ANNALS of the ORADEA UNIVERSITY.
Fascicle of Management and Technological Engineering, Volume VI (XVI), 2007

A SOFTWARE FRAMEWORK FOR WORK CELL ARCHITECTURE

Ilie Octavian POPP, Ioan BARSAN
"*Lucian Blaga*" University of Sibiu, Faculty of Engineering "*Hermann Oberth*"
Departaments of Machine and Machinery, Emil Cioran St. no. 4, Sibiu-2400
E-mail: **ilie.popp@ulbsibiu.ro**, Tel. 0269-216062/454

Key words: FMS, work cell controller, architecture model, software.

Abstract: This paper present a framework for organizing resources consisting of hardware devices (such as machine tools, robots, conveyors, etc) and software modules such as (cell controller, monitoring software) in a CIM environment has been developed. In this section, we focus on the basic building blocks of the framework.

1. INTRODUCTION

The problem of supervisory control/synchronization in a flexible-manufacturing environment is one of the most difficult problems designers'face in the conceptualizing of a Flexible Manufacturing System. It is clear that manufacturing flexibility induces complexity. For example, consider a Flexible Manufacturing System that supports ten different job types, which has ten different machines, each with a buffering capacity of ten. To design and implement a supervisory controller that allows for reconfiguration of resources is a challenging problem. It requires the support provided by a reference architecture model that facilitates the concept of "Plug and Play".
As explained previously in the section on Resource Models, the resources are classified into four different categories based on their functionality viz.

1) Storage
2) Ports
3) Processors
4) Transportation

As defined earlier, a work cell is a composite member composed of some of these basic resource. Figure 1 describes how recursive composition can be used so that the same framework could be used to build a simple work cell composed only of basic resources (leaf nodes) or a complex work cell that is defined recursively in terms of other work cells. Each of the resources in the work cell (including the work cell) is a CORBA object so that it can plugged into a distributed environment with minimal ease. Though the resources of a work cell have been classified into the four basic resources, it is possible to model systems where such a strict taxonomy is not applicable. *Some sample examples on how to model some typical systems are given later in the paper.*

2. WORK CELL BASIC RESOURCE.

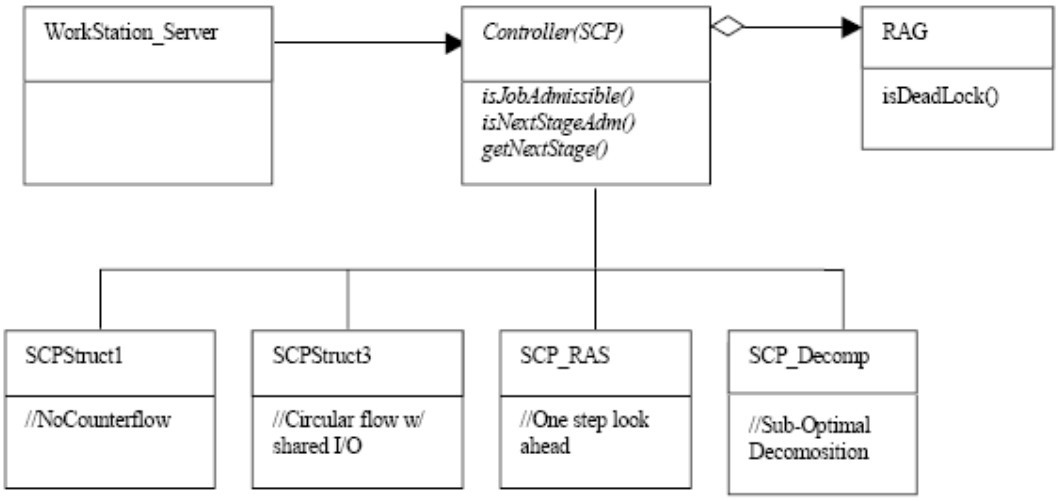

Figure 1: Plug-in supervisory controller used in the work cell (strategy pattern)

Figure 2 captures the use of the strategy pattern in the work cell. The work cell controller uses different structural control policies depending on the work cell configuration and job routes. The four different controllers that could be plugged into the work cell are:

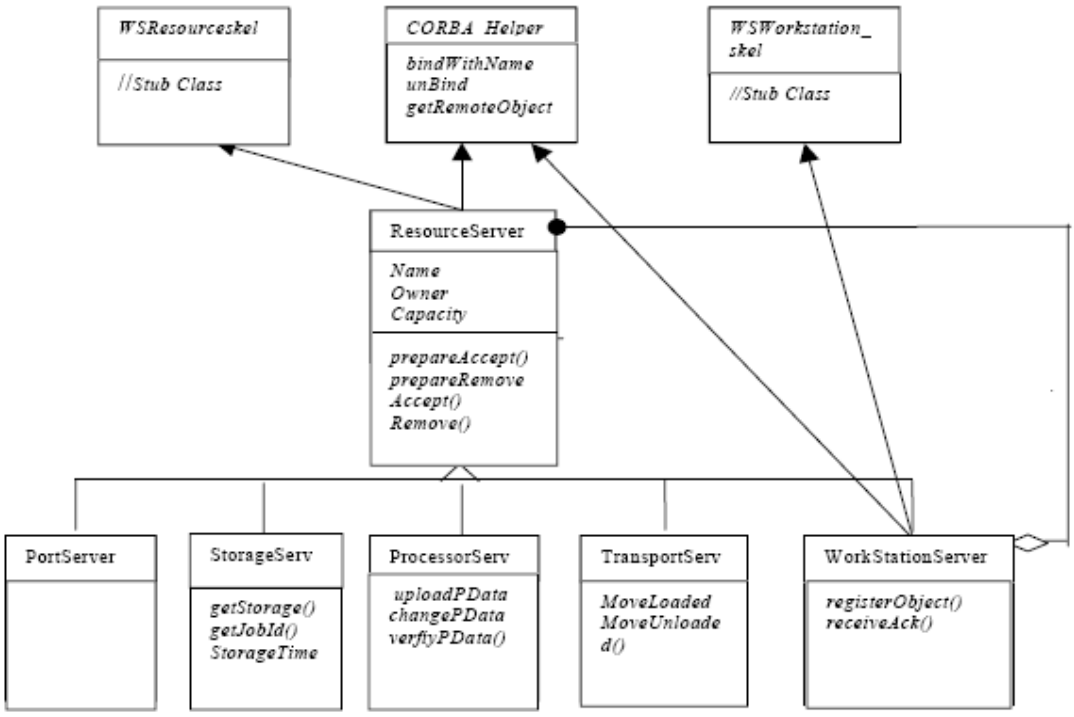

Figure 2: Work cell – Resource Part Hierarchy (Composite Pattern)

1. SCP1 - Used in the absence of counter-flow jobs. Deadlock avoidance is trivial. Suffices to check for capacity availability on device.

2. SCP2 - Used in case of a circular flow and shared input/output. The total number of jobs has to be less than the sum of capacities of all the resources at all times.
3. SCP3(SCP_RAS) - Optimal One step look ahead policy. Used when the capacity of the device is greater than one (or belongs to the special class of RAS's defined in [3], [4].
4. SCP4(SCP_Decompose)- Default case. Handles the most general case by route decomposition.

The use of such a design enables us to plug different supervisory controllers into the work cell server. The new controller has to be derived from the abstract class SCPController. That is to say, the controller has to have the same interface as the SCPController so that the rest of the classes can be reused as is.

The OMT diagram in Figure 3 explains the design of the device drivers for the devices. There might be a need to plug different device simulators into our architecture. However, the interface of each of these simulators might not be available to the resource module expects. We therefore define a device driver object that acts as an adapter for the simulator objects. The device driver publishes the interface expected by the resource module and is linked with the simulator so that it can make the appropriate calls on the simulator for each of its method. This requires that new device drivers have to be written for device simulators that follow their own proprietary protocol. This design offers us more flexibility and more reusability of our framework.

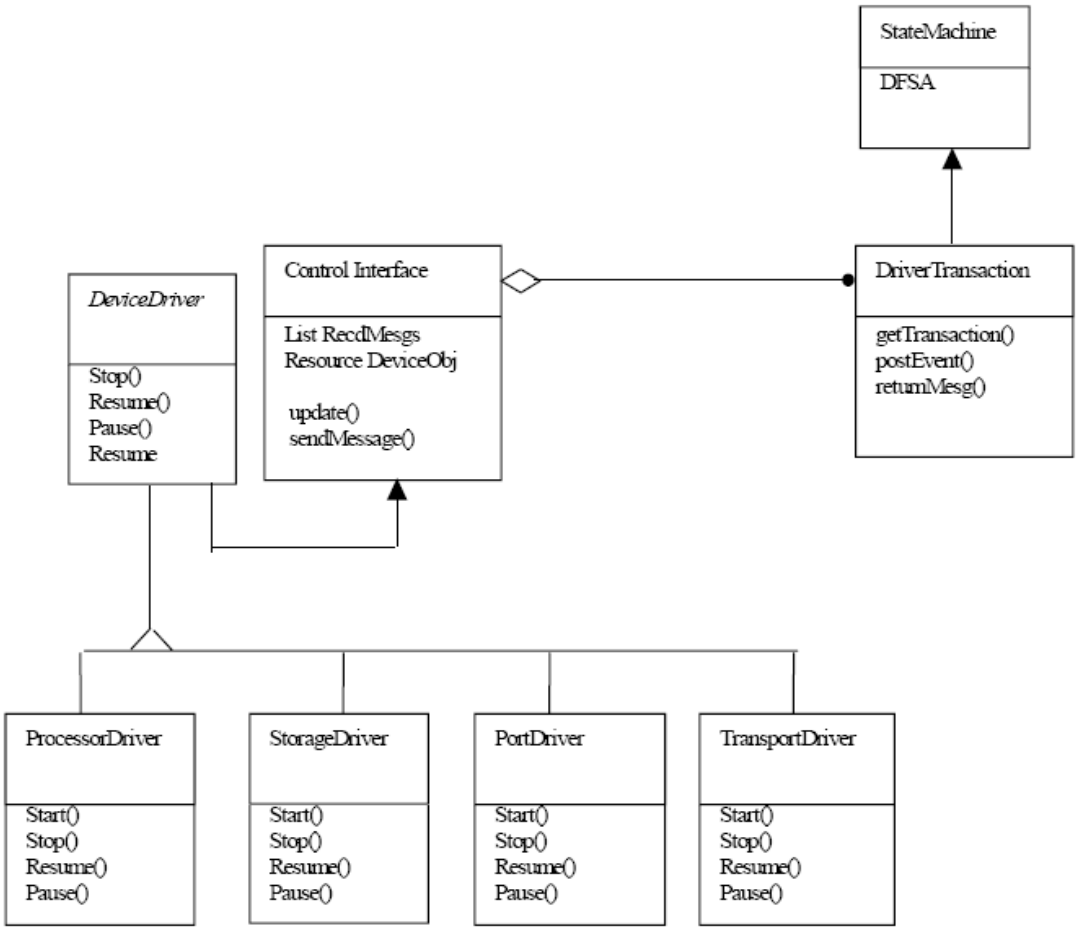

Figure 3: The OMT diagram for the device driver classes (Adapter pattern)

Note the use of a Finite State Automata class in Figure 4. All transactions and messages in the work cell as well as resources are derived from a state Machine. This implies that all the resources have a strict notion of their current states and all the events are state driven. The use of a formal FSA object helps in reducing the bookkeeping that have to be otherwise kept at the server side.

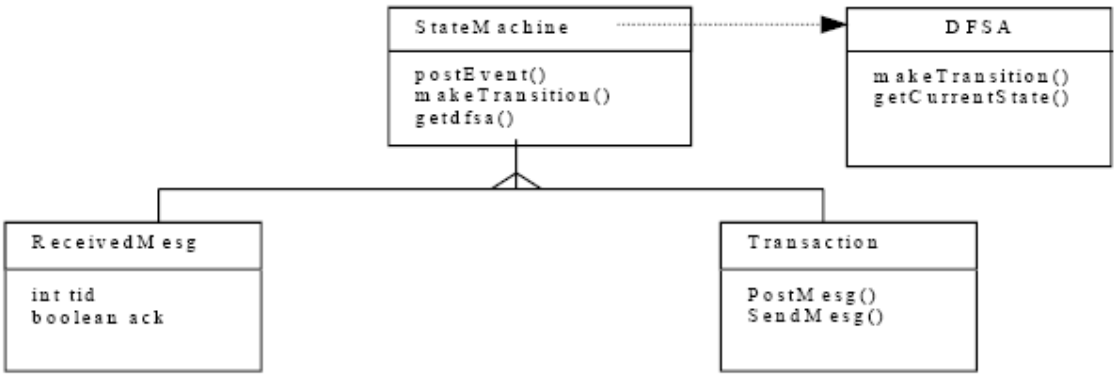

Figure 4: The OMT diagram for the transaction class and the received message used in the work cell and the resource server class

3. MODELING WORK CELLS AND MANUFACTURING SYSTEMS

The steps involved in modeling work cells and manufacturing systems, using this framework, are described in [4], [5].

• The configuration files for each of the basic resource has to be written. These files define the capacity of the resource, the capabilities of the machine (programs, configurations) and the group to which it belongs amongst other things.

• The configuration file for the work cell has to be written. This file defines all the resources that a work cell is composed of. In addition, this configuration file contains defines the connectivity information and the various routes followed by the different job types.

• To attach machine simulators to each of the basic resources, device driver files have to be written that translates commands from the module controllers to the simulators and vice versa.

The fields that are mandatory fields in a configuration file are the following:

• The name used to locate the resource in the distributed environment.

• The type of the resource

• The serverkind field that serves as a de-multiplex key.

• The (buffer) capacity of the resource.

• The port numbers in a resource

• The total number of setups supported by the resource.

• The programs supported in each setup. A program Id (PID) and a filename describe a program.

The additional fields that have to be described in a configuration file for a composite member are described below:

• The members (resources) that the work cell is composed of.

• The capacity of each of the resource.

• The in-ports and out-ports of each of its resource.
• The connectivity information that describes how the ports of the resources are connected to each other.
• There are two keywords used in defining the connectivity information. 'TO' is used to describe one-way connectivity between ports while 'ONTO' means that the connectivity between the ports is two-ways.
• The different job types (routes) supported in each configuration. A route is described by a sequence of stages each stage defined by the resource that the job needs at that stage.
The device drivers for a storage type and a processor type have been described above.
The steps involved in defining a device driver are given below:
• Define the ports of the device.
• The link of the mechanism associated with the port.
• The location of the port with respect to the link co-ordinates.
• Define the programs associated with accept, remove, prepareAccept and prepareRemove commands at various ports.
• Define the mapping between programs IDs (PID) and the programs that would be run by the devices.
However, the port maps have to be defined differently if the capacity of the device is greater than one. The program associated with the accept, remove, prepareAccept and prepareRemove commands are indexed both by the port and the buffer ID.

4. TEST CASES MODELED USING THIS ARCHITECTURE

Case: A simple work cell composed of base resources has been tested successfully using our architecture. The work cell consists of a three-axis machine tool (a unit processor), a pallet changer and an input/output buffer (See Figure 5). The pallet changer is fed jobs/parts from the work cell buffer and it feeds the jobs into the machine tool. The pallet changer picks up a processed job from the machine tool and transfers the job into the output buffer of the work cell.

In the above setup, the capacity of the work cell is four while the processor is of unit capacity. In such a configuration, the interactions between the pallet changer and the processor are dynamic in the sense that they are dependent on the processing time spent by each job on the machine tool and the time at which different jobs enter the system. The controller of the work cell automatically allows/disallows different transitions thereby avoiding conflicts.

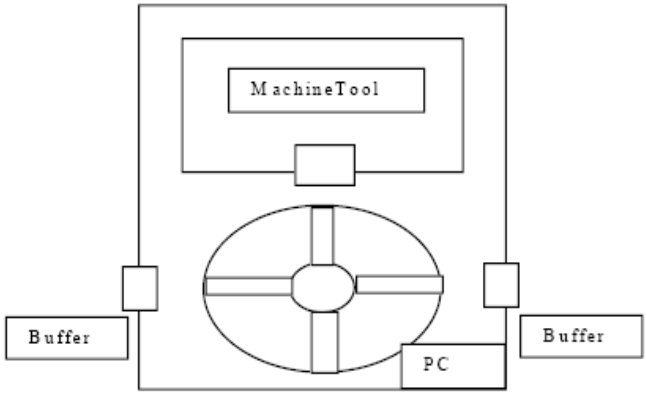

Figure 5: Schematic sketch of a work cell consisting of a pallet changer and a machine tool.

5. CONCLUSIONS

The use of automatic synthesis of supervisory controllers allows a high degree of flexibility in the system. Whenever there has been a significant change in the system configuration (when new job routes have been defined or when resources have been added/removed), the control-laws are recalculated and re-synthesized. We have suggested hierarchical synthesis as a strategy for rapidly configuring large systems. In this paper, a methodology for formally modeling hierarchical resource allocation systems is developed. A distributed hierarchical control policy for ensuring deadlock free behavior in such a system has been proposed. In paper, we apply this methodology to model a FMS setup under the framework of our architecture.

The software module we have implemented based on this architecture is highly configurable to suit the needs of a variety of manufacturing environments. A CORBA based framework has been used to develop the various object modules. This gives us the added benefit of being able to run the application across multi-platforms (operating systems).

Furthermore, the use of distributed object technology to implement the system enables us to run each resource module as a distributed object/server on a computer node. It is possible to access the control panels associated with each resource from a separate computer and this allows the operator to access the system at different control levels (the resource or the work cell).

References:

[1] A d l e m o A., and S.-A., *Models for Specification and Control of Flexible Manufacturing Systems*, Technical Report, School of Electrical and Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1997.

[2] L a w l e y M., *Structural Analysis and Control of Flexible Manufacturing System*, PhD Thesis, University of Illinois at Urbana-Champaign, 1995.

[3] Popp, I., *Consideration regarding model Architecture for Scalable Flexibility in Manufacturing*, International Conference on Manufacturing System, Buletinul Inst. Politehnic din Iaşi, publicat de Universitatea Tehnică “Gh. Asachi”, Tomul LI, secția Construcții de maşini, Iaşi, 2005.

[4] Popp, I., *Consideration regarding a Resource Models and Job Models for Scalable Flexibility in Manufacturing*, Acta Universitatis Cibiniensis, Buletin ştiințific al Univ. “Lucian Blaga” din Sibiu, seria Tehnică, vol. LII, ISSN 1583-7149, p. 77-80, Sibiu, 2005.

[5] Popp I. - *Consideration regarding a Workcell and Resource Model implementation in FMS*, Annals of the Oradea University, Fascicle of Management and Technological Engineering, CD-ROM Edition, Oradea, 2006

[6] R e v e l i o t i s S., *Structural Analysis and Control of Flexible Manufacturing Systems with a Performance Perspective*, PhD Thesis, Univ. of Illinois at Urbana-Champaign, 1996.

[7] W y n s, J., B r u s s e l, H., V a l c k e n a e r s, L., *WorkStation Architecture in Holonic Manufacturing Systems*, Cirp Journal on Manufacturing Systems, Vol. 26, 220-231, (1996).