SOFTWARE FOR INTEGRATED MANUFACTURING SYSTEMS Part I

A. W. Naylor and R. A. Volz
The Robotics Research Laboratory
The University of Michigan

Abstract

For several years the University of Michigan has been developing a broad, unified approach to programming manufacturing cells, factory floors, and other manufacturing systems. It is based on a blending of distributed Ada, software components, generics and formal models. Among other things the machines and devices which make up the components, and the entire manufacturing cell—machines, devices, software—is viewed as an assembly of software components. The purpose of this project is to reduce the cost, increase the reliability and increase the flexibility of manufacturing software.

This paper gives an overview of the approach and describes an experimental generic factory floor controller that has been developed using the approach. The controller is "generic" in the sense that it can control any one of a large class of factory floors maing an almost arbitrary mix of parts.

1 Introduction

The basic difficulties with current software for integrated manufacturing system is that it is too expensive, too inflexible, and needs greater reliability. For the past five years the University of Michigan has been developing an approach to this software which attempts to address these difficulties. This paper reviews this approach and then discusses an experiment which uses the approach.

2 The Approach

The approach is based on five assumptions or beliefs.

1. Manufacturing software should be in the mainstream of modern software.

   It is unrealistic to expect to solve the problems of manufacturing software if we try to develop solutions that are peculiar to manufacturing. Manufacturing software is—after all—software and most of its problems are problems shared by software in general. Manufacturing software must take advantage of the tools and techniques being developed by modern software engineering. For example, manufacturing software should be written in modern general purpose languages and not tailored "manufacturing languages."

2. Software should be created as an assemblage of software components.

   In other words, we should use object oriented programming. For example, the programmer should be able to view a robot, vehicle, material handling system, or a factory floor as a software component. The programming should be concerned with two things: the interface to the component and how the component works, that is, its semantics. Further, there should be orderly ways to assemble components to create new, larger components, example, create a cell component from machine and robot components.

   The advantages are that (a) components can be reused and replaced thereby decreasing cost and increasing flexibility. Further, the object-oriented approach will increase software reliability.

3. This should be done in a largely common—eventually distributed—language environment.

   The use of object-oriented programming really requires a common language environment. However, this does not mean that portions of a large software system cannot be written in other languages. For example, NC machines will undoubledly be programmed using parts programming languages. These will be encapsulated into software components which externally present a public interface in the common language environment.

   Since manufacturing systems can involve hundreds or even thousands of programmable devices and these will be able to communicate with one another, we are inevitable faced with distributed systems. Rather than writing many separate programs which communicate with one another, we believe the entire system should be written as one (of course, highly structured) program

in a distributed language. The advantages are that (1) it relieves the programmer of writing communication software, (2) allows the programmer to think about the program at a level which largely suppresses the processor boundaries, and (3) allow the language translation system to check for bugs across the entire software system.

4. Explicit formal semantic models are required.

   Much of manufacturing software is concerned with real-time control of manufacturing systems, and real-time control inevitably requires a model for the controlled system. For example, the control software for a factory floor requires an understanding of how the factory floor works, that is, an understanding of its semantics. Thus, in addition to using software components, we must also be able to model their semantics.

5. Generics will amplify software reusability.

   By "generics" we mean skeletons for software components which can be instantiated as actual components. The instantiation process requires that information be supplied which allows the skeleton to be fleshed-out into an actual component. for example, one can imagine a generic material handling system which requires information describing the vehicles and the path layout. This would allow the same software to be used with different fleets of vehicles and different path layouts.

6. The experiment.

   We have developed a generic factory floor controller. It expects to be given a model of the factory floor, process plans, and orders. Based on this information, the generic factory floor controllers determines the appropriate sequence of commands to the factory floor. This is done in real-time. The basic control algorithm is a search algorithm which explores possible future scenarios and selects the best next step, and then carries out the cycle again.