Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

Conceptual Modeling of Autonomous Agents paradigms in Flexible Manufacturing Systems

Ludovica Adacher, Carlo Meloni

RT-DIA-44-1999 **Novembre 1999**

ABSTRACT

In this paper is described a set of possible implementations of the autonomous agents concept in flexible manufacturing. Development of intelligent systems to improving manufacturing productivity needs good software modeling approaches to support efficient design and control. Software design concepts based on object-oriented programming are emerging as powerful techniques for developing large scale software systems. This paper presents important features of object-oriented design tools and their relevance in modeling and developing software for Autonomous Agents Manufacturing Systems.

Keywords: Agents, Conceptual Modeling, Coordination, Decentralized Control Systems, Intelligent Manufacturing Systems, Object-Oriented Programming, Simulation.

1 Introduction

Production systems are distributed systems working in a dynamic environment. A recent approach providing a decomposed and modular framework is based on the paradigm of autonomous agent (AA) (Jennings, and Wooldridge, 1998; Barber, et al., 1998), in which the process is the result of decision of several entities each pursuing its individual goal, without a global decision maker.

The autonomous agents concept suggests basic ways of dealing with complex systems, such as the decomposition approach (Uzsoy, and Ovacik, 1997). The concept generalizes and integrates other alternative control and scheduling architectures (i.e. cooperative systems, heterarchical structures, object-oriented programming, real time negotiation of resource assignment, and opportunistic scheduling) designed to meet the complexity as well as the distributed nature of the process (Lin, and Solberg, 1992). The authors propose a framework for parts flow management based on a negotiation protocol among parts and other resources. The effectiveness of the approach has been verified by means of simulation experiments (Adacher, et al., 1999).

In the organization of the factory, it is natural to identify the agents with the elements having a certain amount of behavioral autonomy, be it a department, an office, a machining center, a single worker or a workpart (Adiga, 1989; Blazewicz, et al., 1998; Pinedo, and Yen, 1997).

The AA concept goes one step beyond distributed control. Sequencing rules (Morton, and Pentico, 1993) are widely used in the manufacturing practice, due to their simplicity and relative efficiency. However, a traditional shop floor driven by sequencing rules cannot be considered an AA structure, for two main reasons. First, the workload of a machining center is typically assigned by an external dispatcher, and usually machining centers cannot react to it (Talavage, and Hannam, 1988). Second, no real cooperation or negotiation takes place among the agents. The AA structure calls for a different idea of establishing the workload of the centers. This assignment may be the result of a *bargaining* phase in which the task to be accomplished plays the role of the client asking for a service and the agents dispute among them the right (or the duty) to serve the client.

Agents selection is not a simple mechanical process, since it typically requires insights that can be achieved only through a thorough understanding of the system (Bongaerts, et al., 1996). Of course there is not a unique *right* way of designing and building Autonomous Agent Systems (AAS). Actually, the agent selection phase may be easy, whereas the hard part of the work is to ensure effective communication between agents without destroying their autonomy (Muller, 1997).

This paper presents an approach to design, not a complete design method. The approach presented here can be used as the basis of different formalizations. AAS development may be viewed as an iterative and incremental process. At each step of this process, the main aspects of AAS development (analysis, implementation, evaluation, testing, documentation and management) are refined and specified in more detail.

Section 2 presents the five AA paradigms proposed. These paradigms differ from each other for the amount and complexity of information exchanged, negotiation mechanism, implementation requirements and algorithmic aspects. These elements allow to compare systems having different degrees of autonomy (Adacher, et al., 1999).

Section 3 describes data structures and implementation of a simulation system that have been carried out to evaluate our paradigms. An outline of the Unified Modeling Language

(UML) conceptual model of proposed paradigms is shown. The aim is to describe an architectural framework emphasizing the role of object-oriented modeling in ensuring modularity and reusability and introduce the features of the UML, which is the emerging standard for object-oriented conceptual modeling (Douglass, 1998; Booch, 1994; Quatrani, 1998).
An increasing number of software tools for modeling and designing intelligent agents and multi-agent systems is now available (Norman, et al., 1997; Jennings, and Wooldridge, 1998). These tools exploit the analogy between the notion of autonomy (in AAS) and that of encapsulation in object oriented programming (OOP) (Barber, 1998; Barber, et al., 1998).
In manufacturing, autonomous agents can be naturally identified with the subjects that physically interact with the surrounding environment through sensors and effectors, similarly to the definition given by Russell and Norvig (1995). As a consequence, a key aspect of the implementation of the autonomous agents concept is the trade-off between the amount and the detail of the information available to the agent and the quality of its decisions.
In fact, in principle each agent can be provided with a large amount of information concerning the status of the overall system, but this may be physically infeasible. For instance, consider the machines and the parts as agents in a shop floor. Upon completion of a task, a part may decide on which machining center should it be processed next. In principle, to select the best decision, several elements should be considered, including the characteristics of the parts scheduled on a certain machine, the processing requirements of these parts, their subsequent routing, machine speed and tooling etc.
Conveying such detailed information to each part may be impossible, because of communication and computational overhead. This is in fact a major problem in many centralized architectures. On the other hand, a careful choice of the actually relevant pieces of information may point out that the agents can still work satisfactorily, even without a full knowledge of the whole system.
Most of the dispatching rules commonly used in manufacturing only make sense in a centralized environment, in which a detailed monitoring of the overall system evolution is available (Pinedo, and Yen, 1997). The paradigms presented in this paper focus on two peculiar features of AAS, namely (i) the limited availability of data concerning the rest of the system and (ii) the concept of negotiation, which allows the agents to react to locally unprofitable situations.

2 AA-Based FMS: paradigms implementation

In this paper three types of agents are considered, one related to machines (*Machine Agent*, MA), one to part storage (*Storage Agent*, SA) and the other related to parts (*Part Agent*, PA). Five different implementations of routing and scheduling mechanisms in an autonomous agents framework are presented. For each of them, the interaction among agents when a Part Agent requires to be processed are described. This occurs when a part enters the system, completes an operation or when a machine breaks down.

2.1 Simple_Traffic_Light (STL).

On the basis of its actual workload, each MA declares its own availability to process further parts, by displaying a red (*busy*) or green (*free*) signal. The part visits the machines one at a time, starting from the closest one. The first encountered machine showing a green signal will take care of the part. The relevant information for routing (green or red signal) only flows from the machine to the part. In this case, no negotiation takes place.
This is the simplest operating mode. Each machine displays tooling information, that can be read by any PA. A MA accepts a new part (green signal) if and only if there is room in the input buffer of the machine. Obviously, this paradigm only applies when buffer size is limited.

2.2 Conditional_Traffic_Light (CTL).

This paradigm is like the previous one, except that the PA provides part information (e.g. processing time, due date etc.) to the MA, and the MA replies by a green or red signal on the basis of the part characteristics. As before, the first encountered machine showing a green signal will process the part. Hence, the information flows bidirectionally, first from the part to the machine and then viceversa. This bidirectional flow can be viewed as a form of negotiation, in which the MA has the final word.
Two different implementations of this paradigm are considered. The first is denoted as CTLw (CTL-workload). Let W be the current workload of a machine, i.e., the total workload of the parts actually in the input buffer of the machine. In the second implementation CTLt (CTL-tardiness), the acceptance decision is made on the basis of tardiness considerations. Precisely, a machine accepts a part if this does not increase the estimated total tardiness of the parts currently in its input buffer (computed by arranging the parts in decreasing order of slack time). Otherwise, the machine will display a red signal.

2.3 Conditional_Traffic_Light&Exchange (CTLE).

The same as CTL, but in this case the machine agent has a further negotiation option, consisting in possibly trading one of the parts in its buffer for the new part. The rejected part will have to start again looking for a machine.
The same as CTLw and CTLt, but this time, when a part is rejected by all machines, the part will start again asking for service. At this point, each machine will compare the maximum slack time of the parts in its buffer (i.e., the slack time of the least urgent part) with the slack time of the part asking for service. If the latter is smaller, the machine exchanges the least urgent part for the new part. Otherwise, the part waits in the Unfinished Parts Storage buffer for its turn.

2.4 Part_Machine_Contracting (PMC).

The PA provides its information to all machines, and each machine elaborates its own price, which is a local surrogate for global performance indices. Such price depends on both the part characteristics and the machine status (workload, tooling, efficiency etc.). Hence, the part selects the machine offering the best price. The information flows bidirectionally between the part and each machine. In this negotiation mode the final decision is left with the part.

The PA provides its information to all machines, and each machine elaborates its own price. Such price depends on several parameters. Let T denote an estimate of the total tardiness of the batches to which the parts in its buffer belong to. If the new part is accepted, this may increase the tardiness, and we let ΔT denote such difference (possibly zero). After querying each MA, the PA selects the MA offering the lowest price.
The price is obtained as a linear combination of these four indices (in decreasing order of importance): ΔT; the number of parts in the input buffer; a variable which is 1 if the machine is currently working on some part and 0 otherwise; a variable which is 1 if a part is presently on its way to the machine and 0 otherwise.

2.5 Part_Machine_Contracting&Exchange (PMCE).

In this paradigm, the PA may decide to select no machine, if all the prices are too high. In this case, the MAs may be asked to elaborate a new price, allowing to give away one of the parts currently in the buffer (as in CTLE).

3 Conceptual Modeling of Autonomous Agents in the simulation framework.

The UML is meant to be applicable to the modeling of all types of systems, it applies well to standard software applications, real-time systems and also to AAS development. Hence, the proposed paradigms may be conveniently illustrated using a visual modeling tool based on UML (Unified Modeling Language) (Douglass, 1998).
The documentation partially reported in this section refers to the FMS simulator development. UML defines a visual notation based on various types of diagrams (i.e. *Class Diagram, Use-Case Diagram, Sequence Diagram, Collaboration Diagram, Activity Diagram and State Diagram*). A diagram is a particular visualization of certain kinds elements from a model and generally deals only a subset of the detailed information about those elements and a given model element might appear on several diagrams.

- *Class Diagram* shows the important abstractions in a system and their relationships. Hence, the primary visual elements in a class diagram are class icons and relationship icons. Each class are represented in the UML as a rectangle with three compartments. The first compartment is for the name of the class; the second and third compartment are used to list the attributes and methods defined by the class. Most classes (representing agents) in a system will be related to other classes so that their corresponding agents can collaborate to accomplish more complex activities. So in addition to classes, attributes, and methods, class diagrams also show relationships that exist among dependent classes. The UML notation distinguishes between different types of relationships, each with a special icon and associates meanings (i.e. uni/bi-directional association, dependency, aggregation, inheritance)

- *Use-Case Diagram* is a natural high level description of how a system will be used and provides a view of the intended functionality of the system that is understandable by developers and customers. A single use-case diagram represents many related yet distinctly scenarios (particulars paths through the system functionality).

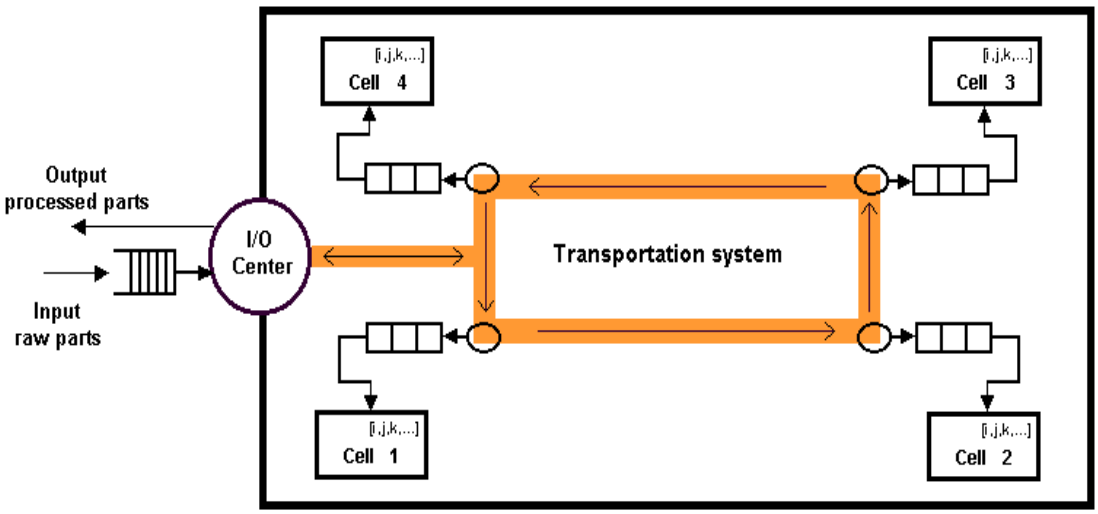

Figure 1: Layout of the simulated FMS.

- *Sequence Diagram* provides notation for modeling scenarios. Sequence diagrams use object icons and vertical time-lines projected downward on the diagram (the time flows from the top of the diagram). Messages passed between objects are represented by horizontal lines. Scripts may explain one or more messages and, for real-time designs, exact timing often must be specified.

- *Collaboration Diagram* is another notation for modeling scenarios. The sequence information (i.e. the order in which the messages are sent) is associated to messages by a progressive number. In this diagrams the structure is more prominent, but the event sequence are more obvious in the sequence diagrams.

- *State Diagram* shows the sequences of states for a reactive class or interaction during its life in response to some signal, together with its responses and actions. It provides a visual model of states (i.e. nested, sequential and complex states), transitions and events.

- *Activity Diagram* is a specialized form of state diagram in which most of all transitions are taken when the state activity is completed.

The development and implementation of the simulation framework as an iterative and incremental process (from simplest paradigm STL) promotes creativity and, in the same time, it must be controlled and measured to ensure high quality in the proposed solutions. Each iteration consists of different components: requirements capture, analysis, design, implementation and test. Not all requirements are known at the beginning of the life cycle but they may change dynamically throughout all phases.
Of course, each of these paradigms may be implemented in different ways depending on the real system characteristics.
The experimental environment is a sample Flexible Manufacturing System (FMS) with four machining centers connected by means of an automated material handling system. In

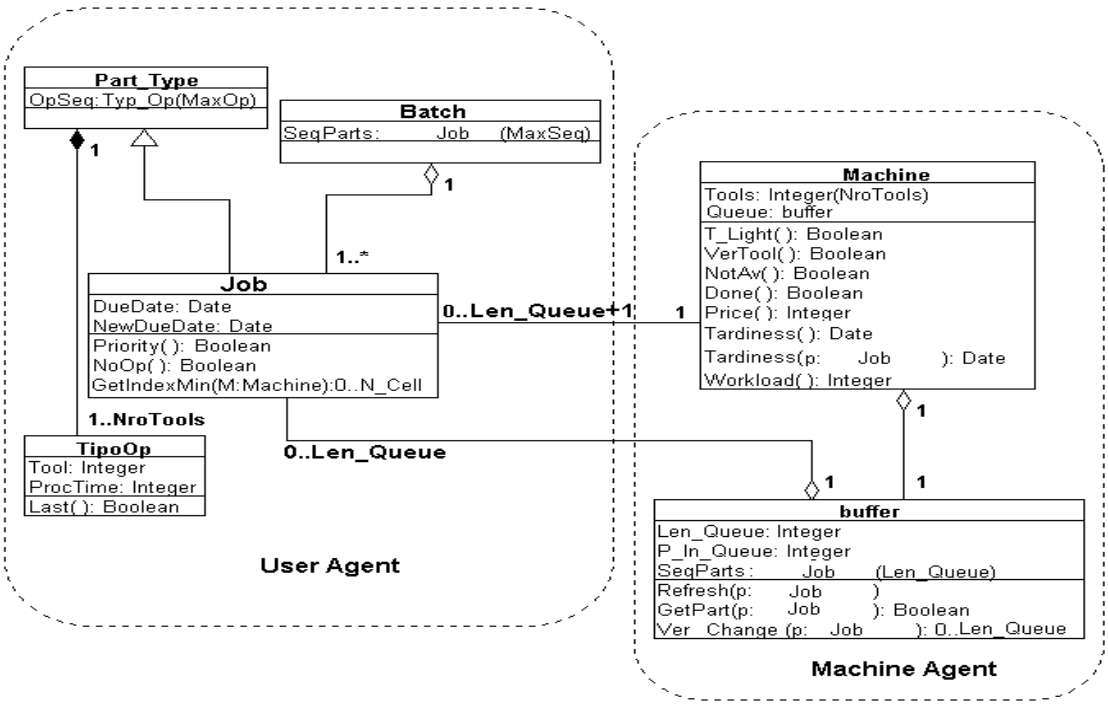

Figure 2: UML Class Diagram for the FMS simulator

a centralized control architecture, the controller (on the basis of the system status and the production plan) makes decisions and sends commands to each single system component. The large number of decisions, data and possible system states makes the development of a centralized controller a very complex task. The adoption of an autonomous agents-based architecture may avoid some of these problems (Shaw, 1989; Smith, 1980).

The five paradigms proposed in the previous section have been implemented and evaluated in a computer simulation prototype (Adacher, et al., 1999). The system consists of four machining centers (each with its own input buffer), a transportation system and an I/O center, as depicted in figure 1.

The I/O Center loads raw parts (from a raw materials storage unit), unloads finished parts, and handles unfinished parts. A Storage Agent (SA) is associated with the I/O Center, letting the raw parts in the system in EDD (earliest due date) order. Others SAs are associated with the Finished Parts, Raw Materials and Unfinished Parts. Each machining center is associated with a Machine Agent (MA). Each machine has its own tools and may therefore perform a set of operations. The model refers to a time interval during which tooling does not change. Note that each operation may be executed by different centers.

The informative unit of a MA takes care of part sequencing and the computation of prices. Machine Agents do not directly negotiate work with the other MAs.

A Part Agent (PA) is created when a new physical part is released in the system. The PA retains both the physical and informative characteristics of the part, as long as the part is in the system. The informative unit of a PA keeps track of the actual status of the part. Part Agents may occasionally broadcast a limited amount of information to other PAs.

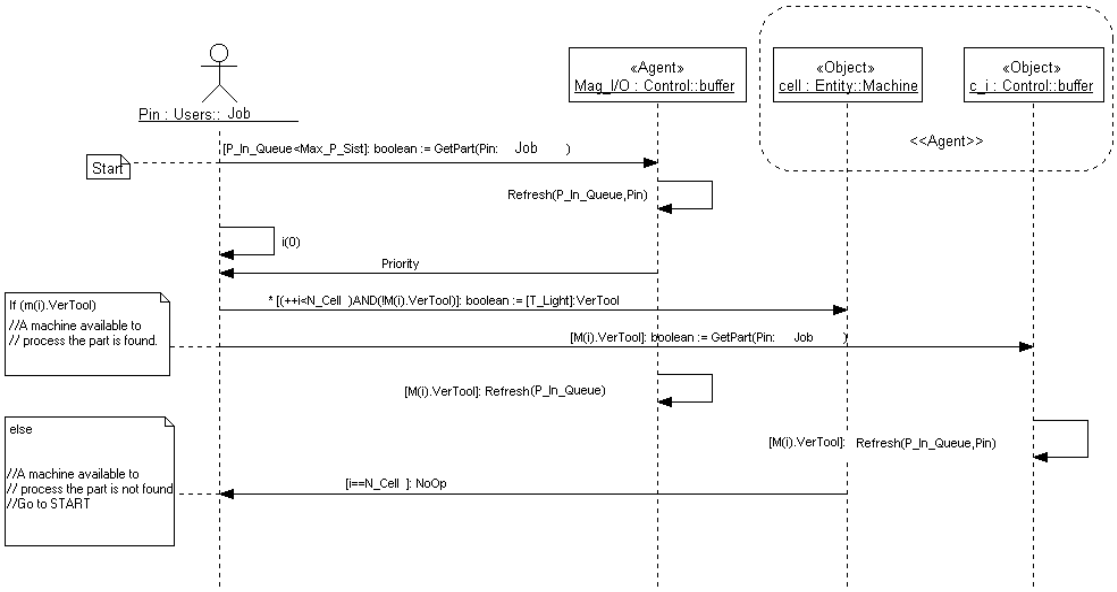

Figure 3: UML Sequence Diagram, a scenario from STL paradigm

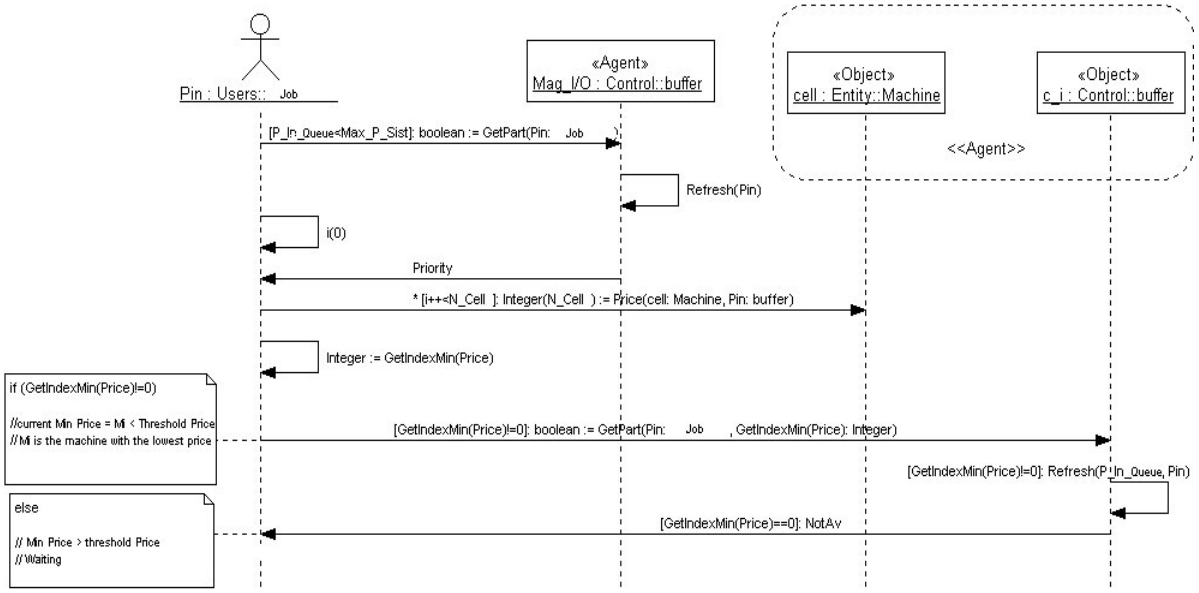

Figure 4: UML Sequence Diagram, a scenario from PMC paradigm

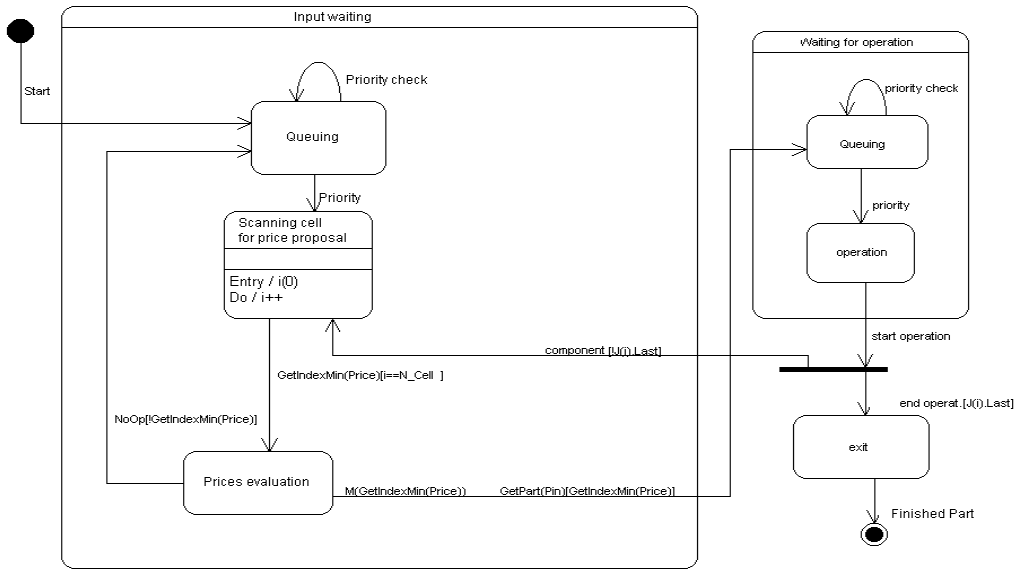

Figure 5: UML State Diagram, the PMC paradigm

When the part completes the last operation and leaves the system, a record with the relevant part information is passed onto a monitoring system, and the PA is destroyed. Unlike MAs, PAs are supposed to be unable to communicate among themselves. This is mandatory, in order to keep the information overhead sufficiently low. Since the analysis is focused on the coordination of agents, part transfer is supposed not to be a critical issue in this model, i.e., transportation facilities (modeled by a unidirectional loop) are supposed always available when needed.

In figures 2 through 6, UML diagrams associated with the proposed paradigms are depicted. These diagrams do not contain all implementation details, they simply mean to sketch the information exchange among agents and their implementative structure.

In figure 2 is depicted the Class Diagram with objects and relationships adopted in the simulation environment.

In the Sequence Diagrams (figures 3 and 4) vertical lines indicate temporal axes (events occur from top to bottom of the diagram), horizontal arrows express interaction between agents over time (e.g. processing request and data exchange). Self-loops indicate individual processing activities (e.g. price computation). Phases marked with an asterisk are repeated as long as the condition in square braces holds. Figures 3 and 4, corresponding to STL and PMC respectively represent a particular scenario refers to a part entering the system and searching for a machine to perform the first operation. For the subsequent operations, the diagrams are almost identical. The further negotiation step may occur with CTLE and PMCE paradigms.

Examples of State and Activity diagrams are shown in figures 5 and 6 for the PMC paradigm and for the same previous scenario.

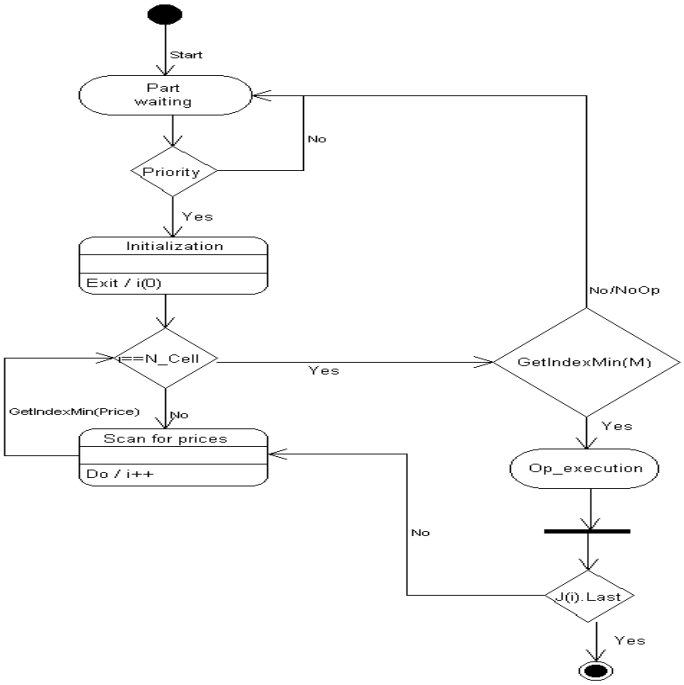

Figure 6: UML Activity Diagram, the PMC paradigm

4 Conclusions

Object orientation is playing an increasing role in the development of many real-time software systems. It is been recognized for some time that modeling is one of the important ingredients contributing to a successful development project.
The unified modeling language (UML) has become increasingly significant as a means of modeling software systems. UML is seen as the principal OO modeling language by many developers and vendors. It is being widely applied to the practice of software development in a variety of industries and for differing applications. Its development continues to evolve.
Real-time manufacturing systems are often complex systems. Developing software for real-time systems is a complex process. Modeling provides a mechanism that helps developers to understand the inherent complexity of real-time systems and to design relevant and successful solutions to real-time problems.
In this context visual modeling laguages, and software tools that supporting them may play an important role. This paper presents important features of object-oriented design tools focusing on their relevance in modeling and developing software for Autonomous Agents Manufacturing Systems.

References

Adacher, L., A. Agnetis and C. Meloni (1999). Autonomous Agents Architectures and Algorithms in Flexible Manufacturing Systems. Research report 43-99. Dipartimento di Informatica e Automazione, Università di Roma Tre.

Adiga, S. (1989). Software modeling of manufacturing systems: a case for an object-oriented programming approach. Annals of Operations Research, 17, 363-378.

Barber, K.S. (1998). The Architecture for Sensible Agents, Proceedings of the Conference on Autonomous Agents, Minneapolis, USA, April 1998.

Barber, K.S., E. White, A. Goel, D. Han, J. Kim, H. Li, T.H. Liu, C.E. Martin and R. McKay (1998). Sensible Agent Problem-Solving Simulation for Manufacturing Environments. Proceedings of the AAAI's SIGMAN workshop on Artificial Intelligence and Manufacturing. September 1998, Albuquerque, NM.

Blazewicz, J., K.M. Ecker, E. Pesch, G. Schmidt and J. Weglarz (1996). Scheduling Computer and Manufacturing Processes. Springer–Verlag, Berlin.

Bongaerts, L., J. Wyns, J. Detand, H. Van Brussel and P. Valckenaers (1996). Identification of Manufacturing Holons. Preprints of European Workshop on Agent-Oriented Systems in Manufacturing. Berlin.

Booch, G. (1994). Object-Oriented Analysis and Design with Applications. Addison-Wesley, New York.

Douglass, B.P. (1998). Real-Time UML. Developing efficient Objects for Embedded Systems. Addison-Wesley. Reading.

Jennings, N.R. and M.J. Wooldridge (Eds.) (1998). Agent Technology Foundations, Applications, and Markets. Springer-Verlag, Berlin.

Lin, G.Y. and J.J. Solberg (1992). Integrated shop floor control using autonomous agents, IIE Transactions, 24, 3, 57-71.

Morton, T.E. and D.W. Pentico (1993). Heuristic Scheduling Systems. John Wiley and Sons, New York.

Muller, J.P. (1997). Control Architectures for Autonomous and Interacting Agents: A Survey. In: Lecture notes in artificial intelligence, Lecture notes in computer science, 1209, (Cavedon, L., A. Rao and W. Wobcke (Eds.)) Springer-Verlag, Berlin.

Norman, T.J., N.R. Jennings, P. Faratin and E.H. Mamdani (1997). Designing and Implementing a Multi-Agent Architecture for Business Process Management. In: Lecture notes in artificial intelligence, Lecture notes in computer science, 1193 (Muller J.P., Wooldridge M.J., Jennings N.R. (Eds.)). Springer-Verlag, Berlin.

Pinedo, M. and B.P.-C. Yen (1997). On the design and development of object-oriented scheduling systems. Annals of Operations Research, 70, 359-378.

Quatrani, T. (1998). Visual Modeling with Rational Rose and UML. Addison-Wesley, Reading.

Russell, S.J. and P. Norvig (1995). Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs.

Shaw, M.J. (1989). FMS scheduling as co-operative problem solving. Annals of Operations Research, 17, 326-346.

Smith, R.G. (1980). The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. IEEE treansaction on computers, 12, 1104-1113.

Talavage, J. and R.G. Hannam (1988). Flexible manufacturing systems in practice. Applications, design and simulation. M.Dekker, New York.

Uzsoy, R. and I. Ovacik (1997). Decomposition Methods for Complex Factory Scheduling Problems. Kluwer Academic Publishing, Amsterdam.