9

An Object-Oriented Approach to the Design of Flexible Manufacturing Systems *

J. Reinaldo Silva †
Computer System Group, University of Waterloo, Waterloo, Canada
e-mail: reinaldo@csg.uwaterloo.ca

H. Afsarmanesh
Computer System Department, University of Amsterdam, The Netherlands
email: hamideh@fwi.uva.nl

D.D. Cowan
Computer System Group, University of Waterloo, Waterloo, Canada
email: dcowan@csg.uwaterloo.ca

C.J P. Lucena
Computer Science Departement, PUC-Rio de Janeiro, Brazil
email: lucena@inf.puc-rio.br

Abstract

In this paper a hybrid top-down/bottom-up method that can be viewed as an extension of the traditional dynamic modeling technique using Petri Nets and Parametric Design is presented as an approach to the design of Flexible Manufacturing Systems. The resulting method supports a clear separation of functionality among the design objects by using the ADV/ADO object-oriented design framework. Thus, the designs as well as the general functional models can be reused. Comparing the method described in the paper with the object-oriented architecture introduced and employed in the PEER object-oriented database system suggests an implementation approach which can support the object clustering properties of ADV's and ADO's.

Keywords

OO-design, Flexible Manufacturing Systems, Petri Nets, Abstract Data Views, design reusability

1 INTRODUCTION

Flexible manufacturing systems often have two conflicting characteristics: a clear physical model given by a set of production processes and machines, and a logical or abstract model usually related to a production plan and accompanying control algorithms. Both models are

*This work is part of the research activities of the Esprit/ECLA Flexsys 76101 and Cimis.net 76102.
†Partially supported by FAPESP. On leave from University of São Paulo, Brazil.

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

necessary to describe the complete system and yet the design methods related to the two models are inherently incompatible. Such a substantial difference in design approaches could easily lead to errors at the early stages in the lifecycle of the entire artifact‡, because the products of the design at this stage are mismatched and relationships are not always clear. Errors made in these initial phases are very expensive to remove since they are often found late in the development process as the integrated flexible manufacturing system is realized.

Development of a method for design of logical or abstract objects such as software that would simplify conversion to prototypes might alleviate this design issue. Then the logical products of the design could be incorporated into the physical model and the incompatibilities could be observed and corrected. Several attempts have been made to find such a method, most of them associated with software development. Typical examples are rapid prototyping, bottom-up (Sommervile, 1992), outside in (Marca, 1988) development and object-oriented design [Booch 91]. Development of such a method might also make design formalisms and tools more attractive to the practicing design engineer because of the ability to produce a prototype that would make the benefits of formal approaches more concrete, thus bridging the gap between new achievements in Design Theory and practical applications in Engineering Design.

In this paper we describe a hybrid design method based on providing abstractions of both the logical components such as the software, and the physical components of the flexible manufacturing system. The abstractions represent the behavior and functionality of both types of components through their interfaces (Cowan, 1995). By using suitable constructors to combine the components and appropriate information hiding mechanisms we can reuse both types of components and hence use previously designed elementary components. The external behavior of each component could be validated at any point in the development process using its dynamic model expressed in PFS/MFG.

External behaviors or interfaces and their corresponding object models are represented here by Abstract Data View /Abstract Data Object (ADV/ADO) pairs (Cowan, 1993a)[Alenc 94](Cowan, 1995) where ADVs are extensions to the object model to support the specification of interfaces. The design approach will be demonstrated using an example of a discrete control algorithm for shop floor control (Bruno, 1986). Finally we describe an implementation of these ideas based on a federated architecture using the PEER database (Afsarmanesh 1993)(Tuijnman, 1993)(Afsarmanesh 94).

2 DESIGN OF FLEXIBLE MANUFACTURING SYSTEMS

A Flexible Manufacturing System (FMS) is a cooperating set of process machines (usually numerically controlled) connected by an automated transportation system (Tempelmeier, 1993). We add to this definition by allowing the FMS to have local storage facilities and local computers dedicated to information handling and control. These added components are not necessary to justify our technique, but were introduced in order to test our approwith a complex system.

A version of the life cycle for an FMS (Tempelmeier, 1993) is presented in Figure 1, where the two main activities of integrated production planning and model optimization are highlighted. Most of the current design methods focus on these activities since they are critical factors

‡The term artifact is used to represent the goal of the design process, and could refer to a physical object such as a mechanical part or a logical object such as a software system or a control algorithm. or some combination of these two categories.

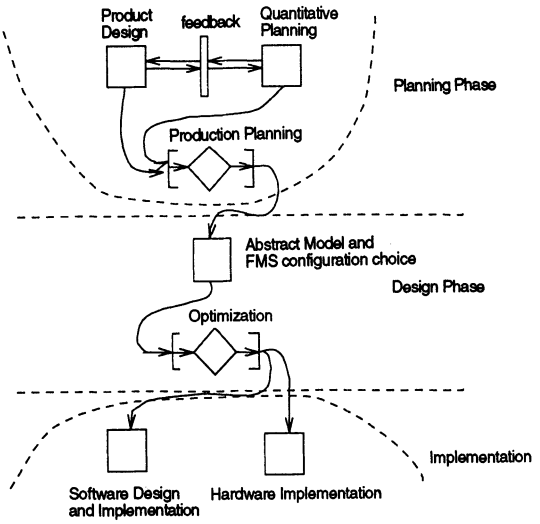

Figure 1 Life cycle of an FMS in PFS/MFG representation

in producing a good design. However, modules such as Product Design, Software Design (especially associated with cell and factory control) and Modelling could make significant contributions to the overall result if a more flexible design approach is used.

We concentrate on an approach that reinforces abstraction in the modelling and evaluation of configuration options, and that could provide the basis for further optimization as well as requirement and production plan validation. The representations we use are intended to support software design and specification. We use a general integration plan as input and rapidly generate abstract models of the FMS configurations that could be refined further through the proper choice of machines and layout. The advantage of this approach is that consistent abstract models can be shared by all those professionals with different backgrounds participating in the design thus, making it easier to reinforce concurrence at each design phase. In addition, when changes and adaptations are required, a common occurrence in today's dynamic marketing environment, the appropriate abstraction will be available.

Another important aspect of FMS design is the manner in which the proposed approach handles complexity. Complexity depends on the size of the system (the number of machines) and of the degree of flexibility, where flexibility is a function of the number of interrelation operations. Current definitions of FMS (Ranta, 1990) range from small (2-4 machines) to complex systems (15-30 machines). Complexity can vary with the size of FMS in several ways. For instance, if a set of numerically controlled machines do the same set of operations with a mix of products, the workload balance will be simpler than if the set of operations is different and the manufacturing of each type of product in the mix has to be put in a sequence in addition to balancing the overall workload. Thus, modeling is very important for small systems, since small and medium-size enterprises usually attempt to obtain more and varied production from small FMSs.

Reuse of old models stored and retrieved from a database can substantially reduce the cost of

redesign and adaptations. Our design approach is reuse-oriented in that components and their relationships are specified solely by their interfaces.

2.1 Petri Net and PFS/MFG Modeling

Petri Net methods have been used extensively to model FMS (Silva M., 1993)[Proth 93](DiCesare, 1993)(Proth, 1993a). Such methods provide formal models for FMS, and are able to handle the complexity inherent to features such as dependencies between cell operations, parallelism, concurrence of tools and/or material. However, Petri Net models (principally those based on Place/Transition Nets [Reisig 89] do not adequately support reusability or abstraction.

Recently, there has been more research in high-level Petri Nets with the objective of introducing abstraction into net-oriented design methods. PFS/MFG (Program Flow Schema/Mark Flow Graph) is one type of high-level net 1988 [Miyagi 88] specifically for the design of discrete manufacturing systems. The idea is to add abstraction to the modeling power of Petri Nets (Condition/Event) and produce less complex graph models for qualitative and quantitative analysis. The structure of a specific PFS/MFG representation can be translated into Dynamic Logic (Silva J., 1992), and then artificial intelligence tools can be used to assist with the modeling and analysis of target systems (Lucena, 1989).

A revision of the conventional PFS/MFG representation that was introduced in (Silva J., 1992a) allows the simulation of a Petri Net model at any level of abstraction, thus, making these formalism more suitable for the top-down design of discrete production systems. The revised formalism was used in a cognitive design model [Takeda 90] (Tomiyama, 1992) that was applied to FMS in (Silva J., 1994). The addition of modularity, information hiding and separation of concerns indicated that a hybrid approach to design may be possible.

3 THE ADV/ADO MODEL

Some of the characteristics of the hybrid-design approach presented in this paper such as, abstraction, information hiding, nesting, and polymorphism can be found in object-oriented design [Booch 91], an approach to design that is primarily bottom-up. However, the majority of the practical design problems in engineering, particularly in the design of FMS, have a functional flavor, and are more closely related to top-down methods. We do not intend to debate which method is more appropriate, rather we wish to combine and integrate the two approaches to obtain a good design solution.

Thus, we first tried to find a framework where functionality could be implicitly or explicitly applied depending on the specific requirements of a particular design phase. Such a feature could be used to combine existing detailed elements with those for which only the general behaviour is known. In other words, we could combine existing elements (reusable blocks) with abstract descriptions. The ADV/ADO (Cowan, 1993)(Cowan, 1993a) which was originally created to allow a clear separation of the interface including the user interface from the application component in software design is used in this hybrid-design approach.

An Abstract Data Object (ADO) has a static description/model of the artifact and methods (behaviors) that can query or change its internal state. ADOs can be combined through operators for composition (nesting) and aggregation (Alencar, 1994) to build complex elements.

The interaction between ADOs, or between an ADO and an external medium such as a user or network, is through an interface object called an Abstract Data View (ADV). ADVs are in

fact extended ADOs that handle input and output events or the exchange of messages and data among existing ADOs. An ADV specifies the external behavior or functionality of a model because an ADV specifies an interface to an ADO. For instance, a numerically controlled (NC) machine with a local magazine can be represented by an ADO where its interaction with the outside world is an enclosing ADV that accepts a program of operations and returns an error message, a progress report of the process, or an acknowledgement indicating that the operation was completed. Both the ADV and ADO are different objects which have their own behavior (methods). Interactions between them can be represented by another ADV which is responsible for the coupling or aggregated behavior, as might happen during the downloading of an APT program and the loading of the NC machine.

This simple example illustrates how to represent the strong separation between the pre-conditions to evoke the machine services and the process. The example could be extended to a manufacturing center, or a Flexible Manufacturing Cell and its integration in the overall FMS.

Top-down design is supported since ADVs provide a functional design interface connected to abstract models of subsystems that could be developed later. Instead of specifying a specific DNC machine or setting general features such as the number of axes or the set of tools in our simple example, we could perform a design with a statistical model of success performing operations or a lower bound time on machine operations as qualitative criteria. Such qualitative rules can guide the modeled behavior encapsulated in ADV's and could be developed later in the ADO model.

Two types of consistency relations between ADVs and ADOs are defined in [Cowan 94a]: horizontal and vertical consistency. A vertical consistency relation is defined between an ADV and its owner ADO as:

$$\mathcal{R} = \{(x, y) | x \in S \wedge y \in P \wedge ADO_j \vdash (x \rightarrow y)\} \tag{1}$$

where S and P are respectively the set of inputs and outputs of the ADV. Thus, any transaction with the outside world must be valid in the ADO model.

The same ADO or subsystem can interact with the external world in several different ways (Tomiyama, 1992), and they must all be consistent with each other. This form of consistency is called horizontal consistency in (Cowan, 1993a). We rephrase horizontal consistency in our cognitive model of design with the expression:

If $\mathcal{A}^j$ is the set of ADV's for the ADO j and

$$\exists \mathcal{A}^j{}_i.(x, y) \in \mathcal{R}_{ij} \Rightarrow [\exists \mathcal{A}^j{}_k.(x, z) \in \mathcal{R}_k j \Rightarrow z = y] \tag{2}$$

that is, ADV's cannot contradict each other.

The concept of consistency ensures the correct integration of concurrent designs or existing components such as the software and hardware implementation in Figure 1 and provides a strong foundation for the reuse process.

Two critical operations in the reuse process are: the search for a reusable component, and the adaptation of suitable candidates in the overall design. The efficiency of the first operation depends upon reducing the search space by providing in advance expected characteristics of the candidates or by performing the search using very short descriptions or metaphors for the components. We believe that in engineering and especially in the design of FMS, synthetic functional models described by ADVs would provide a good search space for reuse if vertical consistency with the ADOs is guaranteed. The importance of consistency to this reusability was

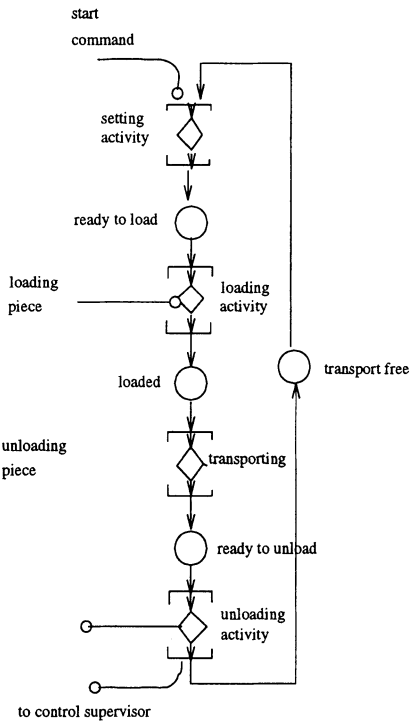

Figure 2 Model definition of a transportation system which takes one piece at a time

discussed in [SilvaJ 92]. The integration of reusable components can be guided by functionality and the maintenance of horizontal consistency.

More details about the concept of ADV and ADO can be found in [Cowan 94a] and in the related bibliography mentioned there. In the next section we will describe a short example to illustrate the basic concepts.

4 A SMALL EXAMPLE

In this section we revisit a short example proposed by (Bruno, 1986) to show how the hybrid method could be applied to the design of FMS. This example will be recast using the ADV/ ADO approach with PFS/MFG used as a modelling tool.

Initially, we will try to identify the objects of the system [Booch 91]. In the area of FMS and Computer Integrated Manufacturing (CIM), composable objects can be easily identified. Some of the basic FMS constituents are: a transportation system, a NC machine, local storage and a local controller. The composed system is a flexible cell totally automated and controlled by signals which are sent from a central station.

A conventional Petri-Net approach (Silva M., 1993)[Proth 93] would model each one of these objects with decision-free nets, that is, nets without any kind of conflict, leaving all decisions to

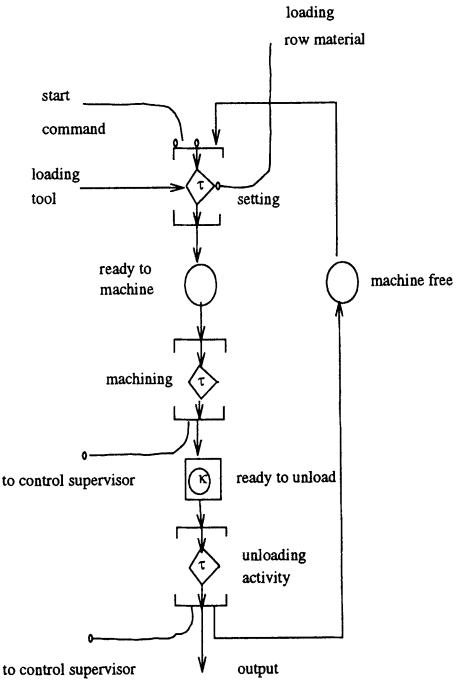

Figure 3 Model definition of a machine process

the control module. The internal behaviour of each object would be modeled in order to identify the control entries and outputs. For the current example, we just rewrite the abstract models presented by (Bruno, 1986) in PFS/MFG.

In a first approach, the transport facilities would be modeled as a system that processes one item at a time (an AGV). The whole operation can be started by a control sign (all remaining tasks such as adjusting positions and tracing a trajectory are performed locally). Figure 2 shows a PFS/MFG representation of the transport system.

Notice that Figure 2 is a decision-free net with external interactions with the control system represented by the start control signal and the signal sent to the supervisory system saying that the transportation process is finished. From the point of view of generating control algorithms and software, we could say that the interactions with the control system represent the principal control flow, to distinguish them from the (secondary) flow of items, such as the exchange of a piece with the outside (load and unload operations). Such a distinction is a key point in the modelling of discrete systems (Proth, 1993)[DiCesa 93][Miyagi 88].

The PFS/MFG high level representation of an NC machine is illustrated in Figure 3, and the storage system is represented abstractly in the diagram of Figure 4. We assume in this example that a complete design for these three elements can be reused taking the abstract model presented in Figures 2, 3 and 4 as a "target model" (Silva J., 1992). Our focus will be in the design of the discrete controller for the FMC.

In the revised presentation of PFS/MFG (Silva J., 1992), a flow relation is defined as an

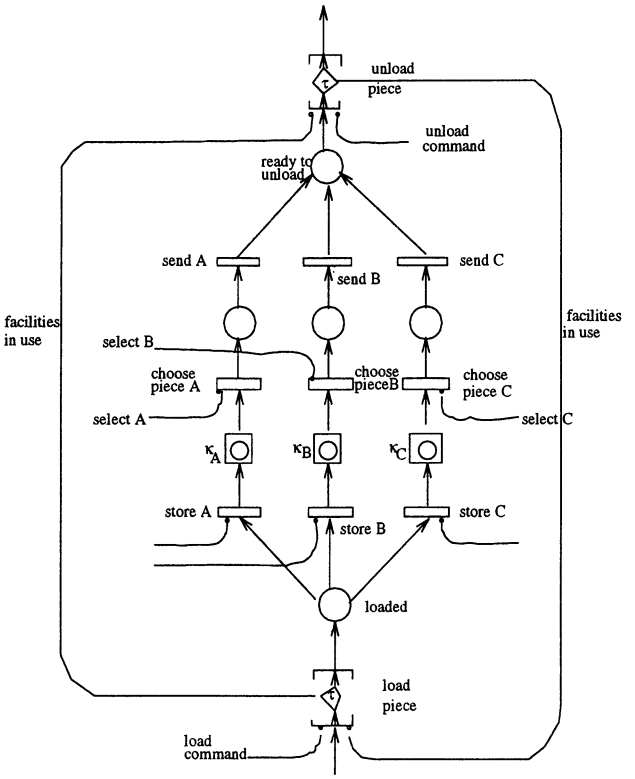

Figure 4 Model definition of the storage system.

object instance of a more generic class: the general relation between boxes (passive elements) and activities (active elements). Boxes are represented by B and each box has an attribute "kind" to indicate whether the box represents a storage, assembly, or distribution element[Miyagi 88]. The activities are classes with at least one attribute to specify the estimated time spent answering a call or finishing a process operation (if t=0 the activity will collapse into the representation of an instantaneous event of the conventional C/E Petri Net). The relationship between boxes and activities is given by:

$$\mathcal{F} = (B \times A) \cup (A \times B) \tag{3}$$

Each pair in this relation is a class, whose subclasses are *gates* and *flows*. Gates stand for non-structured relations and could be instantiated by external or internal gates. External gates represent control signs or calls for external pieces, information, or material, such as the *start command* in Figure 2 (a control sign), the *loading/unloading piece* command in Figure 2 (a call for an external piece), and the *to control supervisor* in Figure 3 (information about the process operation). An example of an internal gate is the *facilities in use* connection in Figure 4.

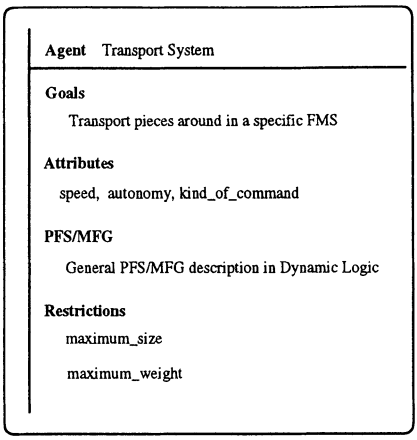

Figure 5 Informal definition of the ADO for the Transportation System.

Introducing this internal gate relation synchronizes the load/unload operation in the storage system, a requirement motivated by a need to share a manipulator robot[§].

Activities represent actions which are accessible only if their pre- and post-conditions are activated and deactivated, respectively. Activities can also encapsulate a cluster of other activities and conditions. For instance, the setting activity in Figure 2 can be refined as a subnet (also without conflicts) composed from two other activities, the loading of a piece (from outside) and the loading of a machine tool (from an internal magazine). Of course, each one of these aggregated activities can be further refined.

As mentioned earlier, the elements such as the transportation system, machine (a numerical control process) and storage are basic objects in the domain of Factory Automation and can be reused through a "standard" object-oriented technique or a combination of these methods and search techniques based on analogy or metaphors (Silva J., 1992). These reusable components can be integrated using a top-down design of the control system, which is depicted later in this paper.

The models of reusable components can be encapsulated as ADOs, together with some short documentation and other attributes (including a detailed model expressed in PFS/MFG). The representation of the ADOs use the techniques described in [Fields 93]. For instance, the ADO *Transport System* is shown in Figure 5.

An important feature of our design representation is that we could analyse the target system properties and/or simulate its behaviour at any level of abstraction by using the PFS/MFG internal model, which is a valuable validation mechanism for control engineering. One function of the ADVs is to specify the proper pre- and post-conditions for the internal activities encapsulated in an ADO. For instance, in the Figure 6 we have a very abstract ADV, representing the major functionality of the transport system which is to move pieces. As it is shown in Figure 2 the operation of a successful transport system depends on external signs and interaction with the control station. The ADV that represents the functionality of the transport system combines its

[§]The introduction of this requirement here suggests that "ad hoc" requirements or economic constraints could be included in the design method.

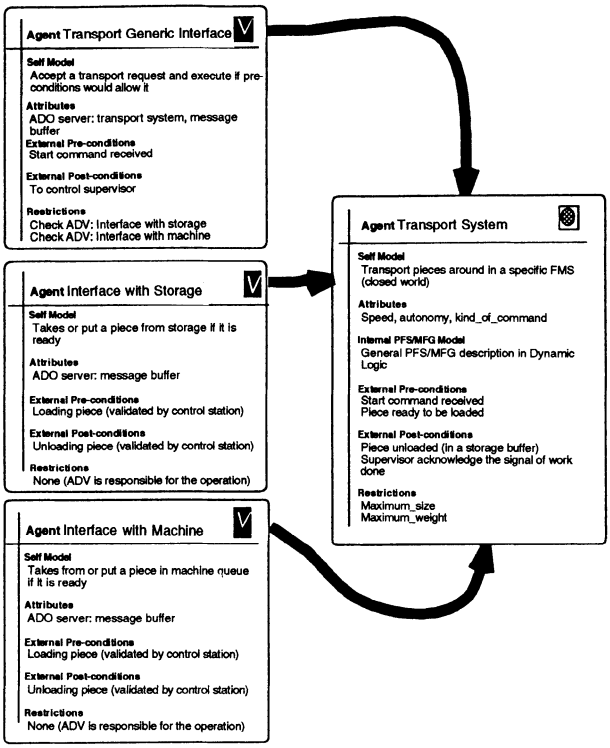

Figure 6 Cluster of behavior and model for a Transport System.

features with features of other ADVs to model the interaction with the control station. Hence, a request to transport a piece from one location to another would depend on the state of the condition *freeAGV* (see Figure 2), which is based on the current state of the ADO model of the AGV, and on the interaction with the ADVs that represent the interface between the AGV and the control station, the supervisor, and the supplier/consumer of pieces.

If we build all the basic elements as encapsulated ADOs and ADVs, the construction of the control software requires two substantial steps:

i) build all ADV links between the element objects (storage, machine and transport system) and the control station ADO;
ii) support all links with discrete control modules and algorithms.

Figure 7 shows a schema of the ADV-links between the control station and the other elements already described. This is the core specification to model a centralized controller which is the most common solution for a control problem where all decisions are left to the controller. However, it should be noticed that a similar framework based on the ADV/ADO approach would work for other control techniques, where local controllers would have a more complex role in the process or a hybrid solution of clusters with more intelligent local controllers and pure server mechanisms.

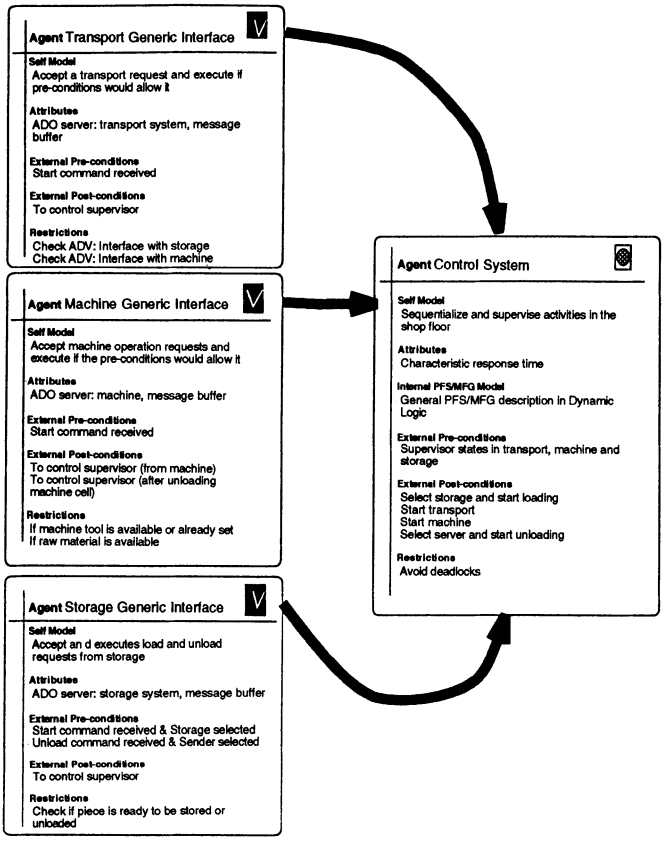

Figure 7 Cluster of ADV's connections with the Control System.

For instance, suppose that the transportation system receives a signal to start an operation¶. However, the loading process depends on the transportation system to recognizing in real time if the piece is ready to be removed. We are assuming it could be done by sensor signals whose interpretation would lead to a decision without any external intervention. That would make the system more reliable, even if in our case study the controller needs to ensure that the correct piece was selected or stored.

Supporting a combination of centralized and distributed systems is very important in the design of modern factory automation where legacy systems (relying on centralized control) are merged with modern autonomous processes.

Another interesting application of hybrid (centralized/distributed) systems is to the information-control problem, that is, to place information systems that supply information to control decisions at different abstraction levels. (Kagohara, 1994) shows a model based on layers, composed of production planning followed by a product design (CAD and CAPP) and finally a shop-floor control-layer. A system to generate coordinated control plans can also be designed using ADVs and ADOs as in the current example.

¶Since there is only one storage system and one machine, a simple argument would be enough to denote an operation from the storage to the machine or from the machine to the storage.

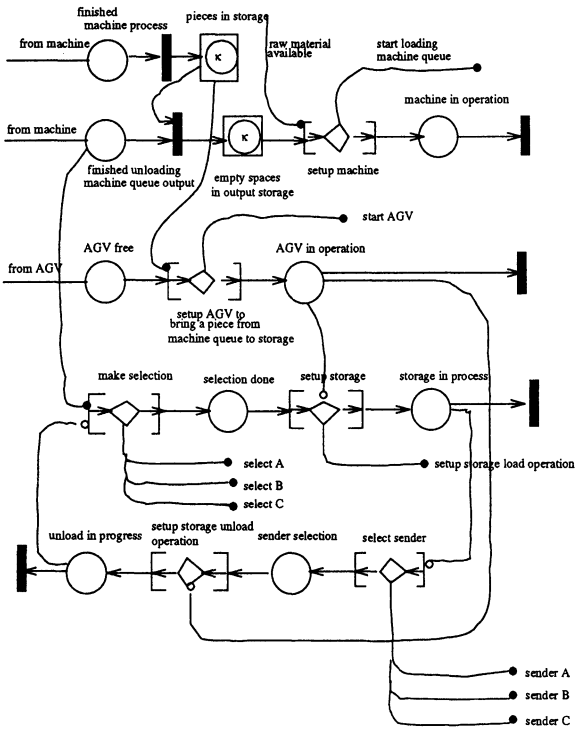

Figure 8 High level modelling of the discrete controller.

Figure 7 shows the abstract model of the control algorithm, and Figure 8 shows details of its internal PFS/MFG net, where the output events represent that control signs sent to the operator display or supervisory system.

In the next section we show the connection of the methodology with the information-sharing mechanism of the PEER database.

5 PEER IMPLEMENTATION APPROACH

As described earlier in the paper the model description and definition chosen for the ADV/ADO is closely related to the object-oriented approach of the PEER federated database management system. In this section, a brief description of the PEER system will be presented, and then by applying PEER to the example presented in Section 4, an implementation approach for the ADV/ADO model of the shop floor environment is addressed.

5.1 PEER database architecture

PEER (Afsarmanesh, 1993)(Tuijnman, 1993)(Afsarmanesh, 1994) is a federated object-oriented database management system designed and developed at the University of Amsterdam. A

federated PEER network consists of a loose federation of autonomous, heterogeneous distributed database systems. In the remainder of this section, we give an overview of several concepts developed in PEER to support the rich and complex CIM application area. PEER supports autonomous cooperating agents which share and exchange information. This is achieved by a sophisticated schema derivation/integration mechanism, which supports importing remote information and restructuring and integrating it with local information (Afsarmanesh, 1993). The sharing of information among team members in PEER is negotiated to preserve the referential integrity (Tuijnman, 1993). PEER also offers support for object clusters shared in the network. An object cluster represents an entity that groups together a set of objects that are interrelated through a directed acyclic graph hierarchy. The representation, identification and boundaries of object clusters, and a linearization mechanism to transform object clusters into a linear format is fully described in (Tuijnman, 1993).

The object-oriented data model and the language of PEER is primarily based on 3DIS (Afsarmanesh, 1989). However, this model has been extensively extended to support more semantics, to represent specific concepts and entities related to manufacturing, and to support the kernel structure for the distributed architecture of PEER. The PEER data model supports the fundamental abstractions of instantiation, generalization, and aggregation. Any identifiable piece of information (both data and meta-data) is uniformly represented as an object. PEER offers a number of facilities that are useful in Concurrent Engineering for CIM environments and are briefly described in this section. The nucleus of a PEER system in a CIM cooperation network is the PEER model and language, and a layer of modeling constructs and operations that are defined specifically to represent the generic abstract data types used in engineering and industrial manufacturing. The distributed object management of PEER is handled by a layer on top of the nucleus, that supports the distributed schema management and object sharing. For the exchange of object clusters as entities between PEER agents and between a PEER agent and an application program, a linear representation is generated. The last layer of the PEER architecture is the interface that supports the access to and communication with users and application programs at the agent level.

Information management in a network of agents is supported by PEER through distributed schema management including integration and derivation of local information and information available from other PEER agents. The semantic interrelationships (loose or tight integration) among the data and knowledge of different agents are established systematically and incrementally. Several schemas coexist in every PEER agent; namely, the local, export, import, and integrated schemas. The local schema LOC in each PEER agent specifies the type structure of the information stored locally at that agent. Part of the local information can be made available to other PEER agents, by specifying one or more export schemas (EXPs), that defines a view on the local information. An export schema in PEER can restrict the exported local information available to other PEER agents. Other PEER agents can acquire these export schemas and designate them as import schemas (IMPs), thus, making the information of other PEER agents available locally. The integrated schema (INT) defines a single uniform type structure on the information and specifies the derivation and integration of the local and imported information. Since the integrated schema is local to a site, different PEER agents may establish different correspondences between their schema and other sites' schemas, thus there is no single global schema for the network.

The implementation of the PEER federated system is written in the C programming language, and runs on a network of SUN workstations. This implementation supports a distributed multi-node environment and includes two tools, a schema manipulation tool (SMT) and a database

browsing tool (DBT). PEER tools are developed using X-windows on SUN workstations. PEER has been implemented at the Computer Systems group of the University of Amsterdam. PEER has been used in the ESPRIT ARCHON project No. 2256 (designing an architecture for the cooperation of expert systems in a multi-agent system), and the ESPRIT CIM-PLATO project No. 2202 (supporting the coexistence of diverse CIM tools in a planning tool box), and is currently being applied to the ESPRIT CIMIS.net basic research project ECLA 004:76102 (focusing on distributed information systems for CIM), and the DIGIS project (the integration of genomic information systems).

5.2 Application of PEER to the example

In this section we briefly describe the implementation of the small example defined in Section 4, using the PEER architecure. For every agent defined for the environment, such as those represented in Figures 6, 7, and 8, a corresponding PEER agent will be defined. The LOC schema for each agent will contain all the information that is locally stored in that agent. Therefore, the "Self Model", "Attributes" and "restrictions" are represented in this schema. While the "Attributes" will be represented in PEER by static PEER objects, the "Self Model" and "restrictions" will be represented by dynamic PEER objects. Dynamic objects or behavioral objects in PEER consists of three categories. One category defines the "constraint evaluators". The "restrictions" defined for agents in Figures 6, 7 and 8 fall into the category of constraint evaluators. Constraint evaluator objects have an associated executable piece of code (method). This code will run after any relevant modification to the data of this agent. Constraint evaluators will check for the consistency of the database state. If the data is modified in a way that violates the restriction rules defined for the agent, then the database is in an inconsistent state, and the modification will not be accepted and must be redone. The "Self Model" defined for the agents in the example consists of the routines that must be executed when the information about the "External-Pre-conditions" are satisfied. The routines performing the self model produce some new results that must be stored within the agent and produce the information about the "External-Post-conditions". Using PEER the "Self model" of an agent will be represented as a dynamic object of the category "storage transaction". Storage transactions are long routines that can be executed and will produce some data to be stored in the agent.

Every agent defines a number of EXP schemas derived from the LOC schema that supports the sharing and exchange of information among the agents and consequently provides the means for cooperation among distinct agents. The EXP schemas defined by one agent represent the part of LOC information that this agent will share with other agents. Another Agent can import an EXP schema (called IMP schema there) and then integrate it with its LOC schema to create its integrated view (INT schema) of all the information that it needs to access. The information represented as "External Post-condition" in an agent will be included in an EXP schema so that other agents can access. Therefore, other agents can access these post-conditions to verify their pre-conditions. Namely, another agent's (A2) external post-condition that is included in A2's export schema will be imported by this agent (A1) as its imported (IMP) schema to become A1's external pre-condition. Every agent will create its own INT schema, through integrating its LOC schema with its IMP schemas. Thus, an agent through its INT schema has access to all the information it needs to check for its pre-conditions and to run its self model.

A PEER implementation of the ADV/ADO system is planned. This implementation will follow the guidelines described above for the definition of the agents involved and their interconnections.

6 CONCLUSION

In the present work we proposed a method based on the object-oriented ADV/ADO framework which comprises visual approch to design (based on Petri Nets and its extentions) and parametric design (based on objects). Bottom-up and top-down approaches are nested in a way that allow the designer to control and document the design process using the same graph formalism applied to artifacts (Silva J., 1992). PEER database supports the parametric and object-oriented composition of models and also may provide a basis for object-oriented design reuse.

In the future we plan to build some realistic applications in PEER and to combine in the same design environment, PEER database, a PFS/MFG object-oriented simulator and a software agent to control and generate queries to PEER according to the needs of the design process.

7 REFERENCES

Afsarmanesh H. and McLeod D. (1989) The 3DIS: An Extensible Object-Oriented Information Management Environment, ACM Transaction on Information Systems, 7:339–377

Afsarmanesh, H., Tuijnman, F., Wiedijk, M. and Hertzberger, L.O. (1993) Distributed Schema Management in a Cooperation Network of Autonomous Agents. In Proceedings of the 4th IEEE International Conference on "Database and Expert Systems Applications DEXA'93", Lect. Notes in Computer Science (LNCS) 720, pages 565–576, Springer Verlag.

Afsarmanesh, H., Wiedijk, M. and Hertzberger, L.O. (1994) Flexible and Dynamic Integration of Multiple Information Bases. In Proceedings of the 5th IEEE International Conference on "Database and Expert Systems Applications DEXA'94", Athens, Greece, Lect. Notes in Computer Science (LNCS) 856, pages 744–753. Springer Verlag.

Alencar, P.S., Carneiro-Coffin, L.M., Cowan, D.D., Lucena, C.J.P. (1994) The Semantics of Abstract Data Views: A Design Concept to Support Reuse-in-the-Large, In Procedings of the Coloquium on Object-Oriented in Databases and Software Engineering (to appear), Kluwer Press.

Bruno, G., Balsamo, A. (1986) Petri Net-Based Object-Oriented Modelling of Distributed Systems, OOPSLA'86 Proc.

DiCesare, F., Mu der Jeng (1993) Synthesis for Manufacturing Systems Integration, in Practice of Petri Nets in Manufacturing, DiCesare, T., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, G.B., (eds) Chapman & Hall.

Cowan, D.D., Ierusalimschy, R., Lucena, C.J.P. (1993) Abstract Data Views, Structured Programming, 14 (1), 1-13.

Cowan, D.D., Lucena, C.J.P. (1993a) Abstract Data Views: A Model Interconnection Concept to Enhance Design for Reusability, Technical Report 93-52, Cjmputer Science Department and Computer System Group, University of Waterloo.

Cowan, D.D., Lucena, C.J.P. (1995) An Specification Concept to Enhance Design for Reuse, to appear in IEEE Transactions on Software Eng.

Fields, B., Harrison, M., Wright, P. (1993) From Informal Requirements to Agent-Based Specification: An Aircraft Warning Case Study, in, Procc. of the Workshop on Specification of Behavioral Semantics in Object-Oriented Information Modelling, IIM (Inst. of Inf. Modelling), Robert Morris College.

Kagohara, M., Toledo, C., Silva, J.R., Miyagi, P.E. (1994) Automatic Generation of Control

Programs for Manufacturing Cells, IFIP Transactions: Applications in Technology, B-19, pg. 335-343.
Lucena, C.J.P, Silva, J.R. et al. (1989) The Specification of a Knowledge Based Environment for the Design of Production Systems, 6th. Sym. on Information Control Problems in Manufacturing Technology, INCOM, Madrid.
Marca, D. (1988) DADT: Structured Analysis and Design Techinque, McGraw Hill.
Miyagi, P.E, Hasegawa, K., Takahashi, K. (1988) A programming Language for Discrete Event Production Systems Based on Production flow Schema and Mark Flow Graphs, Trans. of the Soc. of Instrument and Control Engineers, vol 24, no. 2.
Proth, J. M. (1993) Principles of System Modeling, in Practice of Petri Nets in Manufacturing, DiCesare, T., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, G.B., (eds) Chapman & Hall.
Ranta, J., Tchijov, I. (1990) Economics and Success Factors of Flexible Manufacturing Systems: The Conventional Explanation Revisited, IJFMS, 2, pg. 142-154.
Silva, J.R. (1992) A Formalization to the Design Process Based on Theory of Metaphors: Its Application to Discrete Events System Automation, Ph.D. thesis, (in Portuguese) University of São, Brazil.
Silva, J.R., Pessoa, F.J.B. (1992a) Análise Semi-Automatica de Mark Flow Graphs, Ibero-American Workshop in Autonomous Systems Robotics and CIM, Lisbon.
Silva, J.R., Cowan, D.D., Lucena, C.J.P (1994) Case-Based Approach to the Design of Flexible Manufacturing Systems, (to apear).
Silva, M. (1993) Introducing Petri Nets, in Practice of Petri Nets in Manufacturing, DiCesare, T., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, G.B., (eds) Chapman & Hall.
Sommerville, I. (1992) Software Engineering, Addison-Wesley Pub. Co.
Tempelmeier, H., Kuhn, H. (1993) Flexible Manufacturing Systems: Decision Support for Design and Operation, Wiley Series in System Engineering, John Wiley & Sons.
Tomiyama, T. et. al. (1992) Systematizing Design Knowledge for Intelligent CAD Systems, Human Aspects in Computer Integrated Manufacturing, G.J. Olling and F. Kimura (eds.), Elsevier Science Publishers.
Tuijnman, F., Afsarmanesh, H. (1993) Management of Shared Data inFederated Cooperative PEER Environment, Jour. Intelligent and Cooperative Inf. Sys. (IJICIS).