Consideration the Work cell Conceptual Model for increasing the Flexibility of Manufacturing

I.O. Popp

Abstract**—** In this paper it is described a generic system architecture for increasing the flexibility of manufacturing systems; there is also briefly explained how to configure the different resource modules that actually control the tasks of the physical device and put together a work cell consisting of other FMS resources. Using the principles of distributed object technology there could be implemented each resource in the work cell as a distributed object. A resource, modelled as a distributed object, has a well-defined interface, describing the data and the methods that it supports. A distributed object can execute either on the same computer or another networked computer. Finally, an example on building a FMS using a hierarchical approach is presented.

Keywords**—**Work cell, Modeling, Conceptual Model, Reference Architecture, Supervisory Control.

I. Introduction

AT the enterprise level, manufacturing organizations are faced with accelerating technological cycles, global competition and an increasingly mobile work force. These accelerating technological cycles translate into short product life cycles and increasing product complexity. This results in a product portfolio that is difficult to integrate from a vertical perspective. To this purpose, corporations reorganize to work efficiently to produce a diverse portfolio of products rather than large quantities of a limited product portfolio.

This research and development effort will focused on the concept of developing a generic reference architecture model for the specification, development, control and reconfiguration of a manufacturing enterprise at a work cell level. It introduces the concept of scalable flexibility in manufacturing from the shop floor to enterprise level.

II. Problem Formulation

A. Model architecture

The actual concept of a work cell will be considered as the fundamental building block for the hierarchical synthesis of large and complex systems at all levels of a manufacturing enterprise; from the simple device to the fully automated factory. The techniques for the automatic synthesis of the control policies for the work cell and the hierarchical manufacturing system will be developed. Furthermore, software modules that allows for the implementation of the work cell architecture in a distributed computing environment will also be developed and tested through real industry test cases.

The problem of supervisory control/synchronization in a flexible-manufacturing environment is one of the most difficult problems designers' faces in the conceptualizing of a Flexible Manufacturing System. It is clear that manufacturing flexibility induces complexity.

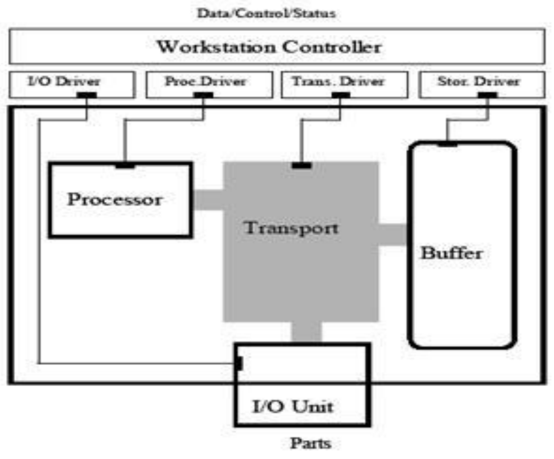

Fig.1 Conceptual Model of a work cell

The conceptual model of such a work cell is shown in Fig. 1. This work cell contains a single processor, a buffer for holding waiting jobs, I/O ports that act as entry/exit points for jobs entering and leaving the work cell and a transportation unit that may be an AGV or a robot.

There is a module that controls each physical device in the work cell. The work cell controller synchronizes the activities of the work cell by sending messages to the resource modules. Designed to support hierarchical synthesis, implies that a number of work cell modules can be used to synthesize a multi-work cell network which, in turn, would have its I/O, inter work cell transportation systems and buffers. It would operate autonomously, in that it would co-ordinate the work flow between each work cell, accept new jobs through its I/O and output completed jobs. Again, it would do so by means of its own supervisory controller. The client-server architecture is employed where all resources (at the lowest level) are synchronized with their activities by sending messages to the cell controller. At higher levels, the work cell would send messages to its own supervisory controller to coordinate the functioning of each work cell. The different tasks that must be addressed when describing the specifications for reference architecture of a work cell are described as follows:

- Specify resource models.
- Specify job models.

B. Description of the System Architecture

1. Architecture

System architecture refers to the architecture of a system or a specific construction. It is the result of a design process for a specific system and lists out the functions of components, their interfaces, and constraints. This design is the basis for detailed design and implementation steps. Defining architecture serves multiple purposes, as given in [1]:

- It provides abstract models for the description of complex systems.
- It reduces the amount of changes to as few modules as possible during a re-design process.
- The architecture indicates the most vital components and constructs that should not be violated when adapting the system to new uses.

2. Modeling of the Work cell

To generically model the work cell we use what is known as a Resource Allocation System (RAS) [3, 5]. A resource allocation system consists of a set of concurrently executing processes, which, at certain phases of their execution require the exclusive use of a number of system resources to run to completion. The resources are finite in number and they are characterized as reusable, since their allocation and reallocation affect neither their quantity nor quality

Furthermore, in the context of FMS, they can be classified as sequential, since it is assumed that every process undergoes a predefined sequence of resource allocation and deallocation steps. The resources it requests at each stage in its route can characterize every process/job in the system. Depending on the nature of the resources for system resources posted by processes at each stage in their routes, one has RASs of varying complexity.

3. Flexible manufacturing system resources

The resource model specification consists of the following:

- *Device Behavior and Interface Model*: Specifying the commands the device is capable of responding to and the response that is expected from the device. Both normal and exception/error responses must be specified.
- *Device Capability Model*: This encodes the abilities of the device. A capability model could be as simple as specifying the set of device programs currently resident on the device (implicit characterization) to a complete model of the device, its internal structure and characteristics (an explicit characterization).
- *Work Cell Behavior (Supervisory Control) and Interface Model*: This model specifies how the devices in the work cell interact with each other to accept process and output jobs.
- *Work Cell Capability Model*: This is analogous to the discussion on the device capability model.
- *Coordination Model***:** For coordinating purposes, a finite state model of the resource is used. This model captures whether the resource is available or claimed or the next resource is claimed. A coordination model similar to the one suggested in [2, 4] is used.

For a single-processor work cell, we shall keep the job model relatively simple. A route characterizes a job. This route is a strict sequence of devices visited by the job in the work cell starting with the device that performs the work cell's input and ending with the device that performs its output. The sequence must include at least one visit to the device that serves as its processor.

C. Resource Models and Job Models

1. Description

A single processor work cell can be thought of as a unit consisting of transportation units, storage units and a single processing unit each functioning autonomously over a predefined set of instructions. The work cell is responsible for physical interfacing with devices that bring the jobs to it, internal handling, transportation and the temporary storage of the jobs during their processing in the cell. The cell controller coordinates the functioning of these units to process the different jobs sent to the work cell. The most basic building block of the system is a programmable machine (or device). This is standardized by an assumption of interface (MMS) and structure.

Resource models are used to describe the resources of a work cell. We use three different views to describe the resource models. Firstly, the resource models that describe the behavior of the device, that is, the interaction between the resources and the work cell controller are needed to support the implementation of the controller module. Secondly we need models that specify the functional abilities of the work cell. Thirdly, we need to model the coordination of the resources. The physical model of a resource in this context would be a programmable machine tool or some other FMS resource. In this framework, a device simulator that supports a standard interface has been used instead. A conceptual model of the device is shown in Fig. 2.

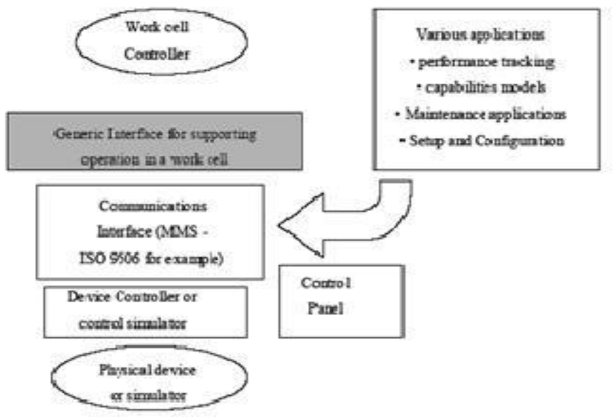
Work cell Controller
Various applications
• performance tracking
• capabilities models
• Maintenance applications
• Setup and Configuration
Generic Interface for supporting operation in a work cell
Communications Interface (MMS - ISO 9506 for example)
Device Controller or control simulator
Control Panel
Physical device or simulator

Fig. 2 A conceptual model of a device

The controller architecture proposed in this system use a client - server architecture where all the devices/resources in the system synchronize their activities by sending messages to the central controller. Each device in the system has a corresponding module in the cell-control system, called the virtual manufacturing device (Fig. 3). The VMD handles the resource-specific control tasks.

2. Behavior Model

The virtual manufacturing device (VMD) can be split into two parts: the generic part provides a uniform interface to the cell controller and enables it to be reused from previous applications. The generic part keeps track of the current state of the external resource and is used to send and receive messages from the cell controller; the specific part of the module is actually a wrapper around the device that translates the high-level messages handled by the generic part to the proprietary protocol handled by the physical device

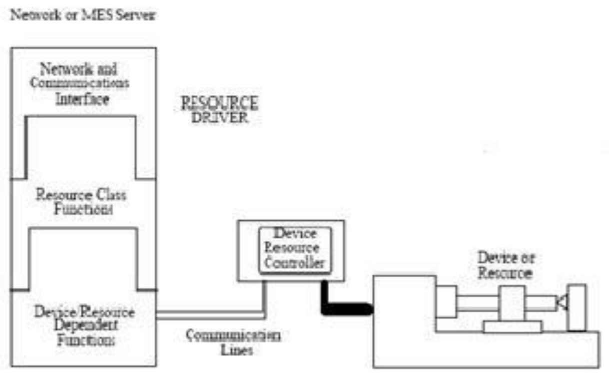
Network or MES Server
Network and Communications Interface
RESOURCE DRIVER
Resource Class Functions
Device/Resource Dependent Functions
Device Resource Controller
Device or Resource
Communication Lines

Fig. 3 Mapping from a physical device to a virtual manufacturing device.

This part of the module can also be thought of as a device driver. The device driver also provides some of the functions that are not provided by the physical device but promised by the generic interface. For example, the device driver incorporates the notion of input and output ports that act as transfer points for the parts flowing into the system.

D. Work cell and Resource Model Implementation

Common Object Request Broker Architecture (CORBA)

An object-oriented approach is used to model the various resources of the FMS. Using multiple inheritance and function overloading, a uniform operational logic can be adopted for different cell configurations. Each resource would, however, also support a device specific interface that might be applicable to other application objects in the system but play no role in the operation of the work cell. The program consists of object models for the transportation units, the storage units, the processor unit and the I/O ports. It also consists of a control module that is based on the supervisory control theory developed in [1, 2, 4].

From the implementation point of view, it is required the communication software that would handle message passing between the different resources of a work cell. Using the principles of distributed object technology, could implement each resource in the work cell as a distributed object. A resource, modeled as a distributed object, has a well-defined interface, describing the data and the methods that it supports. A distributed object can execute either on the same computer or another networked computer. CORBA is an accepted middle-ware standard for building distributed applications. It provides a robust, heterogeneous, inter-operable, multi-platform environment and hence we propose to use CORBA based architecture for the implementation of out architecture.

CORBA is an industry middle-ware standard for building distributed, heterogeneous, object-oriented applications. It details the interfaces and characteristics of the object request broker (ORB) of the Object Management Architecture (OMA). The ORB is mainly responsible for facilitating communication between clients and objects. The ORB delivers requests to objects and returns any responses to the clients making the requests. The client does not know where the target object resides, how the target object is implemented, whether the object is currently activated, and the communications mechanisms used. It allows for client-server and peer-to-peer communication as well. Any distributed object, under the CORBA standard, has to support a list of methods called an interface. The interface has to be defined in OMG Interface Definition Language (OMG IDL) and these definitions would be mapped to a higher level programming language like C++ or Java. This enables us to build clients and servers using different programming languages.

Program structure

Using a CORBA based framework for the implementation of the single processor work cell. Resource is an abstract base class from which all the other objects have been derived. An object relationship diagram clarifies the relationship between the different objects used in the architecture. It makes it clear, for e.g., that all the resources contain a port in their class definition. Each resource module, in essence, behaves like an object server and responds to queries from other application objects. The focus, when developing the resource modules, has been on identifying a set of interface functions that it should support. Recall that the FMS resources have been classified under one of the four categories listed below:

1. Transportation unit
2. Storage Unit
3. I/O Port
4. Processor Unit

The interface of all the different resource modules is defined using OMG IDL. The work cell class is also derived from the resource class and it presents a similar control interface for upward compatibility. The principal difference is that the work cell contains a supervisory control module. Figures 4 and 5 shows the class hierarchy and the object relationship diagrams for the work cell architecture.

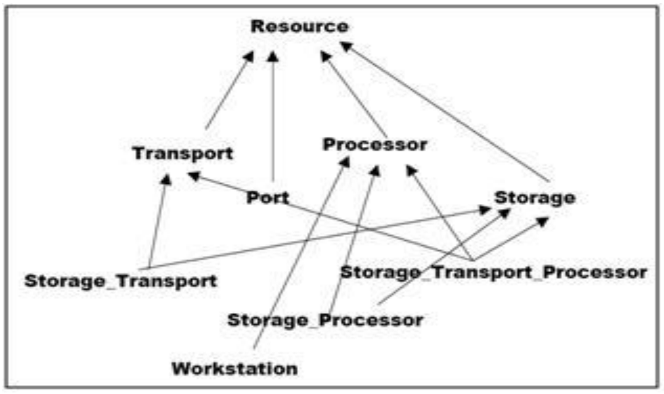

Fig. 4 Class hierarchy diagram for the work cell/station architecture

The control logic used by the module to ensure deadlock free behavior is based on the supervisory control theory developed in [6] and [7]. This makes it easier to build a manufacturing system using a hierarchical approach.

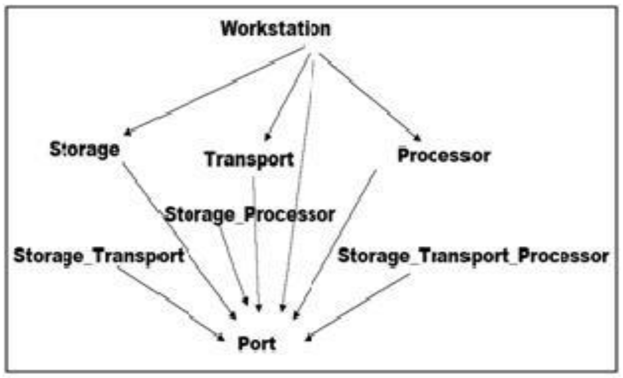

Fig. 5 Object Relationship Diagram for the work cell architecture

The software modules that are used to control the physical devices in an FMS are called the resource modules or the virtual manufacturing devices (VMD). These are divided into the generic and the equipment specific part, also called the device driver.

The controller module knows the status of every resource under it so that correctness of operation is ensured. The work cell module presents an interface similar to the resource module for upward compatibility but in addition it has a dispatcher that schedules the flow of events in it and a controller module that supervises the set of allowable events.

Message Passing: The work cell module has a standardized sequence for sending a handshake message. The handshake message between any two resources is handled through the controller. The sequence of messages that are exchanged during the unloading of a job from a processor (Pri), at port P1, by a transportation device (Tri) is listed below. The transportation device loads the job into port P1 as well:

- Pri->prepareRemoveJob(tid,P1);
- Tri->prepareAcceptJob(tid,P1);
- Pri->send_Message(message1);
- Tri->send_Message(message1);
- Pri->removeJob(tid,P1);
- Pri->sendMessage(message2);
- Tri->acceptJob(Jobid,P1);
- Tri->sendMessage(tid,P1);

This sequence of handshake messages completes the unloading of a job from a processor Pri to a transportation unit Tri. Note that the entire message passing scheme is asynchronous and so we have a provision for the resources to signal the controller when it has completed a specified operation. The equipment specific part or the device driver of a resource contains the code for controlling the physical unit. The different codes to be run are read in by the device driver from a configuration file.

The resource modules for the different class of devices have some fundamental differences. For e.g., the generic resource module for a storage unit has an internal data structure to keep track of the free buffer locations and the length of time occupied by the jobs in each buffer space. When a new job is loaded into the storage unit, the module assigns a free buffer location to the incoming job, makes the appropriate call to the device driver of the storage unit and updates it own data structure. A module for a processor needs to support a function to run a NC program. It sends back a message to the controller on completing the processing of a job. This stage is absent in other resource modules. Thus, the dispatcher in the work cell makes sure that this stage is called on all processors but is bypassed on other resources.

Further, the controller also ensures that conflicting operations on a resource are never issued simultaneously. Let us consider a case where a resource is prepared to accept a job at one of its input ports (say) P1, and then the resource cannot prepare to accept another job even if it has additional free capacity until the first job has been accepted by the resource. Thus, atomicity of a certain set of operations is ensured at the resource level.

Resource Configuration*:*

As per our architecture:

- Every resource in the work cell consists of a port(s).
- Jobs enter/exit a resource only through ports.
- Every port in the device is defined as interfacing with another device in the work cell.

Thus, these ports are the material transfer points. Since every physical piece of equipment in the system need not provide this functionality, the device driver (or the specific part of the VMD) holds the information about these ports and provides the cell controller with this functionality. The connectivity information is defined in the configuration file of the work cell controller. Whenever a job route is defined and uploaded to the system, it is verified to make sure that it is compatible with the connectivity information.

The device driver of each resource is configured so that it would know the actual programs (NC programs) it needs to execute on the physical device on receiving an instruction from the cell controller. The device driver would have a mapping function that would translate a call (say) prepare_to_acceptJob at port P1 into a corresponding NC program. These translation maps would be read in from corresponding configuration files of the resource modules.

A sample configuration is given in Fig. 6. As shown in the figure, every device in the system has a set of ports associated with it. The work cell has a list of input ports and output ports associated with each device. In the above example, P1 is both an input port and output port for MT1. Similarly, P1 is an output port and P3 is an input port for PC1. P_2 is both an input and output port for PC1.

The various programs that can be supported by the resource are specified in the configuration file. It is also possible to dynamically add programs to the resource through the control interface of the module.

The work cell can be setup to support different manufacturing environments. A work cell can be configured to have multiple buffers, separate I/O ports, combined storage/transportation and so on. The ports associated with each device and the connectivity information is defined in the work cell configuration file. The job routes supported at the work cell level are also specified in the configuration file. The operational logic of the work cell controller in such architecture is relatively independent of its configuration. The appropriate procedure calls (handshake messages) are made on the devices sequentially as given in the job definition.

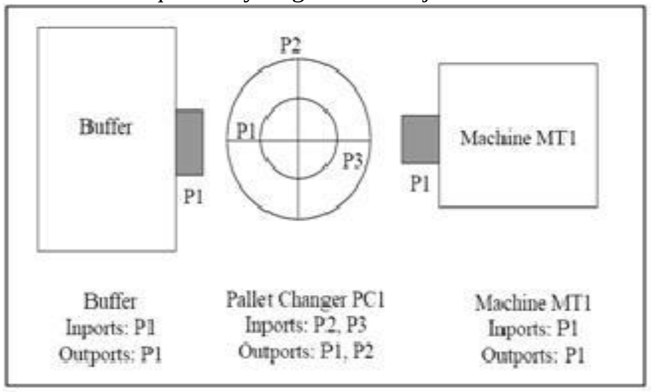

Fig. 6 Resource configuration in a work cell with two devices

The figure 7 shown below depicts the interaction of the resource modules, the device simulators and the work cell controller. The display server shown in the figure is a visualization server implemented using Java 3D. The resource modules (VMDs) that control the device simulators feed the display server with commanded positions of the links of the various mechanisms. Omnibroker is a CORBA compliant ORB that is used for communication between the various distributed objects.

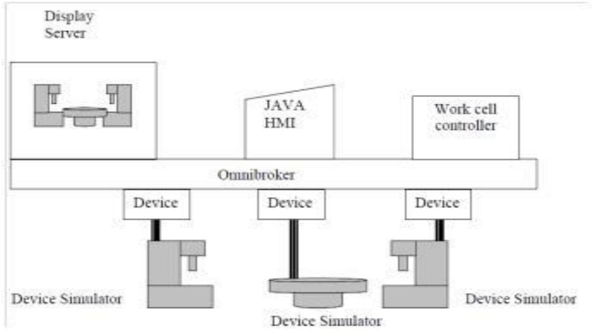

Fig. 7 Interaction of the device simulators, resource modules and cell controller in a distributed environment

A screen shot of the display server is shown in Figure 8.

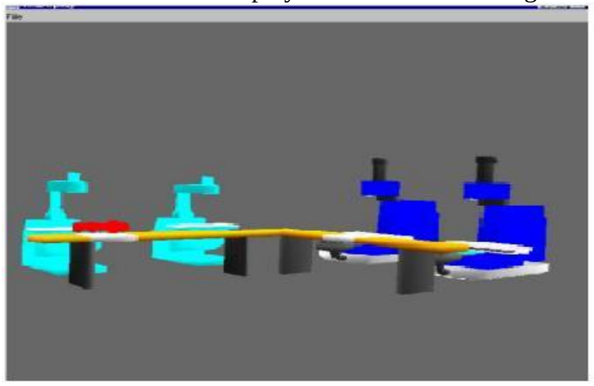

Fig. 8 Screen shot of the display Server (in Java3D)

E. A Software Framework for Work Cell Architecture

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.

The following section is focalised on the basic building blocks of the framework. 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 9 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 section.

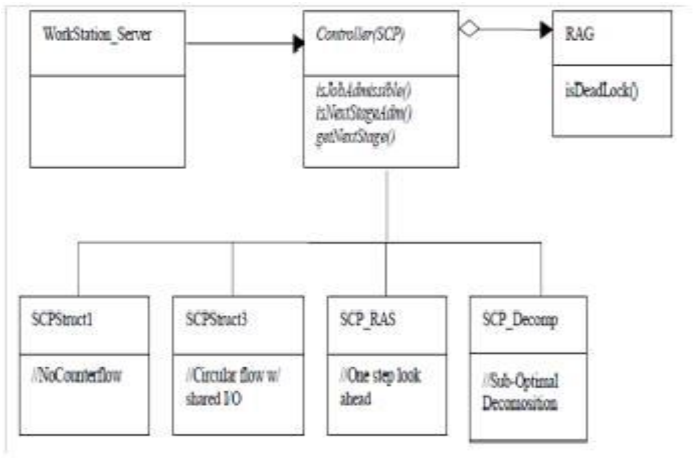

Fig. 9 Plug-in supervisory controller used in the work cell (strategy pattern)

Figure 10 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:

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.

4. SCP4(SCP_Decompose)- Default case. Handles the most general case by route decomposition.

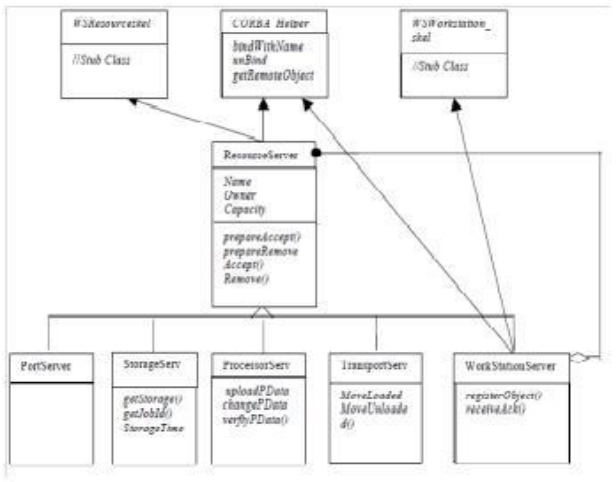

Fig. 10 Work cell – Resource Part Hierarchy (Composite Pattern)

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 (Fig. 11) 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.

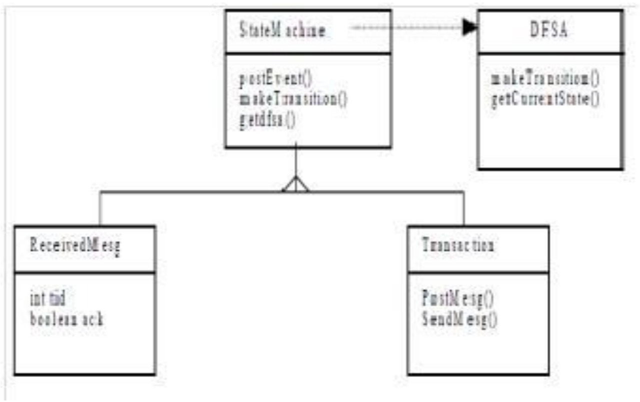

Fig. 11 The OMT diagram for the transaction class and the received message used in the work cell and the resource server class.

Note: the use of a Finite State Automata class in Fig. 11. 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.

III. Problem Solution

A. Modeling work cells and manufacturing systems using this framework.

The steps involved in modeling work cells and manufacturing systems are described in this section.

• 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. (This information is duplicated so that the work cell need not make an additional request to the resource during setup.)
  • 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 portmaps 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.

*B Test Cases modeled using this architecture***.**

Case 1: 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 (Fig. 12). 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.

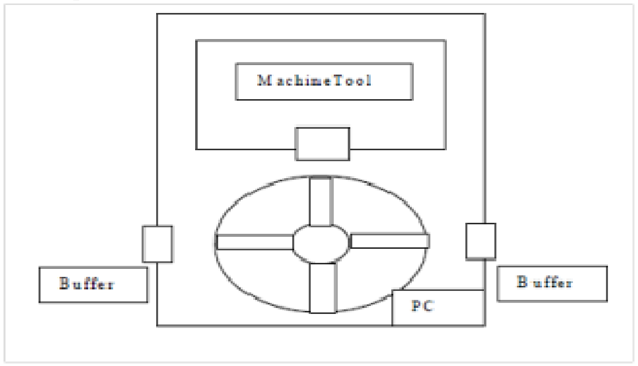

Fig. 12 Schematic sketch of a work cell consisting of a pallet changer and a machine tool.

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 13 Screen Shot of a unit processor work cell

Note that the configuration files for basic resources are very simple while the configuration file for the work cell is more involved. This is not specific to this test case but is a more general situation.

Case 2: A work cell consisting of two Universal high speed placement machines (HSP) in serial and a conveyor that shuttles jobs between the two were modeled using this architecture (Fig. 14). The work cell supports two different job types. The two job types are defined in terms of the devices that they visit.

- Job1 - < HSP1 Conveyor HSP2>
- Job2 - < HSP2 Conveyor HSP1>

Each of the HSPs is modeled as a unit capacity processor while the conveyor is modeled as a unit capacity transportation unit. Since there are counter flow jobs in the system and the conveyor is a shared resource of unit capacity, there is a potential for deadlocks. The work cell controller makes sure that such situations don't arise.

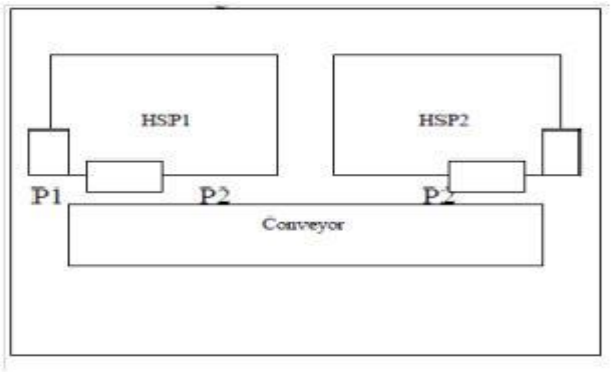

Fig. 14 Schematic Sketch of the HSP cell

Each of the HSPs is modeled as a unit capacity processor while the conveyor is modeled as a unit capacity transportation unit. Since there are counter flow jobs in the system and the conveyor is a shared resource of unit capacity, there is a potential for deadlocks. The work cell controller makes sure that such situations don't arise.

Some sample configuration files for the resources and the work cell are given below. The device driver files for the HSP and the conveyors are also listed below. Note the similarity between the configuration file of the Universal machine and the 3-axis machine tool in the previous example.

This is because the control information for both the machines (unit capacity processors) is the same. The device driver files for each of these machines is obviously different and is specific to the simulator that is used to model the machine. The work cell configuration file for this system is written down exactly the same way as explained at the start of this section. Notice that the geometry at each stage has been defined as 'user defined' in the route definition.

This is a keyword in the language that means that the geometry of the part at each stage would be redefined by the operation that takes place at that stage. That is, the part geometry has to be shared between stages.

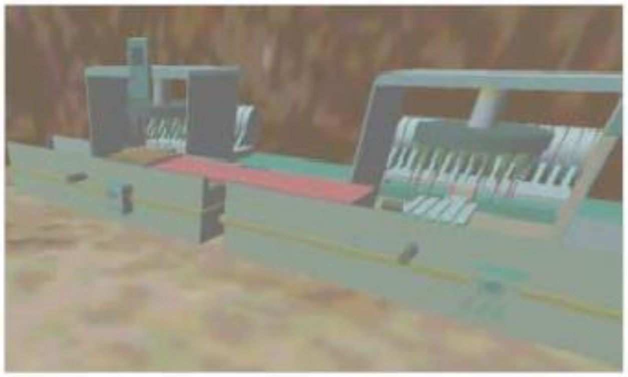
Fig. 15 Screen Shot of the HSP cell

Case 3: In this case is configured a system with a real-time simulator of a machine axis. The purpose of this case was to assess the ability of this system to co-exist with emerging standards for devices (Open Modular Architecture Control or OMAC, in this case) and exploit the fact that the underlying descriptions of both systems were finite state machines.

The test case is consisted and supervisor running on a Linux operating system and communicating with an OMAC axis simulator running on a QNX system.

The scenario consisted of two such single axes devices, with a fictitious job being sent to them. He wanted to check if you could access the axes states, send them commands and co-ordinate their actions according to the protocol defined. Although conceptual interface easily found, a significant effort was spent for the following reasons:

a. Lacking a real-time CORBA environment, had to write a socket connection between the device driver running on the Linux system and the axes running on QNX. The important point here that we learnt was that, when selecting software enabling environments, one must make sure that they support the different operating systems one might encounter (1 man-week)

b. Was attempted to drive a Java3D display server over the network. The overall performance was poor. Could only achieve about rates of about 10 samples per second, when the all the components, the real time simulator, the supervisor and the display server, ran locally. Was expect much worse performance across the internet.

c. Was spent most of time understanding the real-time software and modifying it from being a controller to a simulator (2 man-weeks)

d. Configuring the system when had the above (i.e., writing up the configuration files, etc.) took about an hour.

IV. Final Remarks

This research and development effort proposes such a reference architecture that will allow for the control and reconfiguration of a flexible manufacturing work cell. This architecture will address many different issues, such as, the type of resources in the system, system capabilities, system behavior, and architecture interfaces. It also develops the rules for synthesizing complex manufacturing systems.

Defining such reference architecture will serve multiple purposes as follows:
• It provides an abstract model for the description of complex systems.
• It reduces the amount of modifications to a limited number of modules during a re-design of processes to support the varying product portfolio.
• It identifies the most vital components within the reference architecture system.
• It identifies the constraints that should not be violated when adapting the system to new uses.

The reference architecture will describe the types of system components, their functionality's, dependencies, possible interactions and constraints. In addition, the reference architecture also incorporates some of the rules concerning system development in a specific domain to achieve the following:
• A unified terminology.
• Design simplicity allowing faster and cheaper design of the system architecture.
• Higher quality systems.
• Interfacing and re-usability of modules between different applications.
• Partitioning of implementation tasks amongst different development groups.
• Tractability between solution independent requirements and final realization.

The structural approach that was adopted is that of a hierarchy. A "Standard", configurable, self-contained, autonomous module serves as building block for hierarchical synthesis of larger and more complex systems. At each level of the hierarchy the sub-system is persevered as an autonomous and self-contained unit. At the lowest level, a single processor work cell is identified as follows:
• It has a single processing unit or element
• It is self-contained; with a processing element, a buffer element, an internal transportation element and an input/output (I/O) element for physical interfacing with the outside world.
• It is autonomous as it contains its own internal co-ordination or supervisory controller which completes all logical and informational interfacing with the external world
• It is configurable as the internal buffer, transport, I/O and processing can be effected by a single device or can be effected by a mix of devices. Further, when comprised of a number of devices, the supported workflows can be configured by appropriate specifications and will be effected by the cell's supervisory controller.
• The single-processor work cell presents a standard interface, for processing jobs, to the external world (in a hierarchy, the level above). Also, it communicates to its components (in a hierarchy, the level below, which in this case, would be physical devices) through some standard interfaces.
• The single-processor work cell implements a standard protocol for physical material transfer between its components and between its I/O and the external world.

V. CONCLUSION

In this paper, are summarize the results presented in the research, highlight the significant features of the work and present some guidelines for future research and extensions of the present work.

It is described a generic system architecture for flexible manufacturing systems. Furthermore, the product specification has been integrated with the control system. The approach adopted can be summarized as follows:

- The use of object-oriented approach to model the various FMS resources as resource modules or VMDs. The classification of the various resources into four principal types based on their behavior is given below. These serve as the basic object types for the resource models in the architecture.

• Transport Unit; Storage Unit; Processor Unit; I/O Port Unit

- Specification of job routes as a sequence of job stages.

- Automatic synthesis of a supervisory controller given a set of resources and the product routes.

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.

It is suggested hierarchical synthesis as a strategy for rapidly configuring large systems. 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. It is applied this methodology to model a FMS setup under the framework of the architecture described.

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).

The software module which was 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 gave the added benefit of being able to run the application across multi-platforms (operating systems).

Future Work

In the hierarchical resource allocation systems it was assumed that the base resources (physical equipment) were part of only one HRAS. It would be interesting to analyze the behavior of the system if the resources were to be shared between sibling HRASs.

Security features have not yet been incorporated into the system. Also, error handling has not addressed in this architecture. Problems such as restarting after a breakdown certainly need to be addressed. The structural control policies presented in this work implicitly assumed the absence of uncontrollable events, such as machine failures, and hence further research has to be done towards the development of deadlock-free control policies in the presence of such uncontrollable events.

Currently, the software implementation of this architecture has been restricted only to the resource and the work cell level. The higher-level modules and the distributed, hierarchical controller have not been implemented yet. Future work includes the development of the higher-level modules and the distributed controller that ensures deadlock free behavior of the entire system. This would help in realizing a scalable, 'unifying' operating system for manufacturing systems.

REFERENCES

[1] A. Adlemo., M.Fabian and P.Gullander, *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] M. Lawley., Structural Analysis and Control of Flexible Manufacturing Systems, PhD Thesis, University of Illinois at Urbana-Champaign, 1995.

[3] K. Karthik, An Architecture for control and specification of flexible manufacturing systems, MS Thesis, University of Illinois at Urbana-Champaign, 1998.

[4] S. Reveliotis, Structural Analysis and Control of Flexible Manufacturing Systems with a Performance Perspective, PhD Thesis, University of Illinois at Urbana-Champaign, 1996.

[5] I.O. Popp, I. Barsan, Consideration regarding the design of the system architecture for FMS, *3rd International Conference OPTIROB*, Bucuresti, 2008, pp. 229-233.

[6] I.O. Popp, I. Barsan, Using an Object-oriented Approach for Scalable Flexibility in Manufacturing**,** *Advanced Materials Research*, Trans Tech Publications, Switzerland, Vols. 463-464, 2012 pp. 1035-1038.

[7] R. Mackiewicz, "An overview to the Manufacturing Message Specification", http://litwww.epfl.ch/MMS/mms_IntroSISCO.txt, 1994.

[8] Z.A, Banaszak, B.H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing flows", *IEEE Trans. on Robotics and Automation*, Vol. 6, pp. 724-734, 1990.

[9] J. Brzezinski, J.M. Helary, M. Raynal. and M. Singhal, "Deadlock models and general algorithms for distributed deadlock detection", Technical Report, The Ohio State University, 1994.

[10] H. Cho., T.K. Kumaran. and R.A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems", *IEEE Trans. on Robotics and Automation"*, Vol.11, pp 413-421, 1995.

[11] "MMS: A Communication Language for Manufacturing", Research Reports ESPRIT Project 7096. CCE-CNMA, Springer-Verlag, Vol.2, 1995.

[12] Grady Book, "Object-Oriented Design with Applications", Benjamin/Cummins, Redwood City, California, 1991.

[13] *"Computer Integrated Manufacturing (CIM) Framework Specification Version 2.0"*, SEMATECH. INC, 1998.

[14] "*Virtual Factory Equipment Interface (VFEI) Version 2.2"*, SEMATECH. INC, 1995.

[15] http://www.sleepycat.com/db, http://www.javasoft.com, http://www.ooc.com

I.O. Popp Place and/or date of birth: Sibiu, Romania, 10.03.1961.
Doctor engineer (PhD), „L.Blaga" University of Sibiu, 1999.
Graduate diploma/Engineering studies: Mechanical engineer in the speciality: Machine Building Manufacturing Technology, Technical University of Cluj-Napoca, Mechanical Faculty of Sibiu, 1986.

He has the work experience: Engineer of Technical Department of Production, Mechanical Plant MIRSA Avrig, Sibiu County, 1986-1992.

Assoc. Prof. Dr. Popp: Machines and Industrial Equipments Department, University "*Lucian Blaga*" of Sibiu, Faculty of Engineering, Emil Cioran str. No. 4, 550025, Sibiu, ROMANIA, ilie.popp@ulbsibiu.ro
Membership in professional societies and associations: AGIR (General Associations of Engineers from Romania) 1994 - present.