10

Integration of Object Oriented Programming and Petri Nets for modelling and Supervision of FMS/FAS

J. Barata; L.M. Camarinha-Matos
Universidade Nova de Lisboa - Dep. de Engenharia Electrotécnica
Quinta da Torre, 2825 - Monte Caparica, Portugal
Tel +351-1-3500224 / 2953213 Fax +351-1-2957786
E-mail: {jab,cam}@uninova.pt

W. Colombo; R. Carelli
Instituto de Automática - Universidad Nacional de San Juan
Av. San Martin-oeste 1109, 5400 San Juan, Argentina
Tel +54-64-213303 Fax +54-64-213672
E-mail: rcarelli@inaut.edu.ar

Abstract

An hybrid approach combining frames/object oriented and Petri Nets paradigms for modelling, evaluating and supervising FMS/FAS is presented. Application examples in the context of a pilot FMS/FAS system are discussed.

1. INTRODUCTION

The aim of this paper is to provide a hybrid approach for dealing with a class of control problems occurring in manufacturing flow control. In modern production systems, the introduction of automated flexible manufacturing and assembly systems (FMS/FAS) has increased the need for efficient production planning and control techniques. FMS/FAS can produce multiple types and variants of products using various resources, such as robots, multipurpose machines, etc. While the increased flexibility of a FMS/FAS provides a greater number of choices of resources and routings, it imposes a challenging problem; i.e. the allocation of given resources to different processes required in making each product, the planning of the activities to accomplish the best efficiency and the actual link between a global control program and the real controllers of each manufacturing resource.

Traditionally, the routing of a part to complete a sequence of required processes is considered planning, and the assignement of resources according to the determined routing is considered scheduling. In a FMS/FAS, the planning and the scheduling should be collectively carried out to take advantage of the flexibility of the system. Since a machine in the FMS/FAS can be used for multiple jobs and there are choices of resources to be used, the assignement of resources

L. Camarinha-Matos et al. (eds.), *Balanced Automation Systems*

-scheduling- must be considered when the routing of a part is planned [1].

In this paper we present a hybrid platform to assist in designing, modelling and analysis of FMS/FAS, which uses a Petri Net formulation and frame/object oriented modelling. Petri Nets constitute a mathematical and graphical tool for modelling and analysis of discrete-event systems, which allow to easily represent parallelism and synchronization of activities [2]. Frames/Object oriented programming is introduced to model the structural and operational facets of the manufacturing resources and to establish a link between the model and the local controllers. The proposed platform allows to link object oriented models of FMS/FAS developed by UNINOVA [3-5]with a Petri Net based simulation and evaluation software (SIMRdP) developed by UNSJ [6, 7] The structure is designed to be interactive with the real world, to perform monitoring and control tasks on the actual FMS/FAS system.

2. GENERAL ARCHITECTURE

Figure 1 shows the general structure of the proposed system.

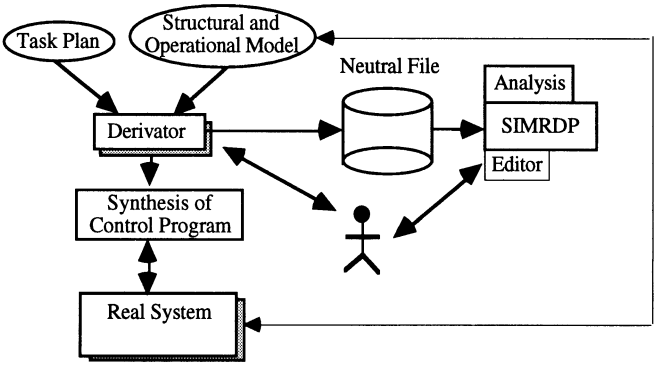

Figure 1 - System Architecture.

The system consists of the following parts:

1. A model for derivation of a Petri Net from a task plan. This plan is a graph, represented in frames, that describes the production tasks at the operational level.
2. A module for creation of structural and operational model of the FMS/FAS, based on a frame/object oriented representation and a subsystem for derivation of Petri Net models of the FMS/FAS system.
3. A module for editing, evaluation (simulation, structural and quantitative analysis) of FMS/FAS system modelled by means of Petri Nets.
4. A module to synthetize a control supervision program from the analyzed Petri Net.
5. A neutral file for linking the above mentioned modules. This data structure is important as the various modules are separetely developed by the two universities.

3. STRUCTURAL AND OPERATIONAL MODELS

In our approach the frame representation paradigm, with some facets of OOP, is adopted to model the structure and operational behaviour of the manufacturing system.

3.1 Structural aspects

The various examples to be discussed in order to introduce the main modelling concepts use a cell as the basic modelling unit. A cell is a composite entity that is capable of making some transformation, movement or storage related to some product or part. In structural terms, each cell has components to support the input of parts, an agent to perform the transforming actions and components to support the output of products/processed parts.

The input, output and agent elements will be supported by manufacturing components. Some components can only support one function but there are others components which can support more than one function. Components adapt themselves to the roles they can perform. Some components are more adaptable than others. For instance, a Conveyor belt is very flexible because it can perform an input, output or agent role, but a CNC machine only can play an agent role.

The generic cell concept can be specialised by activity. There can be cells specialised in assembly, painting, welding, storage, machining, transportation, etc. A shop floor is just a set of specialised cells.

Metaknowledge can be associated to each specialised cell to represent the specificities of its application domain. For each domain, the specific cell has the same structure as the generalised Cell concept (Input Agent, Processing Agent, Output Agent) but the domain and cardinality of the implementing components is different in each specialisation (Figure 2). For example, in a Painting or Welding Cell, a vibrator feeder is not a valid Input item, but this component is valid in an Assembly Cell. The Metaknowledge seems to be a very important element at the configuration phase, assuring the validity of cells.

```
FRAME   CELL
name:
base_coordination_system:
processable_products:
input_parts:
connected from:
processor:
connected to:

FRAME   ASSEMBLY-CELL
is-a: CELL
val-inp-ag: vibratory_feeder,
            buffer,
            gravitic_feeder,
            Index_Table, agv,
            conveyor
val-out-ag: conveyor, agv, buffer,
            index_table
val-proc-ag: robot
```

```
FRAME   ROBOT_COMPONENT
is-a: manufacturing_component
Base_coordinate_system:
Controlled by:
Applications: assembly, gluing, ..
DOF: 6
Working_area:
Load:
Repeatability:
Current_position:
Cost:
Cycle_Time:
Next_maintenance:
N_working_hours:
Weight:
Max_speed_by_axes:
```

Figure 2 - Example concepts of cell, assembly cell and component.

At this stage it is convenient to clarify the concepts of agent, input and output, and their relationships to the components/manufacturing resources.

Components are entities which participates in the productive process with a specific function and can be controlled by a computational entity. Components models are context independent descriptions of their static and dynamic characteristics. A robot component model (Figure 2), for instance, includes all the characteristics which completely characterize its structural and dynamic aspects.

A robot **agent** is a model of a robot and associated resources, like tools or auxiliary sensors, when inserted in a particular context. A robot can play different **roles** (Figure 4) in different **contexts**. The (expected) behaviour of a robot in an Assembly context is different from its behaviour in a spot welding context.

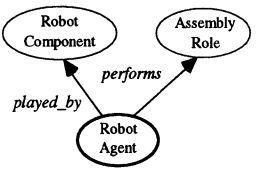

Figure 3 Structure of a robot agent.

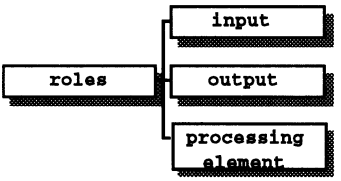

Figure 4 Role taxonomy: main level.

On the other hand, when a robot is performing a given role, it may resort to auxiliary resources, like tools, sensors, buffers, etc., that extend the robot's functionality in order to fulfil the functionality required by this role. A robot agent is, therefore, a model of the robot when playing a particular role and extended by selected attributes inherited from the auxiliary resources.

The entity that effectively participates as an assembly robot, for instance, is one which has those characteristics from the robot component model to perform the assembly role.

The agent entity ASSEMBLY_ROBOT is a structure which is supported by two relations: *performs* and *played_by* (Figure 3). The relation *performs* assures the inheritance of the role characteristics to the structure while *played_by* assures the inheritance of those agent relevant aspects, from the component. *Main_attributes* and *component_attrib* are attributes to be used by *played_by* and *performs* relations.

```
FRAME  ASSEMBLY_ROBOT
is-a: agent
performs: ASSEMBLY_ROLE
played by: ROBOT_COMPONENT
main_attributes:force_sensor,
                current_tool,
                available_tools,
               available_resources
component_attrib:Base_coord_system,
                 Controlled_by,
                 Working_area,load,
                 Current_position
```

```
FRAME  AG_ROBOT_ASSEMBLY_ROLE
is-a: role
tools_domain:(grippers,
              screwdriver)
aux_res_domain: (buffers)
force sensor:
current tool:
available_tools: gr1, gr2, sd2
aux_resource: buf1, buf2
assembly_device: fixture1
```

Figure 5 Example concept of an agent.

The slots, *tools_domain* and *aux_res_domain* represent domain-knowledge that is important during configuration time. The slot *current_tool* is a relation that associates the main player of this role (robot component) to a particular tool. By the "inclusion" restriction, only tool_operations will be inherited by the ag_robot_assembly_role. *Assembly_device* is an

attribute describing where assembly operations are really done. *Fixture1* is an instance of a component specialised in holding parts.

```
RELATION   PERFORMS
type: intransitive
inherit_slot: main_attributes
inverse_relation: performed_by

RELATION   CURRENT_TOOL
type: intransitive
inherit_slot: tool_operations
inverse_relation: attached_to
```

```
RELATION   PLAYED_BY
is-a: relation
type: intransitive
inherit_slot: component_attributes
inverse_relation: play
```

Figure 6 Definition of relations *performs ,played_by* and *current_tool.*

A cell is made of entities that are playing different roles.

This modular approach to cell representation facilitates the creation of complex systems by simple "concatenation" of cells. A particular manufacturing unit is made of several subsystems (Transportation Cells, Painting Cell, Assembly Cell, ...). A manufacturing unit could be modelled by a FMS/FAS entity, which has access to all characteristics and functionalities of the subsystems involved in the Unit.

The frame representation paradigm, specially when it allows the definition of new relations, is quite adequate to model this kind of structures.

3.2 Dynamic or Operational Aspects

Dynamic aspects are related to the components internal state changes. The dynamism presented by components is achieved through controller actions. Every component with dynamics must have a controller associated to it. The way the model reflects physical changes on the component and the way physical component reflects model changes is the most important point when discussing dynamic aspects.

Dynamic aspects can also be discussed with two different views: (1) considering the components as isolated entities or (2) considering complex structures, like cells, made of components. In the first view the key point is how components are actuated, without any concerns about their interrelationships. In the second view, aspects related to synchronisation are the most important ones (this will be analysed in the PETRI NETs chapter). In this point the concern is with the first view.

```
RELATION   CONTROLLED_BY
is-a: relation
type: intransitive
inherits: inclusion (move_wc,
                     move_jc,
                     hardhome,
                     acceleration,
                     speed)
inverse_relation: controls
inverse_relation: controls
```

```
FRAME   ROBOT_CTRL_COMPONENT
is-a: controller
move_wc:method move_wc_fn(x,y,z,q)
move_jc: method
          move_jc_fn(m1,m2,m3,m4)
hardhome: method hardhome_fn
acceleration:demon if write
                accel_dem
speed: demon if write speed_dem
input: byte demon if needed
input_dem
output:byte demon if write
output_dem
```

Figure 7 Model of a component and its controller.

Every component model with dynamic behaviour should have a controller model. This model should be like an image of the real controller. Using a frame oriented paradigm, the controllers functionalities could be defined by methods or demons. In this way most of the controller's model is a list of methods, a method for each functionality.

A component is related to its controller by a *controlled_by* relation while the controller relates to its component by a **controls** relation.One of the most important points in this discussion is the way a controller model is connected to the local controller. This connection is sometimes not easy because it involves the cooperation of two different computational worlds: the computational world where the model runs and the real controller. To make things even more difficult, sometimes, real controllers have closed architectures. From our experience, a lot of effort has usually to be put in trying to open real controllers architecture, and implies the production of an interpreter that runs on it. This interpreter accepts commands from an image that runs in the modelling world.

The methods of a Controller model implement the actions that are needed to send the right commands to the real controller. The real controller image should be developed using a client-server approach. In this way, implementation methods can ask this server to perform the needed actions. These methods hide the underlying hardware structure from the application, i.e., any application using a robot component doesn't need to know anything about the real robot controller and its image or server. The applications only know what functionalities are provided by the robot component model. This approach could be very suitable to integrate existing controllers, making the integration of legacy systems an easier task.

Figure 8 illustrates this approach that was followed in the Uninova's FMS/FAS system (NovaFlex).

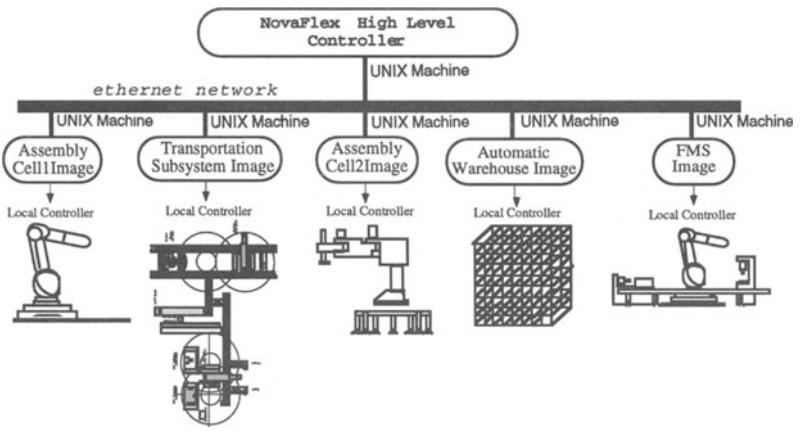

Figure 8 - NovaFlex physical infrastructure.

The Flexible Manufacturing and Assembly System -- NovaFlex -- installed at the Center for Intelligent Robotics (CRI) of UNINOVA was conceived as a demonstration unit able to handle a set of typical activities of a Computer Integrated Manufacturing (CIM) system [4].

Besides the machining and assembly subsystems, the Pilot Unit includes a storage component, an input section for raw materials, a delivery section for finished products and a transportation subsystem that links all the other components. The transportation medium is a

pallet-based conveyor belt. Each pallet can be adapted to transport different kinds of parts and products.

As one of the basic design goals, the system was required not to be restricted to a particular type of product. The objective was to build a relatively generic infrastructure, that could be adaptable to a range of products with minimal setup effort. The requirement was for a flexible infrastructure, with a representative set of manufacturing resources, and not a special purpose system.

Another very important aspect is the possibility of different groups being simultaneously using different subsystems of the Unit for separate experiments. As a matter of fact, this situation is expected to be the most common practice during the system's life time.

These requirements led to an architecture in which NOVAFLEX can be operated either as an integrated FMS/FAS system or as a set of isolated subsystems (machining, assembly, transportation and storage, etc.). This last aspect has particular consequences on the design of the control architecture.

Therefore, the need to support these different research areas implied the design of a flexible architecture, from the topology to the control points of view. An easy reconfiguration of its operating mode is an important requirement to support concurrent research activities.

4. FMS/FAS EVALUATION USING PETRI NETS

In this chapter it is proposed to use Temporized Petri Nets (TPNs), as a discrete event specification tool for modelling the FMS/FAS under study. Petri Nets (PN) have been designed specifically to model FMS/FAS [8-10] They provide suitable models due to the following reasons:

- PNs capture the precedence relations and structural interactions of unpredictable, concurrent, and asynchronous events. In addition, their graphical nature helps to visualize such complex systems.
- Deadlock, conflicts, and buffer sizes can be modelled easily and efficiently.
- PNs models represent a hierarchical modelling tool with a well-developed mathematical and graphical foundation.
- PNs allow the use of different abstraction level models and the use of powerful modelling methodologies: refinement and modular composition, each of them with its corresponding validation analysis.

It is very important to take into account that the main functions of production/assembly systems, be them flexible or not, are to input raw material, to perform a certain number of transformation tasks, assemble and disassemble of initial parts and, finally, to output finished parts. Therefore, the designer has to identify the input and output sequences of material parts, at the same time all the elementary operations which have to be applied on these parts, and the main characteristics of the resources (turning, milling machines, manipulators, AGVs, etc.) involved in each of these operations.

According to the above considerations, the Petri Net based simulator -SIMRdP- has a logical model where:

- One place is associated to one state of the produced part in its operating sequence, or the state of a resource (iddle or busy by a part).

- One transition is associated to only one operation on the produced part, or to the assembly operation of several parts.
- One token is associated to one material part, or information about the resources of the FMS.
- To each place of the net, a capacity place or capacity monitor is associated to indicate the maximum number of tokens this place may have [11].This capacity is introduced to indicate that a buffer, a storage area, or another resource can only contain a limited number of parts at one time.

In addition, all the resources and operations verify the following set of constraints:

- The operations are completed in finite time, and they can be decoupled from the time point of view.
- One part is submitted to only one operation at a time, and each resource can perform only one task at a time.
- One separate part can be submitted only to transformational or informational functions. A transformational function consists in modifying the physical attributes of the part (shape, constitution, surface, etc.). They are the machining functions: turning, milling, manipulation, and conditioning functions: termical treatment, painting, washing, etc.. An informational function consists in verifying that the operations have been accomplished correctly.

4.1 Functional Analysis

After the modelling, the designer must develop a complete study and proof for the correct behaviour of the modelled system specifications. In order to derive the functional (i.e., logical) properties, the structure of the Petri Net model is considered for the analysis, which is called qualitative validation in the literature.

For developing the qualitative validation, it is possible to use some of the methods which are extensively described in the literature of Petri Nets [8, 12]. According to the structural characteristics of the Petri Nets used in this approach, it has been developed a software tool [13],which includes the following Petri Net analysis methods: the Coverability Graph method by means of simulation of the Petri Net token-game [12] and Structural Analysis for obtaining the p- and t-invariant relationships of the net [12].

After performing the simulation of the net evolution, and obtaining the invariants relationships, very important conclusions on the net behaviour, concerning the dynamics of the modelled system can be stated, such as: boundedness, liveness, and reversibility of the net; reachability of some special state -marking- of the net; mutual exclusion among marking of places; capacity of storage zones and other modelled resources; the development of managing strategies for the optimal firing of transitions -optimal sequence of modelled operations- and refining of the modelled control processes, by adding, for instance, a new resource.

Only if the Petri Net model has good behaviour properties, like the ones above described, the designer can reach the next stage of the design methodology, using a temporized Petri Net model. This new stage corresponds to the performance validation of the modelled system.

SIMRdP enables to work with the following kinds of Petri Net models: ordinary Petri Nets, generalized Petri Nets, Petri Nets with predicates associated to their transitions and temporized Petri Nets with time associated to their transitions.

Remark: For modelling with each one of the above named kinds of Petri Nets, it is necessary to take into account the use of monitor places, when the net has places with capacity [11].

4.2 Performance analysis of FMS/FAS modelled through Temporized Petri Nets

4.2.1 Basic concepts

A Flexible Manufacturing/Assembly System (FMS/FAS) -whether simple or complex- includes any raw materials, tools, operators, and control policies for the operation of the equipments. At the sector level, it also includes flow of material and information through a collection of tools. It can even extend to the procurement, production and distribution networks of suppliers, plants, and distributors. Using computer simulation for modeling and analyzing such systems helps the designers to predict and improve a system's performance, as measured by such elements as capacity, cycle time, inventory, utilization, service level, and costs.

Once a model of the system is developed and running, the simulation qualitative analysis tool come to play. Information about the dynamics and kinematics of the involved equipment of the FMS/FAS, the product routing, and the production and parts-availability schedules must be inputs to the software simulator. Using the graphical user interface of the simulator, these inputs could be described and modified easily without programming.

Information such as the production capacity of machines on the FMS/FAS, the duration of various operations performed by different machines on different products, buffer sizes, and the staffing must be processed before they entry to the simulator, in order to allow their adaptability to the signal handling of the simulator.

Key outputs of SIMRdP are the system throughput (the quantity of products produced in each time period), a work-in-process inventory (the quantity of products being worked on in the system), the cycle time (the time from release of jobs into the system to completion), and machine utilization (the proportion of time the machines work on products). These results can be displayed as bar or line charts. By examining the above results, the designers are able to readily identify the system bottlenecks, and to suggest answers to how to alleviate the problems that may arise.

Since a simulation imitates the behaviour of a system as it evolves over time, basic to the approach of this simulation is the building of a model that highlights the vital characteristics of the system. Good models are needed to capture the characteristics of the Flexible Manufacturing Systems [10], namely: concurrency or parallelism, asynchronous operations, deadlock, conflicts and event driven.

4.2.2 Performance Indexes defined for the Flexible Manufacturing Systems

Taking into account the statistical information obtained from the simulation of the temporized Petri Nets, new performance indexes can be defined. Nevertheless, this new information must be refered to the dynamical behaviour of the modelled systems and their functioning specifications.

At this stage, performance evaluation in FMS/FAS allows to answer [8]., how many machines and/or automatic guided vehicles (AGVs),which transport topology and routing strategy are best suited to obtain, among others: small makespan and/or more balanced flows, optimal work in progress, minimal bottleneck situations, minimal death time for resources,etc.

According to the above considerations, the TPN model proposed in this work and the algorithm for its simulation, the following performance indexes for Flexible Manufacturing/Assembly Systems can be introduced [7]:

1. *Production period for a part (PPfP).* The total time necessary for the FMS to produce a part, calculated by taking into account the duration of all operations performed over the raw material taken from storages, over the intermediate manufactured parts, and the final

processed part stored into the product storages. From the TPN point of view, this index can be reached by computing the *Cycle time of the net* corresponding to the evolution from an initial marking characterized by: raw material storage busy, all resources of the system free, and product storages free, and another marking corresponding to a state having the product storage busy by the first manufactured part.

2. *Production period for a batch (PPfB).* The total time necessary for the FMS to produce a batch of parts, calculated by taking into account the duration of all operations performed n the raw material picked from the storages, over the intermediate manufactured parts, and the final processed parts deposited into the product storages. From TPN point of view, this index can be reached by computing the *Cycle time of the net* corresponding to the evolution from an initial marking characterized by: raw material storage busy, all resources of the system free, and product storages free, and another marking corresponding to a state having the product storage busy by all parts composing the batch.
3. *Percentual use of a resource related to a part (PURP)*: By analyzing the Gantt diagram of the temporized evolution of the TPN, and taken into account the production cycle for a part (PPfP), and all operations performed by the resource (modelled through transitions, this index is defined as:

$$PURP = \{[\sum_i \sigma_i^r . \theta(t_i^r)] / PPfP\}.100 \quad (1)$$

where

σ_i^r it is the number of occurrence of transition t_i^r, which models the operation op_i^r of the resource r over the part.

$\sum_i$ this summation takes into account all the transitions which model the operations of the resource under study.

4. *Percentual use of a resource related to a batch (PURB)*: By analyzing the Gantt diagram of the TPN temporized evolution, and taken into account the production cycle for a batch (PPfB), and all operations op_i^r performed by the resource r (modelled through transitions t_i^r), this index is defined as:

$$PURP = \{[\sum_i \sigma_i^r . \theta(t_i^r)] / PPfB\}.100 \quad (2)$$

where

σ_i^r it is the number of occurrence of transition t_i^r, which models the operation op_i^r of the resource r over the batch

$\sum_i$ this summation takes into account all the transitions which model the operations of the resource under study.

5. *Manufactured Parts per Time Units(MPpTU)*: Following the definition of "Percentual use of a resource related to a batch (PURB)", and considering the number of parts to be processed in a batch (b), the index is defined as:

$$MPpTU = b / PPfB \quad (3)$$

By considering the application of the above defined analysis methodology, the designers can

now draw major conclusions about the functioning of the modelled Flexible Manufacturing Systems. The optimization of the systems will depend on the final decisions imposed by cost factors, which can be derived from the above presented performance index. For example, if additional capital expenditures are made, what is the time frame for the increased throughput to pay back the investment? Cost analysis will be divided into three activities: assigning cost attributes to modelled resources, defining new performance-cost indexes, and analyzing the results after running one or more simulations.

5. SYNTHESIS OF CONTROL PROGRAMS

In previous chapters we discussed modelling aspects in terms of the functionality of the FMS/FAS. Once a system is evaluated and a good functional model is obtained, we have a platform on top of which we can run application programs.

Such programs or plans can be generated either manualy or resorting to an automatic or interactive planner. Independently of the process used, lets now discuss some aspects related to task plan representation.

5.1 Task Representation

A task plan can be represented by an hierarchical structure, following a NOAH - like model. High level operators are expanded into lower level operator [14]. This operator expansion ends when the lowest (primitive) operators are reached. Each primitive operator is performed by a resource agent (physical component). Grasp and ungrasp operations, for instance, will be supported by current tool controller.

In order to determine precedence between operations (precedence graph) two phases are normally considered:

1. product related constraints (determined during process planning)
2. manufacturing system constraints (determined during scheduling)

Once such ordering of actions is determined, we'll have a precedence graph for each level of the hierarchical representation of the plan.

Figure 9 illustrates a partial ordering for an intermediate level.

From each level a Petri Net which represents the assembly operations to be carried out, can be directly derived from the precedence graph. Figure 10 illustrates a Petri Net which represents the same partial assembly plan as in figure 9.

Each graph node can be expanded to another graph, where nodes correspond to next lower level operators. The same kind of expansion can be applied to the Petri Net model, where each place can be expanded to a new Petri Net

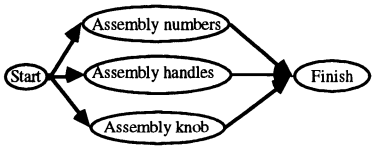

Figure 9 - Graph Plan.

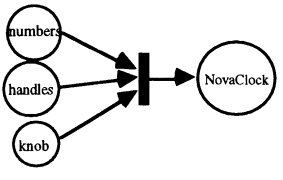

Figure 10 - Petri Net

5.2 Controller Synthesis

As mentioned before, Petri Nets are an important tool to model and evaluate the structure and behaviour of controllers and application programs in a manufacturing environment. Complex system dynamics can be described and analysed in a structured way (mathematical methods).The benefits of using a high level modelling tool, like a Temporized Petri Net (TPN) or a Predicate Transition Net (PTN), shouldn't end in their descriptive characteristics but there should be a direct connection between the description and the real controllers. This means that besides supporting the analysis and evaluation, the model could also drive directly its associated local controller. To assure this connection two different approaches could be followed: (1) using PTN to directly program the local controller, which implies a support by its manufacturer or (2) using a PTN translator which converts PTN to the language of the local controller.The first approach is unrealistic, at current stage, because the concept of PTN is not well disseminated among controllers' manufacturers, which makes the second approach a better one. The PTN description should be compiled in order to generate a program which interacts with the real system in the way described by the PTN.

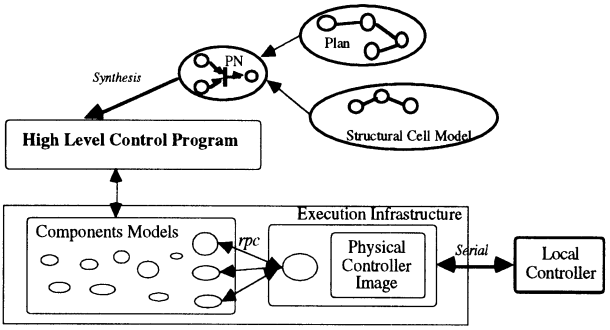

Figure 11 - Interactions between HLCP and Execution Infrastructure

The application program or High Level Control Program (HLCP) is generated from a PTN which describes components behaviour and interactions (Figure 11). The HLCP interacts with the execution infrastructure, mainly with the components models. These models are described using an Object/Frame paradigm. As mentioned before, the components operational behaviour is implemented by methods, which interact with the component's local controller through a server which supports an image of the local controller functionalities.

It should be noted that a Component Model can interact with more than one server, depending on the number of needed controllers to control the components being used. For instance there can be different servers to control de robot, the gripper, gravity feeders, etc.

In order to generate the HLCP from a PTN some considerations should be made about PTN. Components actions can only be done in places with marks. Predicates associated to transitions specify conditions. In order to match places to components actions, places' names include components' name and action name separated by an underscore (for instance, the name for the place that describes the grasp action of a robot is named **robot_grasp**).

The HLCP program generated is like a simulator of the PTN being modelled. Different PTNs may have the same kind of simulator, differing only in which order transitions will be fired and which actions will be done. Program generation module was developed using Prolog, and the generated program is also described in Prolog with a frames extension developed by UNL - Golog [15].

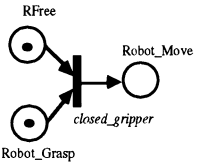

Figure 12 A Petri Net example.

The generated program obtained from the Petri Net described in Figure 12 can be seen below. Each section of the generated program is described.The first section is concerned about place definition; every place is defined by an object/frame whose main attribute is the slot mark to store the place's mark value. During this phase the program that contains the components model is consulted.

```
:- consult('models.pl').
:- new_frame(places), new_slot(places, mark).
:- new_frame('RFree'), new_slot('RFree', isa, places), new_slot('RFree', mark, 1).
:- new_frame(robot_grasp),new_slot(robot_grasp,isa,places), new_slot(robot_grasp,mark,1).
:- new_frame(robot_move),new_slot(robot_move,isa,places), new_slot(robot_move,mark,0).
```

After this, transitions are defined. Each transition is defined by checking its enabling condition. When this occurs, input places are updated and transition fires with output place updating and the corresponding method activation: *call_method(robot, move, [true]).* This method's code will send a message to the server which will react by sending the command "move" to the robot.

```
t1 :- get_value('RFree', mark, X0), X0 > 0,
     get_value(robot_grasp, mark, X1), X1 > 0,
     NVal0 is X0 - 1, new_value('RFree', mark, NVal0),
     NVal1 is X1 - 1, new_value(robot_grasp, mark, NVal1),
     call_method(robot, move, [true]),
     get_value(robot_move, mark, VOal0), NVOal0 is VOal0 + 1,
     new_value(robot_move, mark, NVOal0).
```

The main program consists of a forever cycle that continuously apply the existing transition names and randomly choose one which will be checked for its enabling condition.

```
rep_run([]).
rep_run(List) :- length(List, Tam), Pos is ip(rand(Tam)), position(Pos, List, Tr),
                 remove(Tr, List, RList), call(Tr, Success), !, fail == Success, rep_run(RList).
run :- repeat, rep_run([t1]).
```

This generated program runs with a similar behaviour as the PTN shown in figure 12.

6. CONCLUSIONS

In this paper we have presented a joint research work between UNL/UNINOVA and UNSJ, aimed at developing an hybrid platform for designing, evaluating and controlling Flexible Manufacturing/Assembly Systems. At present, the individual modules have been developed and separately tested and the main concepts related to their linking/integration have been designed.

A first applicative case was made for a subset of the NovaFlex pilot FMS/FAS system. Next phase will be the integration of individual parts through normalized models representation.

ACKNOWLEDGEMENTS

This work has been partly funded by the European Commission (ECLA Flexsys Project), JNICT (SARPIC Project), CYTED-D Programme and CONICET (Arg).

REFERENCES

1. Lee, D.Y. and F. DiCesare, *Scheduling Flexible Manufacturing Systems using Petri Nets and Heuristic Search,* in *IEEE Transactions on Robotics and Automation*199
2. Silva, M., *Las Redes de Petri en la Automática y la Informática.* 1985, Madrid - Espana: Editorial A.C.
3. Barata, J., L.M. Camarinha-Matos, and J.F.R. Chavarría, *Modelling, Dynamic Persistence and Active Images for Manufacturing Processes,* in *Studies in Informatics and Control*1994, p. 173-183.
4. Barata, J. and L.M. Camarinha-Matos, *Development of a FMS/FAS System - The CRI's Pilot Unit,* in *Studies in Informatics and Control*1994, p. 231-239.
5. Barata, J. and L.M. Camarinha-Matos. *Dynamic Behaviour Objects in Modelling Manufacturing Processes.* in *CAPE'95 - The Fifth International Conference on Computer Applications in Production and Engineering.* 1995. Beijing - China:
6. Tello, R. and A. Martínez, *Software for Analysis, Simulation and Validation of FMS,* E.E. Graduation Thesis, Universidad Nacional de San Juan, 1994
7. Colombo, A.W., *Modelling and Analysis of Flexible Production Systems,* MsC Thesis, Universidad Nacional de San Juan, 1994
8. Silva, M. and R. Vallete, *Petri Nets and Flexible Manufacturing,* in *Advances in Petri Nets.* 1989, p. 374-417.
9. Zhou, M., F. DiCesare, and A. Desrochers, *A Hybrid Methodology for Synthesis of Petri Net Modells for Manufacturing Systems,* in *IEEE Transactions on Robotics and Automation*1992, p. 350-361.
10. David, R. and H. Alla, *Petri Nets for Modelling of Dynamic Systems - A Survey,* in *Automatica - IFAC*1994, p. 175-202.
11. Ezpeleta, J. and J. Martínez. *Petri Nets as Specification Language for Manufacturing Systems.* in *IMACS World Congress on Computation and Applied Mathematics.* 1991. Dublin - Ireland:
12. Murata, T., *Petri Nets: Properties, Analysis and Applications,* in *Proceedings of the IEEE*1989, p. 541-580.
13. Colombo, A.W., *et al. Simulador de Sistemas Flexibles de Manufactura usando Redes de Petri Temporizadas.* in *6º Congreso Latino-Americano de Control Automático.* 1994. Rio de Janeiro - Brasil:
14. Camarinha-Matos, L.M., *et al., Interactive Assembly Task Planning and Execution Supervision,* in *Studies in Informatics and Control*1994, p. 185-193.
15. Lopes, L.S., *GOLOG - Um gestor de Objectos em Prolog* - 1993, DEE - UNLisboa,

BIOGRAPHY

Dr. Luis M. Camarinha-Matos received his Computer Engineering degree and PhD on Computer Science, topic Robotics and CIM, from the New University of Lisbon. Currently he is auxiliar professor (eq. associate professor) at the Electrical Engineering Department of the New University of Lisbon and leads the group of Robotic Systems and CIM of the Uninova's Center for Intelligent Robotics. His main research interests are: CIM systems integration, Intelligent Manufacturing Systems, Machine Learning in Robotics.

Dr. Ricardo Carelli is auxiliar professor (eq. associate professor) at the Electrical Engineering Department of the University of San Juan. His main research interests are: CIM systems , Manufacturing Systems.

Walter Colombo is a Phd student. His main research interests are: CIM systems, Manufacturing Systems.

José Barata received his Computer Engineering degree on Computer Science, from the New University of Lisbon. Currently he is junior assistant at the Electrical Engineering Department of the New University of Lisbon and belongs to the group of Robotic Systems and CIM of the Uninova's Center for Intelligent Robotics. His main research interests are: CIM systems integration, Intelligent Manufacturing Systems, Machine Learning in Robotics.