil Lee, et al., "A Database Server for an Agile Manufacturing System with or without Time Constraints," *Conference on Agile and Intelligent Manufacturing Systems*, Troy, NY, October, 1996.

[6] David J. Miller and R. Charleene Lennox, "An Object-Oriented Environment for Robot System Architectures," *IEEE Control Systems*, Feb. 1991, pp.14-23.

[7] S. Adiga and P. Cogez, "Towards an Object-Oriented Architecture for CIM Systems," *Object-Oriented Software for Manufacturing Systems*, Chapman & Hall, 1993.

[8] Frank Buschmann and Regine Meunier, "Building a Software System," *Electronic Design*, February 20, 1995, pp.132-144.