Modelling, Dynamic Persistence and Active Images for Manufacturing Processes

José Barata; L.M. Camarinha-Matos
Universidade Nova de Lisboa
Monte Caparica
PORTUGAL

José Freddy Rojas Chavarría
Universidad de Costa Rica
San Jose
COSTA RICA

Abstract: This paper presents a general overview of the research approaches in the topics of **modeling, dynamic persistence of objects and active images** for manufacturing processes as carried out at the University of Costa Rica and the New University of Lisbon. This paper was prepared within the context of the ECLA (European Communities and Latin America)co-operation programme on the CIMIS.net project. Current approaches of both universities are presented in order to define a reciprocal cooperation towards the definition of some general concepts and the design of some tools in the mentioned topics.

1. Introduction

Object Oriented techniques have been intensively used in modeling manufacturing systems and processes. OOP provides a structured modeling approach, allowing for multiple levels of abstraction, a convenient approach for complex systems modeling

Particularly in the area of shop floor control, OOP can be used to model various manufacturing agents - robots, NC machines, transportation systems or even continuous processes equipment. Classical real-time aspects, like asynchronous events / interrupts, device drivers, etc., may be adequately modeled/abstracted using an OOP approach.

Graphical interfaces stand for one important aspect in the field of process controllers. An integration of OOP techniques and active images provides a convenient framework for the development of advanced control panels and dynamic information browsers.

On the other side, in a manufacturing environment, many information sources - sensors, state variables of local controllers, etc. - have their own "life", independent of the computer that is running the general controller, because they have local processing power. This may lead to a concept of dynamic persistence, that we intend to explore in modeling manufacturing systems.

1.1 Basic Concepts

1.1.1 Dynamic Persistence of Objects.

Object Persistence is the property of extending the life of an object beyond the running session of the application software that created or changed it. This characteristic is important for applications that may interact with long lifetime objects.

The traditional way of dealing with Object Persistence is storing the objects in secondary memory. In some approaches, classical Database Management Technology has been integrated with OOP languages in order to manage the flow from main to secondary memory and vice versa.

The concept of Dynamic Persistence of Objects is not very different from that of normal persistence. A basic difference would be the way persistence is supported: by the local memory of devices' controllers.

The Object Oriented paradigm happens to be a good tool for modeling a manufacturing process (assembly, welding, ..), because it can support the modelling of both static and dynamic characteristics of the physical entities involved in the process. As a manufacturing process is composed of intrinsically dynamic entities, the most interesting aspect is the possibility of modelling the dynamic characteristics, those that express the behavior of the entities.

The use of reactive programming (demons) and methods to "link" the object model to the real cell controllers allows for a permanent update of the dynamic object model. In this way, a special kind of persistence is achieved - **dynamic persistence**. It is dynamic because the object model reflects, at every time, the status of the physical object. The persistence is ensured by the "memory" present in the device controller. There is a tight connection between the object "living" in the main memory and the physical controller. We can say that the object virtualises the physical controller.

The persistence of those objects could be supported using the traditional way of secondary memory. This approach seems redundant because the memory always present in the controllers can be used as long as the controllers are active. On the other side, this kind of dynamic information only makes sense or is useful when the controllers are active. For instance, the current position of a robot is a dynamic attribute that is important when that robot (and its controller) is active. If it is switched off, the current position is not dynamic any longer (or not important at all since a "hard home" action will be performed when it is switched on again). Therefore, traditional persistence may be used for the static properties of objects, but dynamic attributes are better modelled if resorting to the concept of dynamic persistence.

1.1.2 Active Images

Current technology of graphical interfaces offers the possibility of designing "control panels" and dynamic information browsers with special capabilities of displaying all kinds of elements such as: push buttons, scales, bars, sophisticated instruments (gauges) and schematic flow diagrams. When these graphical items, and in particular the gauges, are logically "linked" to object's dynamic attributes, they give, at any time, an updated pictorial representation of those attributes. Such dynamic graphics is called **"active images"**.

A programming framework combining active images with the notion of dynamic persistence, offers a very practical approach to designing interfaces for high level control systems.

1.2 Requirements in Costa Rica

In Costa Rica, at the current stage, continuous processes are more important than discrete event systems. However, some hybrid situations can be of interest if we plan high level plant-wide control systems. For instance, in a sugar plant, most processes have a continuous nature, but after packing the sugar, the last phase of the process is discrete.

From an implementation point of view, the locally developed process controllers are based on low level libraries embedded in the programming languages or some high level tools. Some of these systems offer graphical elements to show the values of variables and some capacities of handling curves of continuous processes. These systems often work with real time events. See Lab Windows, for instance.

On the other side, the installed computational park basically consists of PCs. Therefore, the OOP and Active Images supporting technologies, although promising in this industrial scenario, must be supported by small equipment.

The combination of the local experience and of the perception of industrial needs with the concepts developed at UNL is regarded as a promising cooperation.

2. Modelling

2.1 Modelling Aspects

The Object Oriented paradigm or its "mate" Frame-based/Reactive Programming represents a convenient tool to model the inherent complexity of a manufacturing system. This complexity results from the amount of relationships among components in association with the diversity of components. The topic of modeling is a pre-requisite for systems integration into CIM. Specifically speaking about system controllers, there is a need for a model to support the interaction between high and low level controllers, and, at the same time, support the configuration of new systems.

The model should emphasize the relationships among the various components in the cell and hide the specificity of hardware. This latter item can be easily achieved with the Object Oriented/Frame based paradigm using methods or demons. Methods associated with the component can hide the underlying hardware infrastructure.

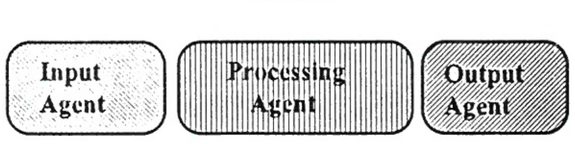

Figure 1. Cell Structure

In the following discussion the basic modeling unit will be a cell. A cell is a composite entity capable of making some transformation, movement or storage related to some product or part. In structural terms, each cell (C) has components to support the input of parts (I), an agent to perform the transforming actions (A) and components to support the output of products/processed parts (O). Therefore, a cell is the tuple: C = (I, A, O).

An example of a cell model:

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

Connected from is a relation that links the cell model to the entity(ies) performing input activities. The cardinality of this relation depends on the available number of input entities.

Connected to is a relation that links the cell model to the entity(ies) performing output activities. Again, the cardinality of this relation depends on the available number of output entities.

Processor is a relation that links the cell model to the agent performing transforming activities.

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

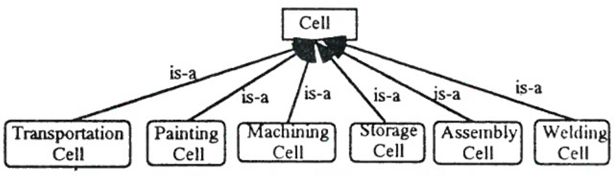

Figure 2. Examples of Types of Cells

Metaknowledge should be associated with each specialized cell to represent the specificities of its application area. For each domain the specific cell has the same structure as the generalized Cell concept (Input Agent, Processing Agent, Output Agent) but the domain of the implementing components is different in each specialization. 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. Metaknowledge seems to be a very important element at the configuration phase, certifying the validity of cells.

An assembly cell can be described as:

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

On the other hand a Painting cell can be described by:

FRAME PAINTING-CELL
is-a: CELL
val-inp-ag: buffer, agv, conveyor
val-out-ag: conveyor, agv, buffer
val-proc-ag: robot

The input and output activities can be performed by several agents, i.e. to perform these activities there are several candidates, depending on the application.

At this stage it is convenient to make a distinction between the concepts of agent and component/manufacturing resource.

For instance, the model of a robot **component** is a context independent description of its static and dynamic characteristics. 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** in different **contexts**. The (expected) behavior of a robot in an Assembly context is different from its behavior in a spot welding context.

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

Let us consider an example (Figure 3):

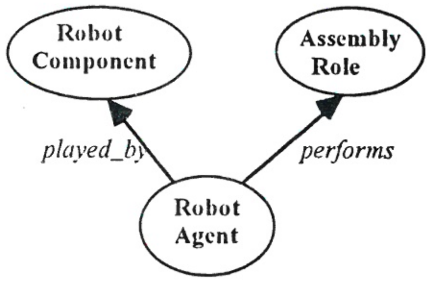

Figure 3. Structure of a Robot Agent

An example of a robot component model:

FRAME ROBOT_COMPONENT
is-a: manufacturing_component
Base_coordinate_system:
Controlled_by:
Applications: assembly, gluing, ..
DOF: 6
Working_area:
Load:
Repeatability:
Current_position:
Cost:
Manufacturer:
Cycle_Time:
Next_maintenance:
N_working_hours:
Weight:
Resolution:
Max_speed_by_axes:
etc.

In this model, *Controlled_by* is a relation that links the model of the robot to the model of its controller. An example of this model is presented below:

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

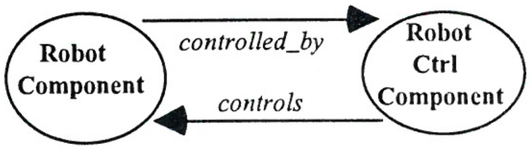

Figure 4 . *Controlled_by* and *Controls* Relations

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

The operations specified in the inherits slot are inherited by the robot component.

Example of role definition:

FRAME 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
main_attributes: force_sensor, current_tool, available_tools, available_resources
component_attributes: Base_coordinate_system, Controlled_by, Working_area, load, Current_position

Main_attributes is a slot related to the inheritance mechanism of *performs* relation. In this case, it specifies which are the characteristics of assembly role that will be relevant to a processor agent.

Component_attributes has the same functionality as *main_attributes*, but, in this case, associated with the relation *played_by*. This slot describes the most relevant component attributes that are important to the processor agent.

Tools_domain and *aux_res_domain* represent domain-knowledge that is important during configuration time.

Current_tool is a relation that associates the main player of this role (robot component) to a particular tool.

Assembly_device is an attribute describing where assembly operations are really executed. Fixture1 is an instance of a component specialized in holding parts.

RELATION CURRENT_TOOL
is-a: relation
type: intransitive
inherits: inclusion (tool_operations, tcp, tool_status)
inverse_relation: used_by

Finally, an example of an agent is:

FRAME ASSEMBLY_ROBOT
is-a: agent
performs: ASSEMBLY_ROLE
played_by: ROBOT_COMPONENT

The relation *performs* associates an agent with a specific role, an example is:

RELATION PERFORMS
type: intransitive
inherit_slot: main_attributes
inverse_relation: performed_by

The slot specified in the *inherit_slot* contains the slot names to be inherited by the processor agent.

The relation *played_by* associates an agent with its intrinsic properties (components). An example is:

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

The slot specified in the *inherit_slot* contains the slot names to be inherited by the processor agent.

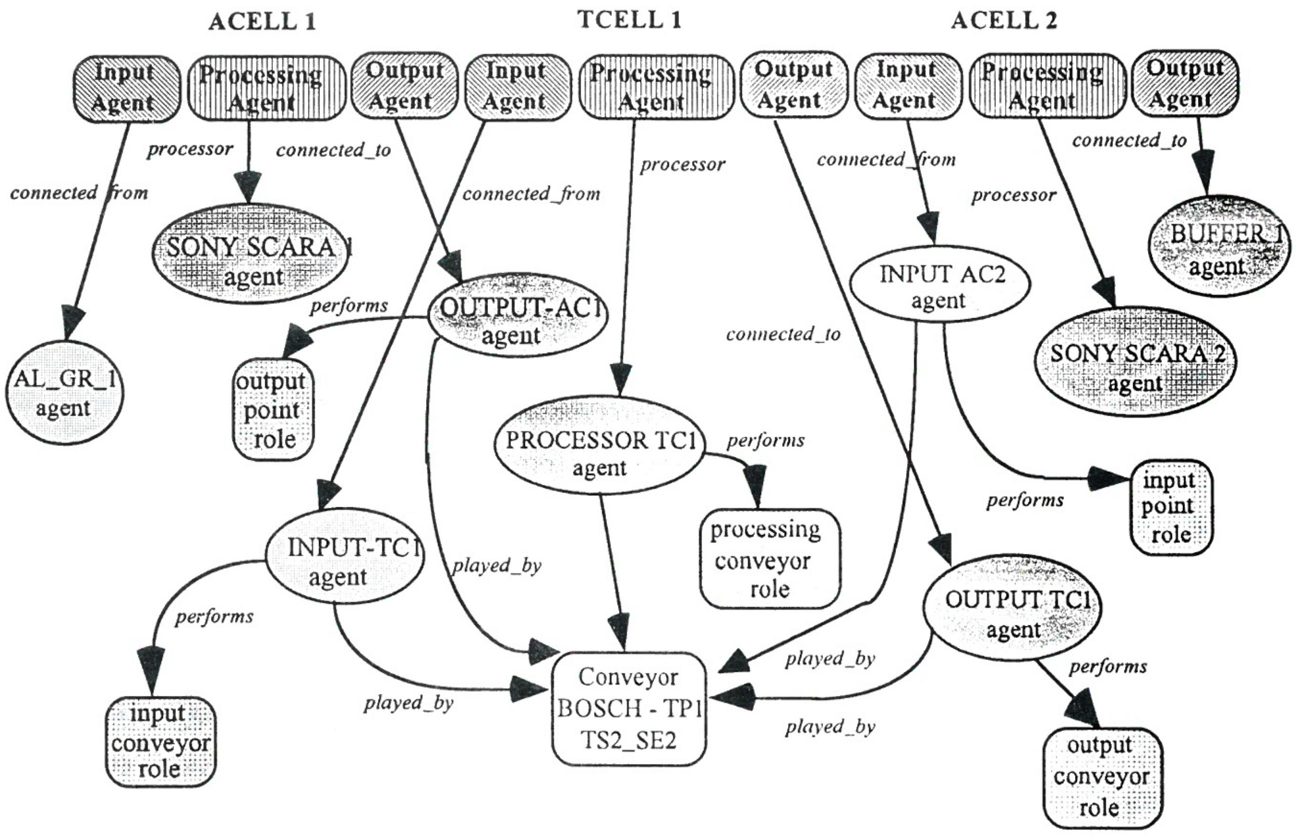

Figure 5 . Two Assembly Cells Connected by a Transportation Cell

A manufacturing system can be composed of several of the previously described specialized cells. The connections among them are possible by: (1) sharing of input/output agents or (2) via a transportation cell. A cell can have several input and output agents, i.e. a cell may have more than a single input or output point. The raw- materials flow and the assembly/subassembly parts are both managed by input and output agents.

In a multi-cell complex system the *Connected_to* and *Connected from* relations describe how cells are inter-connected.

A very interesting cell case that should be pointed out is the transportation cell. It can be used to connect other types of cells ensuring the flow of raw-materials and the assemblies/subassemblies or parts. The cell's processing agent could be a Robot, an Agv or a Conveyor. What makes this concept interesting, allowing more flexibility, is the fact that a conveyor or an agv is not an input or output component, but they can play as transportation cells agents.

In order to clarify these concepts, an example of an assembly system with 3 cells: 2 assembly cells and a transportation cell are presented (Figure 5).

ACELL1 is an assembly cell with a gravitic feeder to supply parts and a robot to assemble those parts. Processed parts are placed in a conveyor to be transported to ACELL2. In ACELL2 some processing activities are done by an agent robot and delivered to an output buffer. TCELL1 is a transportation cell which connects assembly cell 1 to assembly cell 2.

ACELL1 and TCELL1 share a physical picking point. The output point of assembly cell 1 (ACELL1) and the input point of transportation cell 1 (TCELL1) is the conveyor input point (ref_origin). The output point of transportation cell 1 and the input point of assembly cell 2 (ACELL2) is the conveyor output point (destination_ref).

BOSCH conveyor component is shared by 5 different agents (OUTPUT-AC1, INPUT-TC1, PROCESSOR-TC1, OUTPUT-TC1 and INPUT-AC2), via *played_by* relations.

An example of the instances involved in the Transportation Cell TCELL1 is presented:

```
FRAME TCELL1
    instance_of: conveyor-transportation-cell
    name: TCELL1
    global_referential: ref0
    processing_products: pallets
    input_parts: pallets
    Connected_from: INPUT-TC1
    processor: PROCESSOR-TC1
    Connected_to: OUTPUT-TC1
```

This frame was created during configuration phase using Metaknowledge included in the TRANSPORTATION CELL model.

```
FRAME BOSCH_TS2_SE2
    instance_of: conveyor_component
    brand_name: BOSCH
    transportation_family: TS2
    unit_type: SE2
    ordering_number: 3842999120
    system_designation: TP1
    track_width: 240
    length: 3000
    powered_by: electric
    transportation_speed: 12
    motor_power: 370
    reversible: false
    belt: non-conductive
    controlled_by: SONY_CTRL
    base_coordinate_system:
    origin_ref: ref1
    destination_ref: ref2
    current_product:
    transports: WT2
    load_method: by_robot
    unload_method: by_robot
    forward: method forward_fn
    backward: method backward_fn
    stop: method stop_fn
```

```
FRAME INPUT_CONVEYOR_ROLE
    is-a: role
    available_part:
    main_attributes: available_part
    component_attributes: transports, load_method,
            origin_ref, base_coordinate_system
```

Attributes available in INPUT-TC1, via *performs* and *played_by* relations are:

```
FRAME INPUT-TC1
    is-a: agent
    origin_ref: ref1
    performs: INPUT_CONVEYOR_ROLE
    played_by: BOSCH-TS2-SE2
    [transports]: WT2
    [load_method]: by_robot
    [base_coordinate_system]: reft
```

Attributes within brackets are inherited.

```
FRAME PROCESSING_CONVEYOR_ROLE
    is-a: role
    main_attributes: nil
    component_attributes: forward, stop, backward
```

Attributes available in PROCESSOR-C1, via *performs* and *played_by* relations are:

```
FRAME PROCESSOR-TC1
    is-a: agent
    performs: PROCESSING_CONVEYOR_ROLE
    played_by: BOSCH-TS2-SE2
    [forward]: method forward_fn
    [backward]: method backward_fn
    [stop]: method stop_fn
```

```
FRAME OUTPUT_CONVEYOR_ROLE
    is-a: role
    main_attributes: nil
    component_attributes: transports, load_method,
            destination_ref, base_coordinate_system
```

Attributes available in OUTPUT-C1, via performs and played_by relations are:

```
FRAME OUTPUT-TC1
    is-a: agent
    performs: OUTPUT_CONVEYOR_ROLE
    played_by: BOSCH-TS2-SE2
    [destination_ref]: ref2
    [transports]: WT2
    [load_method]: by_robot
    [base_coordinate_system]: reft
```

For ACELL1, a particular aspect regarding the output agent is exemplified by:

```
FRAME OUTPUT_POINT_ROLE
    is-a: role
    moving: false
    main_attributes: moving
    component_attributes: origin_ref,
            base_coordinate_system
```

```
FRAME OUTPUT-AC1
    is-a: agent
    performs: OUTPUT_POINT_ROLE
    played_by: BOSCH-TS2-SE2
    [origin_ref]: ref1
```

[base_coordinate_system]: reft

For ACELL2, a particular aspect regarding the input agent is exemplified by:

FRAME INPUT_POINT_ROLE
is-a: role
available_part:
main_attributes: available_part
component_attributes: destination_ref,
base_coordinate_system

FRAME INPUT-AC2
is-a: agent
performs: INPUT_POINT_ROLE
played_by: BOSCH-TS2-SE2
[destination_ref]: ref2
[base_coordinate_system]: reft

This modular approach to cell representation facilitates the development of complex systems by simple "concatenation" of cells.

A particular manufacturing unit is made up of several subsystems (Transportation Cells, Painting Cell, Assembly Cell, ...). A manufacturing unit could be modeled by a SYSTEM entity, which has access to all characteristics and to the functionality of all subsystems involved in the Unit via the *has_subsystem* relation. The applications (controllers, for instance) are accessing the unit only through SYSTEM.

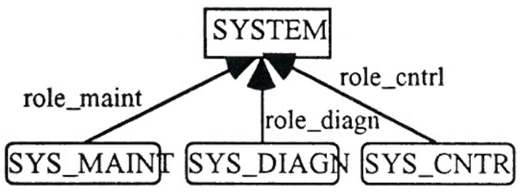

Figure 6. Different SYSTEM Views

The way applications see the unit varies with their needs. An application concerned with maintenance activities has different <u>needs</u> from SYSTEM than an application concerned with supervision activities. These differences could be easily reconciled using the <u>view</u> concept. Using this concept an application only sees the information relevant to its activity.

This is a very convenient concept because it supports information structuring and consistency. Every component belonging to a basic cell unit has all kinds of information, but not all this information will be used within the same context. Thinking of a robot, an attribute that counts the number of its working hours, will be important in case of a maintenance application, but it would be irrelevant to a direct control application.

These views represent the roles performed by SYSTEM. The role relationships encapsulate the information needed to provide views. Every relation has a kind of Metaknowledge specifying what information is inherited by each view. Every view corresponds to a different entity (SYS_MAINT, SYS_DIAGN, SYS_CNTRL). Applications have access to the specific view they are interested in. The validity of each access should be confirmed by the view entity.

We can even think of applications in the same activity area, i.e. accessing the same view, but having different requirements. In this case the access is determined not only by the role of the client but also by its status.

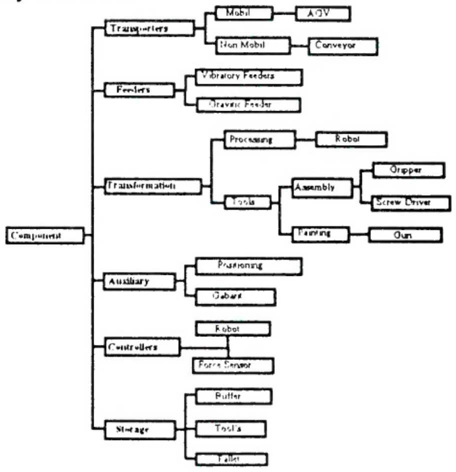

Figure 7. Partial Taxonomy of Components

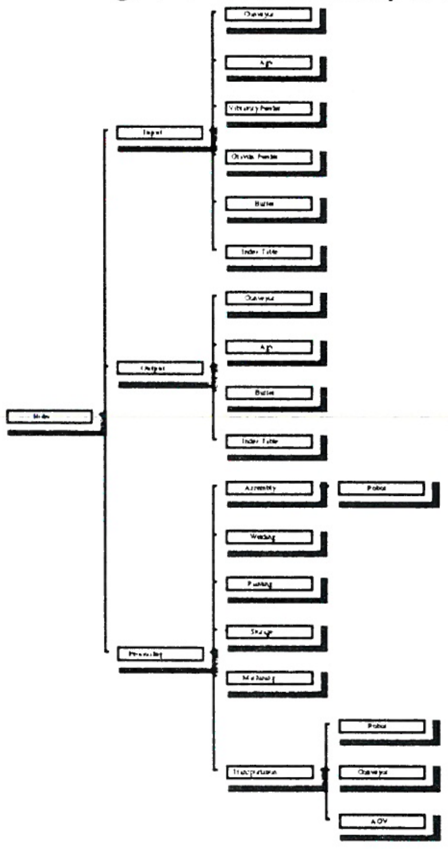

Figure 8 . Partial Taxonomy of Roles

The status of applications may differ in accessing a view entity, having conditioned access determined by their status.

In order to create and configure cells it is convenient to have catalogs of components/manufacturing resources and roles organized as taxonomies.

Figure 7 illustrates a partial taxonomy of components.

The components are fully described in this taxonomy. This description contains all the information available about the component, from mechanical to electrical/electronic characteristics, not only those that are relevant to the control but also those relevant to configuring a system.

The taxonomy is organized by components' functionality.

Figure 8 illustrates a partial taxonomy of roles.

2.1.1 UNL Experiences

UNL has been involved, for several years, in manufacturing systems modeling and CIM information systems [1], [2], [4].

The obtained results include an Express-based Engineering Information Management System, featuring multiple representation interoperability and a multi-user integrating infrastructure into support the integration of a heterogeneous community of application modules in CIM.

A supervision architecture built on top of EIMS provides a first step towards a platform enabling concurrent engineering.

Modelling experiments of robotized cells have been made using Express, frame representation systems and OOP. Current activities address the dynamic persistence aspects and the modelling of cells from a monitoring point of view. The objective is to develop machine learning techniques for monitoring, error diagnosis and recovery.Cell adequate models are a prerequisite of the development of a learning cell controller [6]. Another complementary direction in cell modeling is that towards the development of a distributed multi-agent dynamic scheduling system [7].

2.1.2 UCR Experiences

Nowadays in Costa Rica the use of computers to control industrial processes is not generalized and few places use them. Most of the systems are based on "personal computers" and the main operating system is DOS. Some other systems are based on UNIX or OS/2, in particular for modelling and simulation of hydraulical or paper plants.

Some companies are working with leased equipment, or have invested in the use of PLCs (Programmable Logical Controllers) with or without a link to a PC.

The most common logical link between computers and processes is based on the use of some languages like: Pascal, C, C++, Assembly language. In all these cases it is necessary that a lot of codes for using the computer resources should be generated (code for interrupt routines, manipulation of stacks, queues, drivers, and so on). As a result, libraries of basic resources are now available and, together with PLCs, do offer a starting platform for the next stage of integration. From the point of view of hardware, the main link between computers and PLC information is provided by communication processors. Specific resources libraries to "encapsulate" this link had to be developed.

As an example, one experimental prototype developed by UCR and called SLMTR was presented at the COPIMERA workshop. This system is like a Shell written in the Assembly language, C and C++, comprising a Real Time-Multitasking environment, a Real-Time hardware driver for data acquisition and control, an object oriented graphical interface and a database to be used in DOS with the C language.

Currently, some people are working with OS/2, UNIX and it is possible to have leased workstations for process control. Some applications are ready for Microsoft Windows for PC linked with PLC and there are some other applications based on Lab Windows and Lab View for PC.

Most of these experiences were related with continuous processes and not with discrete manufacturing. Current goal is to develop a mixed platform to support both continuous and discrete processes in order to implement a CIM concept in the typical industrial scenario of Costa Rica.

2.2 Physical Connection with Data Sources

2.2.1 UNL Experiences

The Intelligent Robotics Group of UNL has got experience in using reactive programming to connect frame based models with local controllers of manufacturing equipment [1], [3]. Figure 9 illustrates the basic principle. Each time we try to access the Location attribute of the object/frame Robot, a demon is fired and a connection with the robot local controller is established. This approach was used, for instance, in an experimental set-up for the execution supervision of assembly tasks

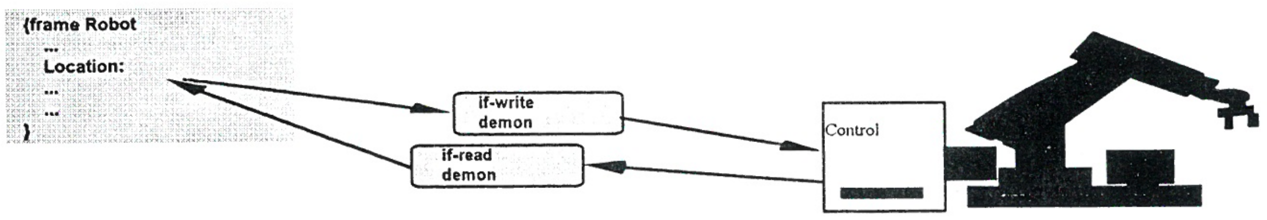

Figure 9 Use of Reactive Programming To Support Dynamic Persistence

The robotic assembly cell is composed of one robot SCARA (SONY SRX 4CH), three robot tools (grippers), magazine and corresponding tool exchange mechanism, two special purpose feeders and one fixture, a force torque sensor and several binary sensors.

The actual communication link, in a UNIX environment, is supported by Remote Procedure Calls (RPCs) - Client-Server model. In fact, a Cell Front End composed of a set of server processes, was developed under UNIX environment and provided the interface with different parts of the Assembly Cell. One of these processes is dedicated to interfacing the Robot Controller. The main functions made available in this server process are related to robot movement commands:

- move to a point
- move to point in linear interpolation
- move to point in circle interpolation
- relative movement in each axis
- relative movement in four axes
- forward shift
- point definition
- set velocity
- set overtime
- set acceleration time
- set hand system
- delay
- step stop

An interpreter of these commands was written in the Robot Controller language. The input and output ports of the Robot Controller, each having 16 independent bits, can be used to actuate other resources of the cell, such as robot tools, feeders and fixture and to obtain status information from simple sensors like proximity sensors. Therefore, the Robot Controller Server also provides read and write operations in those I/O ports.

Another server process is the Teach Pendant Emulator, which, according to its name, emulates the Teach Pendant of the robot. This is achieved by sending through the serial line of the Teach Pendant signals simulating its buttons. As this robot controller is not very friendly regarding its integration into a higher level system, this process accomplishes several tasks of great importance. In the first place, to download a program to the Robot Controller and make it operational, obviously, it is necessary to observe a handshake protocol involving the Teach Pendant and the Programming Unit.

In our case the program that will run in the Robot Controller is a command interpreter (a materialization of the server commands) that has been previously stored there. The Teach Pendant Emulator is used to render the command interpreter the default program and then start its execution. When some fault occurs which causes breakdown of the Robot, the Teach Pendant is normally used to reinitialize it. The Teach Pendant Emulator is used to perform the same task automatically.

Finally, the Force/Torque Sensor Server process performs all communications with the Force/Torque Sensor Controller. Its main functions are:

- sensor calibration
- consult Force/Torque values in an instant
- trace Force/Torque values over a period of time.

Via Remote Procedure Calls (RPC), these server processes make the commands of the controllers of the cell resources available in UNIX. Another layer above this, the Virtual Cell Controller, combines the commands in order to produce a set of services that represent the global functionality of the assembly cell.

The frame/OOP based model of the cell components get access to these front end services via demons attached to the dynamic attributes.

A large scale experimentation is now being carried out for the Pilot FMS/FAS Unit of the Center for Intelligent Robotics of the UNINOVA Institute.

2.2.2 UCR Experiences

In Costa Rica the connection between the environment of physical processes and the data structures is often made using low level drivers.

Those systems are frequently based on hardware interrupts which usually asked for all codes being written from scratch.

The most common design models are based on two layers: one for a physical connection with hardware and another for providing a logical connection between the logical driver and the application. The physical connection has a low level handler which reacts to hardware interrupts. This handler does not know anything about the main application; it is normally a hidden object.

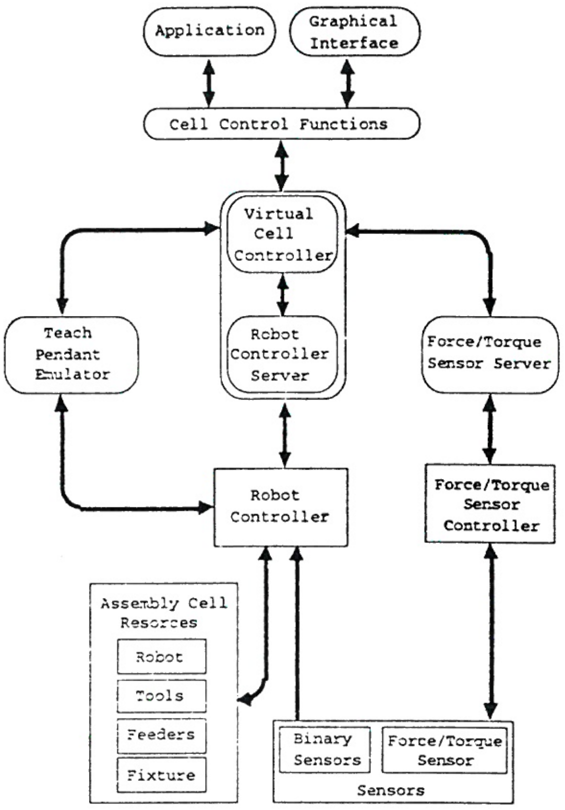

Figure 10. Infrastructure for Control of a Cell

A logical connection is the way how to link the results of the hardware interrupts (layer one) with the application. It is a data structure which hides the physical world and only presents an input/output driver (black box) for the application.

To build the logical connection between software and hardware, OOP techniques are used. In this way, the system user needs know nothing about the physical connection. In particular, EXPRESS like representations, integrating information originated from digital sources is being explored in order to read a common integration platform

3. Active Images

As mentioned before, the main idea is to associate "dynamic" images, like gauges, with the object/frame attributes that represent the dynamic aspects of a modeled system. Each time the internal model is changed, i.e. a dynamic attribute gets modified,and its corresponding graphical visualisation should implicitly be updated.

3.1 UNL Experiences

UNL´s experiences with active images have consisted in a frame modeling system and in reactive programming.

Figure 11 illustrates the general approach. Demons are used to update gauges logically attached to dynamic attributes. Please note that the same attribute -- as a dynamic one -- may be linked, also via reactive programming, to the local device controller.

This illustrates an integration of the dynamic persistence and of the active images concept.

3.2 UCR Experiences

In Costa Rica the use of graphical environments with active images is not new, however, the research on such concepts is not large and often comes to support some special applications. Wide- spread commercial environments are: Lab Windows

(National Instruments), Windows (BC++), Lab View, and so on. SLMTR has its own small graphical active images interface to support the real-time multitasking tool. The most common applications with active images are oriented to benefit the user and only some of them have been devised to work with a real- time process environment.

Some people are working with Lab Windows (National Instruments), but this software has some inconveniences when sophisticated applications are intended. In the case of Microsoft Windows, the applications are made in Borland C++ but take a long time effort to implement and overcome some problems with the concurrent environment of Windows. For OS/2 the process of designing some applications is just at the beginning..

In the framework of CIMIS.net and of the joint co-operation UCR/UNL, an integrated prototype for alarms control based on the presented concepts, is being developed, in parallel with a CIM approach to information integration .

4. Conclusions

The combination of OOP/Frame-Based modeling, dynamic persistence and active images concepts provides a very convenient tool for enabling the development of advanced controllers of manufacturing systems.

Starting from two complementary experiences, both in terms of application domains -- discrete and continuous processes -- and tools -- AI or low level based approaches -- a fruitful co-operation the two groups of UNL and UCR embarked upon. Reaching a common understanding of the problem and finding a common glossary, have been the first phase objectives. Next phase will pursue the development of a general methodology for the use of dynamic persistence and active images in manufacturing processes as well as for a deeper integration of continuous and discrete event systems.

Acknowledgments
This work was funded in part by the European Communities (B-LEARN, FlexSys and CIMIS.net projects), and JNICT (SARPIC and CIM-CASE projects).

REFERENCES

1. STEIGER-GARCAO, A. and CAMARINHA-MATOS,L.M., **Design of a Knowledge-based Information System**, in . Bernhardt, Dillman, Hörmann, Tierney (Eds.) Integration of Robots into CIM, Chps. 21 & 22 CHAPMAN & HALL, 1992.
2. CAMARINHA- MATOS, L.M. and SASTRON, F. **Information Integration for CIM Planning Tools**, Proceedings of CAPE'91 IFIP Conference on Computer Applications in Production and Engineering, ELSEVIER Publishers, Bordeaux, France, 10-12 September 1991.
3.CAMARINHA-MATOS,L. M. and SEIGER-GARCAO, A., **Knowledge Architecture for Flexible Programming of Robotic Cells,** Proceedings of the 20th International Symposium on Industrial Robotics, Tokyo, Japan, 4-6 October 1989.
4.CAMARINHA-MATOS, L.M. and OSORIO, L., **CIM Information Management System - An Express-based Integration Platform,** Proceedings of the IFAC Workshop on CIM in Process and Manufacturing Industries , CHAPMAN & HALL Publishers, Espoo, Finland, 23-25 November 1992.
5.OSORIO, A.L. and CAMARINHA-MATOS, L.M., **Information- based Control Architecture for CIM**, Proceedings of IFIP International Conference Towards World Class Manufacturing, Phoenix, Arizona, USA, 12-16 September 1993.
6. CAMARINHA-MATOS, L.M., LOPES, L.S. and BARATA, J.,**Execution Monitoring in Assembly with Learning Capabilities,** IEEE Int Conference on Robotics & Automation, San Diego,CA, USA, May 1994.
7. RABELO, R. and CAMARINHA-MATOS, L.M., **Control and Dynamic Scheduling in Virtual Organization of Production Researchers,** IFIF Int Conference on Evaluation of Production Management Methods, Porto Alegre, Brazil, March 1994.

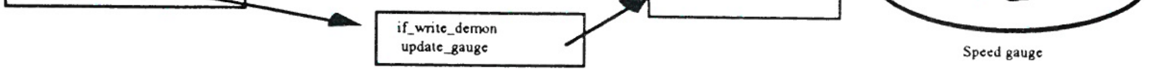

Figure 11. ReActive Programming as a Tool To Implement Active Images