Pergamon

Robotics & Computer-Integrated Manufacturing, Vol. 11, No. 1, pp. 41–50, 1994

0736–5845/94 $7.00 + 0.00

0736–5845(94)E0011–V

● *Paper*

USING PETRI NET MODELS AT THE COORDINATION LEVEL FOR MANUFACTURING SYSTEMS CONTROL

J. L. VILLARROEL and P. R. MURO-MEDRANO

Departamento de Ingeniería Eléctrica e Informática, University of Zaragoza, María de Luna 3, Zaragoza 50015, Spain

This paper focuses on the coordination of manufacturing systems. A software architecture for manufacturing systems control is presented. The architecture follows a hierarchical approach for control, ranging from real-time coordination to long-term scheduling, where the different layers share a common knowledge base. A knowledge representation schema for manufacturing systems is proposed. This schema integrates knowledge representation techniques based on frames and high-level Petri nets to describe plant behaviour, and it follows an object-based methodology. The coordination function is materialized by making the coordination model evolve; this is done by a centralized, concurrent and interpreted implementation of the underlying Petri net.

1. INTRODUCTION

A manufacturing system pursues the objective of manufacturing products to the satisfaction of the client (which is basically measured by the product's quality and delivery date) while making a profit for the manufacturer. To achieve these objectives, the manufacturing system must possess a considerable degree of automation and flexibility. This requires an integrated computer control system where issues such as hierarchy, complexity, concurrency, optimization, capacity, uncertainty and feedback become very important. Many researchers have agreed on the hierarchical decomposition of manufacturing systems control. Four levels have been considered in the hierarchical decomposition adopted in our work: *production planning*, *operation scheduling*, *coordination of plant elements* and *local control*.

In this paper, our attention is focused on the coordination level. To carry out the different coordination functions, a precise model of the manufacturing system behaviour is needed. The number of manufacturing system elements, and the high degree of concurrency between them, make the coordination model very complex. In our approach, factory behaviour specifications are modelled by Petri nets, whereas the interpretation of the Petri net model provides the guidance needed to carry out the coordination task. The use of Petri nets to model discrete event systems, and manufacturing systems in particular, has well-known advantages that are extensively explained in the related technical literature (see Ref. 9 or 19, for example). Petri nets are a formalism that can be used to enforce the precision (absence of ambiguities) of models, and to allow qualitative and quantitative analysis and simulation. Moreover, Petri nets constitute an executable formalism that allows one to use the same model for analysis and for real control of the system. This fact avoids errors in the control system implementation and facilitates flexibility. The use of Petri nets and high-level Petri nets (HLPNs) for the coordination functions has received a lot of attention in the technical literature in recent years. We review two relevant approaches that are related to the work presented here.

References 4, 17 and 20 propose the use of tools from the Petri net family for the modelling, analysis and implementation of the coordination and monitoring functions. The Petri net expression power is expanded in Ref. 21 to represent state- and time-related knowledge that is not known with certainty (using fuzzy-timed Petri nets). Abnormal events and normal operations can also be represented with the same representation schema.

CASPAIM[6,8] represents another important approach, where Petri nets are used for the design of control systems in manufacturing applications. The model used in CASPAIM is based on adaptive and structured Petri nets. The coordination function is distributed among a set of plant controllers, which can communicate directly or through a supervisor. Coordination decision problems are solved by a higher control level that is itself implemented using rule-based techniques. A special interface is used by the Petri net model to communicate with this decision system.

Previous work to this paper is presented in Refs 1, 11 and 22, where coloured Petri nets are proposed for the design, analysis and specification of the coordination function. On the other hand, in Refs 12, 13 and 15 artificial intelligence techniques are used for the decision-making process in higher control levels.

Finally, the basis of this work has been presented in Ref. 23.

This paper is organized as follows. Section 2 provides an overview of the general system architecture, which addresses all control levels identified above. HLPNs are proposed in Sections 3 and 4 as a mechanism for the design and implementation of the coordination model, while the internal structure of the coordinator subsystem is illustrated in Section 5. The coordinator has specialized modules to deal with the control of the manufacturing system in its normal evolution as well as in exceptional situations. The coordination of a simple manufacturing cell is developed as an illustrative example in Sections 3 and 6.

2. CONTROL SYSTEM ARCHITECTURE

The functionality of the global control system (planning, scheduling and coordination) can be implemented in two different ways: (1) a single software module integrating the overall functionality, or (2) a set of communicating software modules. The first approach produces a more compact software. However, the second approach has been selected here for flexibility and modularity. With this option each function can be designed and implemented separately with specific techniques and can therefore the more efficient.

Figure 1 outlines the main components of the global control system architecture adopted in our approach. The architecture is structured in four subsystems sharing a common knowledge base. The basic functions of the *coordination system* are the coordination and monitoring of plant elements. In this framework, the coordinator responsibility is to manage and supervise the execution of local controllers' tasks. The coordinator evaluates which operations can start at any given time and which of them can do it simultaneously. Plant monitoring (data collection and exception handling) is carried out by an internal monitoring module.

The *dispatching subsystem* is responsible for providing solutions to the decision problems in real time. The coordinator sends messages posing decision problems and the dispatcher responds with a specific solution. The dispatcher makes final real-time decisions and solves discrepancies between the system status and the schedule by means of a flexible interpretation of the schedule (a similar approach is considered in Ref. 20).

The *scheduling subsystem* is responsible for generating the schedule of operations for each period of production time using the available information about orders, inventory, plant characteristics and resources, production objectives, etc. We also assume an ability to revise incrementally the current schedule in situations where significant discrepancies between the schedule and the current factory state are detected.

The main function of the *business planning subsystem* is the production planning. From the client's orders, the state of warehouses and the manufacturing capacity of the plant, a list of production orders is generated for each period.

Finally, the *global knowledge base* is a common store of the system declarative knowledge (factory characteristics and status, production plans, schedules, constraints, heuristics). This information is shared by the four subsystems of the control architecture but each subsystem has its particular vision.

3. KNOWLEDGE REPRESENTATION SCHEMA

An important step in the development of the control system presented here was the definition of a knowledge representation schema for CIM to be used for a variety of applications, ranging from plant coordination to production planning. Production goals, current production schedule, system status and system model are the main informations to be stored in the common knowledge base. The representation of manufacturing entities such as production orders, manufacturing operations, products, materials, manu-

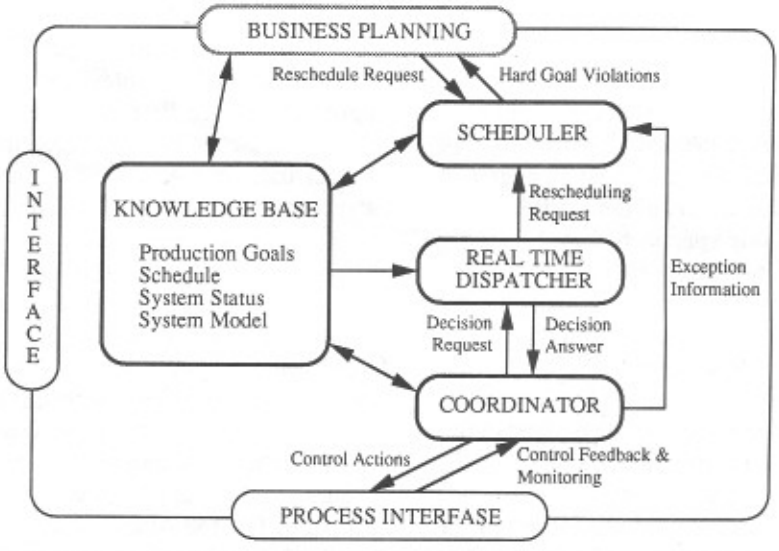

Fig. 1. Control system global architecture.

facturing resources, transports, stores, ... and their relationships must be adequately supported. There are many nontrivial concepts and a complicated relationship structure. To deal with this representation problem a knowledge representation schema, called KRON (knowledge representation oriented nets) has been adopted.[15,22] KRON is based on the following main paradigms:

- *Frames:* Frame-related concepts were used as a basis to support the representation of knowledge due to their expression power to represent concepts and theirs relationships.
- *Object-based methodology:* This approach offers advantages such as:[5] (a) supports conceptual models near to human thinking and is implementation-independent (thus the models are easy to understand), and (b) facilitates reusability and model extensibility based on encapsulation and inheritance characteristics.
- *HLPNs:* A manufacturing system model to be used in a control application must represent the knowledge about the dynamic behaviour and system status. Manufacturing systems are nontrivial discrete event systems with a complicated structure of concurrency. It is also well known that HLPNs are well-suited tools for the modelling of complex discrete dynamic systems. Additionally, HLPNs allow the formal analysis, graphic representations, simulation and efficient execution of models.

In KRON, a model is composed of objects, which can contain data, procedures, relations and meta-knowledge. In addition to these features generally supported by object-oriented languages, a set of semantic primitives implementing the HLPN formalism is included. The HLPNs provide the mechanism to describe the internal behaviour of dynamic objects and the interactions between them. This paper will not present the HLPN formalism (the reader is referred to Ref. 10).

There are some knowledge representation schemata similar to our approach. The object Petri nets[18] are the most referenced schema in the related technical literature: a system model is composed of a set of related objects (which define the HLPN marking) and an HLPN specifying the system global evolution. In this approach, the HLPNs are not fully integrated in the object-oriented environment. Another interesting approach is the one proposed in Ref. 2, where the internal behaviour of objects and their interaction are specified by HLPNs. However, this approach does not have the power of frame-based schemata to represent relations (different from dynamic interactions) between objects. Finally, Ref. 14 proposes another interesting frame and HLPN-based representation schema, but the object-oriented advantages are missing.

We will illustrate the proposed representation schema by creating the model to represent a typical manufacturing machine, which will be called a **transformation-machine**. A **transformation-machine** model can be represented as an object (frame) containing information about its attributes, constitutive parts and structure. Characteristics such as abstraction hierarchy relations, predicted information and data collection, physical characteristics, interface features and graphic representation characteristics are also represented within the object definition.

In addition to the previous features, the dynamic behaviour must be represented. Figure 2 shows the HLPN graphical specifications for the dynamic characteristics of an isolated **transformation-machine** prototype. The behaviour of this physical entity is defined such that it can load parts to be processed and it has a limited available capacity.

Three places (**available-capacity**, **loaded** and **unload-ready**) and three transitions (**load–M**, **process–M** and **unload–M**) are shown in Fig. 2. Places are identified in the **transformation-machine** textual representation (Fig. 3) by state slots and transitions by action slots. The values of state slots identify the marking whereas the values of action slots are pointers to transition objects (objects **load–M**, **process–M** and **unload–M** in Fig. 3). Transitions in the model are associated with system actions used to change the state, which is itself represented by the net marking. Tokens or colours can be identified in this case with products evolving in the production plant. The presence of a token in a place is indicated by the presence of the frame name as a state slot value.

As an example, let us create now the model of a manufacturing work cell. To simplify, we consider a workcell **C1**, with three machines **M1**, **M2** and **M3**, a random access store *ST* and a robot **R** for palleting. First, machines, stores and the transport system of the cell are separately defined. Machines **M1**, **M2** and **M3** can be defined as instances of the **transformation-machine** prototype. Following a similar method, **ST** and **R** must be defined as instances of isolated prototypes modelling a random access store and a robot. The structural description of the workcell is completed by establishing interfaces between machines, the store and the transport system, defining the flow of parts between them. This is done by

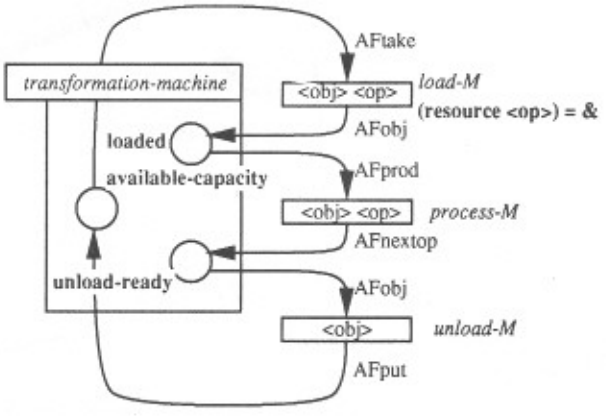

Fig. 2. Underlying Petri net modelling the **transformation-machine** prototype behaviour.

synchronising the appropriate actions (e.g. **load** and **unload** actions of machines are synchronized with the **pick-and-place** action of a robot; the results are the **load–M**$_i$ transitions). Figure 4 shows the HLPN modelling the dynamic behaviour of the complete workcell **C1**.

4. MODELLING PROCESS AND ITS DEVELOPMENT ENVIRONMENT

The design of the control software must be facilitated by a computer-aided development environment. From the knowledge representation point of view, our environment[24] has two layers. The first layer is an analysis and design tool for discrete event systems of general purpose. This layer consists in the lower-level representation schema, the inference mechanism[3] and its conflict resolution module. The second layer is a specialization for the manufacturing domain; it contains useful object hierarchies for processes, operations, parts, machines, transports, manufacturing relations and aggregations, etc.

Model analysis and validation can be performed by mathematical analysis of the underlying Petri net or by simulation. A module is available to extract the generalized Petri net from the model in a format understandable by a generalized Petri net analysis package. This package produces information about: invariants, deadlocks, traps, linear description improvement, structural deadlock-freeness analysis, structural bounds, etc. We have to say at this point that

```
{transformation-machine
    is-a : transformation-resource machine
    states : loaded unload-ready available-capacity
        ; state slots
        loaded :
        unload-ready :
        available-capacity :
    actions : load process unload    ; action slots
        load : load-M
        process : process-M
        unload : unload-M
    ; predicted information and data collection
    pending-operations :
    statistics :
    max-part-size :    ; physical characteristics
    constraints :
    set-up :
    average-processing-time :
    precision-level :    ; abstraction hierarchy relations
    has-resources :
    resource-of :
    type-of-synchro :    ; interface features
    input-synchro :
    output-synchro :
    icon :    ; graphic representation characteristics
    wickname :
    comment :    }
```

```
{load-M
    is-a : action-obj
    pre-net-relations : (& available-capacity AFtake)
    post-net-relations : (& loaded AFobj)
    associated-data : <obj> <op>
    predicate : (resource <op>) = &
    ao-conflict-of :    }

{unload-M
    is-a : action-obj
    pre-net-relations : (& unload-ready AFobj)
    post-net-relations : (& available-capacity AFput)
    associated-data : <obj>
    predicate : t
    ao-conflict-of :    }

{process-M
    is-a : action-obj
    pre-net-relations : (& loaded AFprod)
    post-net-relations : (& unload-ready AFnextop)
    associated-data : <obj> <op>
    predicate : t
    ao-conflict-of :    }

AFobj : (name <obj>)
AFtake : (name one)
AFput : (name one)
AFprod : (name <obj>; operation <op>)
AFnextop : (name <obj>; operation (next <op>))
```

Fig. 3. Composed object modelling the machine prototype (some slots represented here belong to its classes).

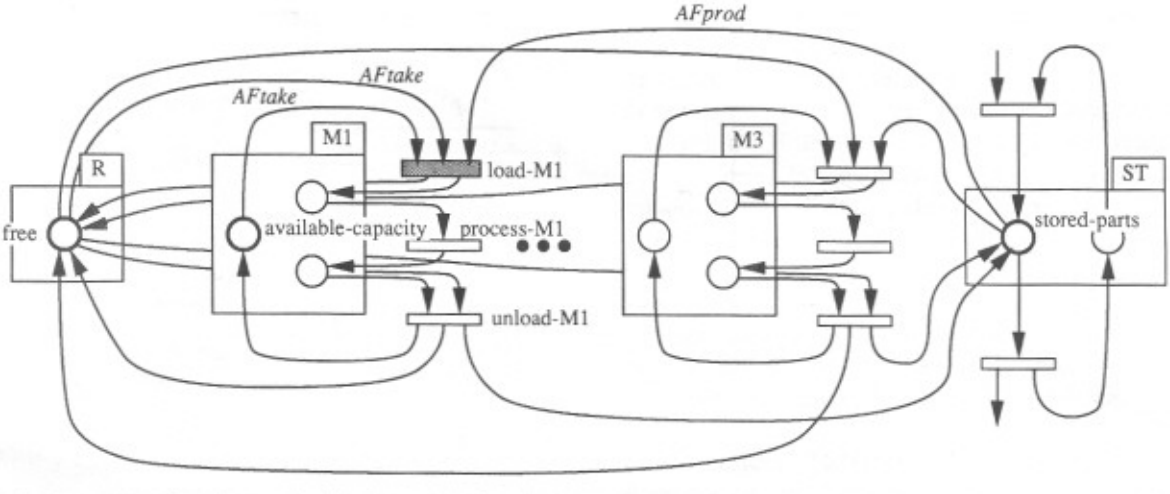

Fig. 4. Structure of Petri net underlying the model of workcell **C1** at the machine level of abstraction.

analysis information usefulness decreases once the model reaches a certain complexity. The model can also be directly used by a simulation module.[16] The simulator has the typical utilities to associate random distributions, create events and for data collection.

Software reliability is facilitated by the methodology: the designer is constrained to work with a high discipline imposed by an underlying discrete event systems formal paradigm (Petri nets) and the use of a well-recognized design paradigm (object-oriented design).

Two graphical interfaces are available in the environment to facilitate the creation of models. The first one facilitates the work at a detailed level; this is a graphic–textual interface used to facilitate the design of the internal part of the objects. The most important internal aspect for control is the one related to the estates, transitions and its pre- and post-conditions; the designer has visual access to the graphic features of the Petri net. The designer has access to the object hierarchies and their methods. On the other hand, the tool allows the creation and modification of nets, the establishment of synchronization relations between transitions, the animation of model evolution, the control of that evolution by the user, etc.

The second one is devoted to the manufacturing engineer. This is an expert in the manufacturing domain but not necessarily an expert in Petri nets or object-oriented design. This higher-level graphic tool is specialized in manufacturing system while it hides model internal features. It allows the graphic manipulation of MIKRON objects and primitives, and it becomes the graphical support for the animated simulation. In addition to icon manipulation, it provides specialized windows for textual visualization and manipulation of objects, and access to the relevant object hierarchies.

5. COORDINATOR

Coordination and *monitoring* of plant elements are the basic functions of the *coordinator subsystem*. To achieve these goals, the coordinator communicates with local controllers through a set of signals:

- *Controller signals*. The coordinator sends *orders* to the controllers for the execution of activities. When the activities are successfully finished *activity-end* messages are sent in return. *Exception* messages are also sent to the coordinator if any error or exception arises during an order execution.
- *Sensor signals*. Set of incoming signals from sensors complementing coordinator information about plant evolution.
- *Alarm signals*. Alarm signals (such as machine and part breakdowns, abnormal stoppages) can be sent to the coordinator by operators or special supervisor systems.

The coordination method is based on the *interpretation* of the dynamic behaviour specifications represented in the system model. The coordination function is materialized by making the coordination model evolve, which in turn makes the production system evolve. This is a closed-loop control where the actions are the orders sent to the controllers and the feedback is the composition of the activity-end, exception, sensor and alarm signals. Decision problems, which are not solved by the coordinator, can appear during the evolution of the coordination model. That decision problems are posed to the dispatcher, which is responsible for its solution. Plant *monitoring function* consists in two activities: data collection and exception handling. Figure 5 illustrates the internal architecture of the coordination subsystem and its main modules are explained in the following paragraphs.

The proposed working architecture allows the use of

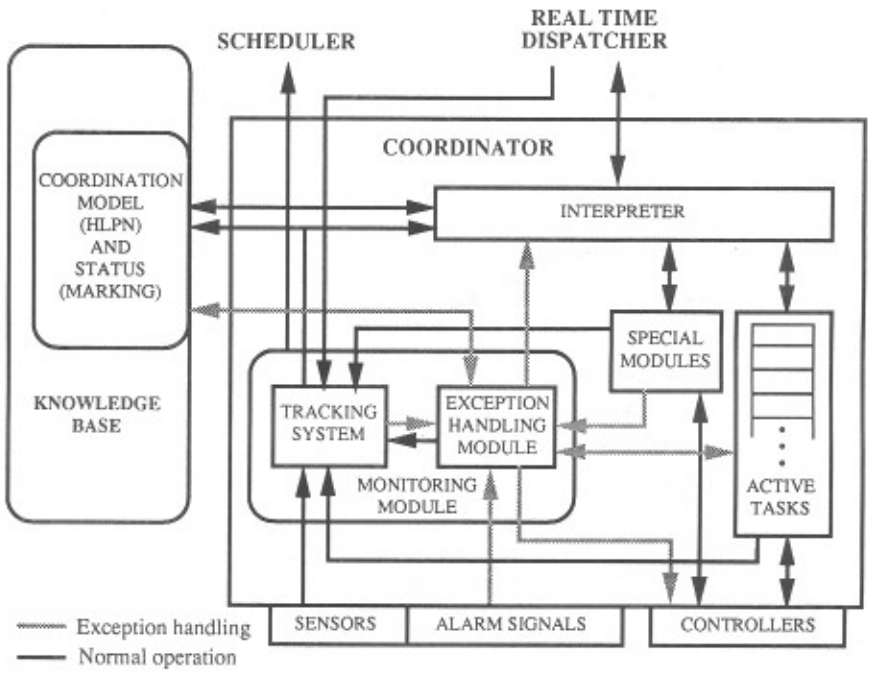

Fig. 5. Coordinator subsystem.

the same KRON model during system modelling, simulation and real system control phases. This approach avoids errors and decreases costs in the implementation phase, and provides flexibility because control strategies can be easily changed by a model updating.

5.1. *Interpreter*

The *interpreter* subsystem is the main one responsible for the coordination function. The interpreter is a centralized, concurrent and interpreted implementation of an HLPN,[7] that is, a module that materializes the coordination model evolution by interpreting a data structure. The interpreter is commonly called the *token player* and its main functions are:

- *Enabling and firing analysis*. The evolution of a Petri net model is carried out by transition firing. Moreover, in a HLPN model each transition has several *firing modes*; this means several firing possibilities depending on the current marking. The interpreter establishes the set of enabled firing modes concerning the HLPN state and the external signals. However, due to conflicts and discrepancies with the schedule, only a subset of enabling firing modes will be fired. This filtering operation, ordered by the interpreter, is carried out by the dispatching subsystem.
- *Firing execution*. Transition firing modes represent production system activities. Firing a transition with respect to a firing mode implies the execution of a piece of code that implements the communication protocol between coordinator and controllers. These pieces of code are configured as tasks to allow their concurrent execution. Figure 6 shows a typical example of an Ada task used to implement the code associated with a firing mode (the code is autodocumented).
- *Marking updating*. The tokens involved in a transition firing are retired from the input places when firing starts, whereas the corresponding tokens are deposited in the output places at the end of firing.

The communication between the coordinator and the local controllers is implemented by the following handshake protocol (its graphical illustration can be seen in Fig. 7):

(1) A coordinator task communicates with a controller and sends an order to run a specific control program. If the control program is a Petri net

```
task load_M1 ;
task body load_M1 is
  --
begin
  loop
    -- The Interpreter orders the execution of associated activity to (part, operation)
    -- firing mode of transition load_M1
    interpreter.start(load_M1)(part,operation) ;
    declare  time_out_M1: exception ; time_out_ROBOT: exception ;
    begin
      -- The task sends a begin operation order to M1 and ROBOT controllers
      M1_controller.begin_op (part, operation) ;
      ROBOT_controller.begin_op (part, operation) ; T := clock ;
      -- The task waits for end operation messages from M1 and ROBOT controllers
      select
        ROBOT_controller.end_op (message_ROBOT) ; T_ROBOT := T - clock ;
        select  M1_controller.end_op (message_M1) ;
        or  delay max_processing_time (operation,M1) - T_ROBOT ;
            raise time_out_M1 ;  -- The operation on M1 has exceded the maximum processing time
        end select ;
      or  delay max_processing_time (operation,ROBOT) ;
          raise time_out_ROBOT ; -- The operation on ROBOT has exceded the maximum processing time
      end select ;
      case message(message_M1, message_ROBOT) is
        when end_op_OK =>  tracking_system.end_task (load_M1, part, operation) ;
        when error_M1 => raise exception_M1 ;
        when error_ROBOT => raise exception_ROBOT ;
        when general_failure => raise general_exception ;
      end case ;
      -- The operation has been completed successfully
      interpreter.end_task (load_M1, part, operation) ;
    exception
      -- An exception has occurred in the operation execution
      when time_out_M1 => exception_handling.time_out_exception (M1) ;
      when time_out_ROBOT => exception_handling.time_out_exception (ROBOT) ;
      when exception_M1 => exception_handling.controller_exception (M1) ;
      when exception_ROBOT => exception_handling.controller_exception (ROBOT) ;
      when general_exception => exception_handling.general_exception (load_M1) ;
    end ;
  end loop ;
end ;
```

Fig. 6. Example of Ada task associated with a firing mode.

implementation, the order can be seen as an event that enables the firing of a start transition.

(2) The task waits for the controller answer. The arrival of an *activity-end* message means that the order has been successfully completed. Then, the task reports to the monitoring module the end of the operation, synchronizes with the interpreter (to establish the end of transition firing) and finishes its execution. If the order has been abnormally finished, an error code is received by the task. In this case, the task does not establish the synchronization with the interpreter but it transfers control to the monitoring module to manage the exception.

There is another communication protocol, between interpreter and dispatcher for decision problem solving purposes. These problems, from the coordination point of view, are located in:

(1) *Conflicts*. They represent situations where there exist several excluding firing modes. These problems arise for example when: a product can be loaded in several resources, a resource is required for several operation executions, . . .

(2) *Transitions* modelling operation starts. If all preconditions of an operation are satisfied, then it must be decided when the operation will be executed (immediately or some time in the future). These decisions are generally based on previsions provided by the current production schedule.

Certain elements of the production system may need specific coordination functions performed with techniques others than those based on an HLPN. Thus, for example, it may be important to perform the internal management of an automatic warehouse with a specific control system. All that is required in this case is the construction of the interface between this system and the HLPN interpreter, which will support the dialogue regarding request, input and output of parts from the warehouse.

5.2. *Monitoring module*

Monitoring activities (data collection and exception handling) are realized in the proposed architecture by the *monitoring module*. This module is itself composed of two main components: tracking system and exceptions handling module.

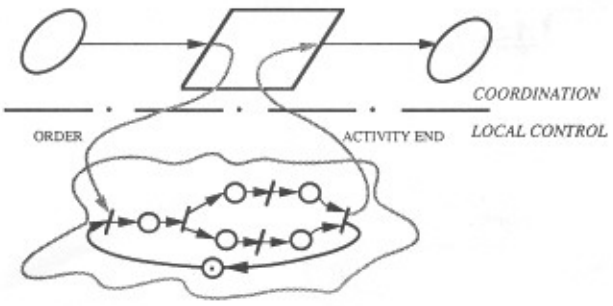

Fig. 7. Communication protocol between coordinator and local controllers.

The *tracking system* receives information from the transition firing tasks, special modules, the dispatcher and, directly, from the plant floor through sensors and other automatic information gathering devices. The tracking system generates statistics and historical data reports from the physical components (e.g. utilization of tools by machines, down times, set up times, etc.), operations (e.g. processing times) and orders (e.g. satisfaction of due dates). It also manages incoming information from sensors and the dispatcher. This information is used by the tracking system to compute the values of events that constitute additional preconditions for transition firings. The information coming from the dispatcher provides the solutions for the decision problems (generally scheduling problems).

The *exception handling module* is in charge of handling the exceptional situations arising in the production plant and it is itself decomposed into three parts: detection, diagnosis and action.

- The *detection* function is distributed by the coordination system modules that have a relation with the production plant (active tasks, special modules, tracking system and exception handling module). For example, a task associated with a firing mode can realize the supervision (e.g. time out as in the example in Fig. 6) of order execution by a local controller.
- *Diagnosis* and *action* functions are centralized in the exceptions handling module. This module receives messages from the coordinator's modules and directly from the plant. Messages describe: machine stoppages, part breakdowns, communication failures, accidents, etc. The exception module will try to recover the error that has arisen or, if it is not possible, to lead the production system to a secure state. The exception module can send emergency stoppage orders to the controllers and abort the execution of tasks associated with transition firing. In any case, this module must guarantee the coherence between model and production system states in exceptional situations, and inform the scheduler of a possible redoing of the operation schedule (see Fig. 9). Reference 3 proposes the use of observers and watch dogs over the net evolution for detection, artificial intelligence and Petri net techniques for diagnosis, and net reconfiguring (change of net structure or marking) for action. This approach for monitoring can be easily integrated in the framework proposed here.

6. COORDINATION EXAMPLE

To illustrate the coordination ideas illustrated above, let us return to the example presented in Section 3. The coordination function is based on the transition firing of the model. For example, transition **load–M1** is modelling on the loading of parts in machine *M*1. To load a part in the machine, several constrains must be satisfied: (a) the robot must be available (a token in place **free** of the robot state object), (b) the part must be

in the store (part marking object must be present in place **stored-parts** of the store state object), and (3) the machine must be free (any token in place **available-capacity** of the machine state object). Moreover, the specified operation on the part must be able to be carried out on the machine [predicate (**resource** <**op**>)=**M1** must be satisfied].

The activity modelled by transition **load–M1** involves the cooperation between two resources, the robot and machine **M1**. For the machine to be loaded, the robot must pick up a part from the store and start the approach to the machine **M1**. Concurrently with that process, machine **M1** tools must be prepared for operation and it subjection clamps be opened. Then, the robot is allowed to locate the part in the clamps. This action is detected by a sensor, which starts the subjection. When the part is grasped, the robot must release it and go to a home state. Figure 8 shows a Petri net representing the described local control function. In the sequel, we suppose that the local control is also implemented by Petri nets.

A transition firing, concerning a firing mode, implies the execution of a task that communicates with the local controllers. Loading part **P1-01** on machine **M1** involves the firing of transition **load–M1** and the execution of the task (see Fig. 6) associated with that firing mode. The interpreter evaluates the enabling of **load–M1** and the dispatcher decides its firing. Then, the interpreter removes the involved tokens from the input places and makes active the task associated with the firing mode [**start(load–M1)** message in Fig. 8]. Orders to the robot and **M1** controllers are sent by the task. These orders enable the firing of **start** transitions. Its own net models are executed by each controller until the transition **end** is reached. At this moment, an end activity message is sent to the coordinator. On

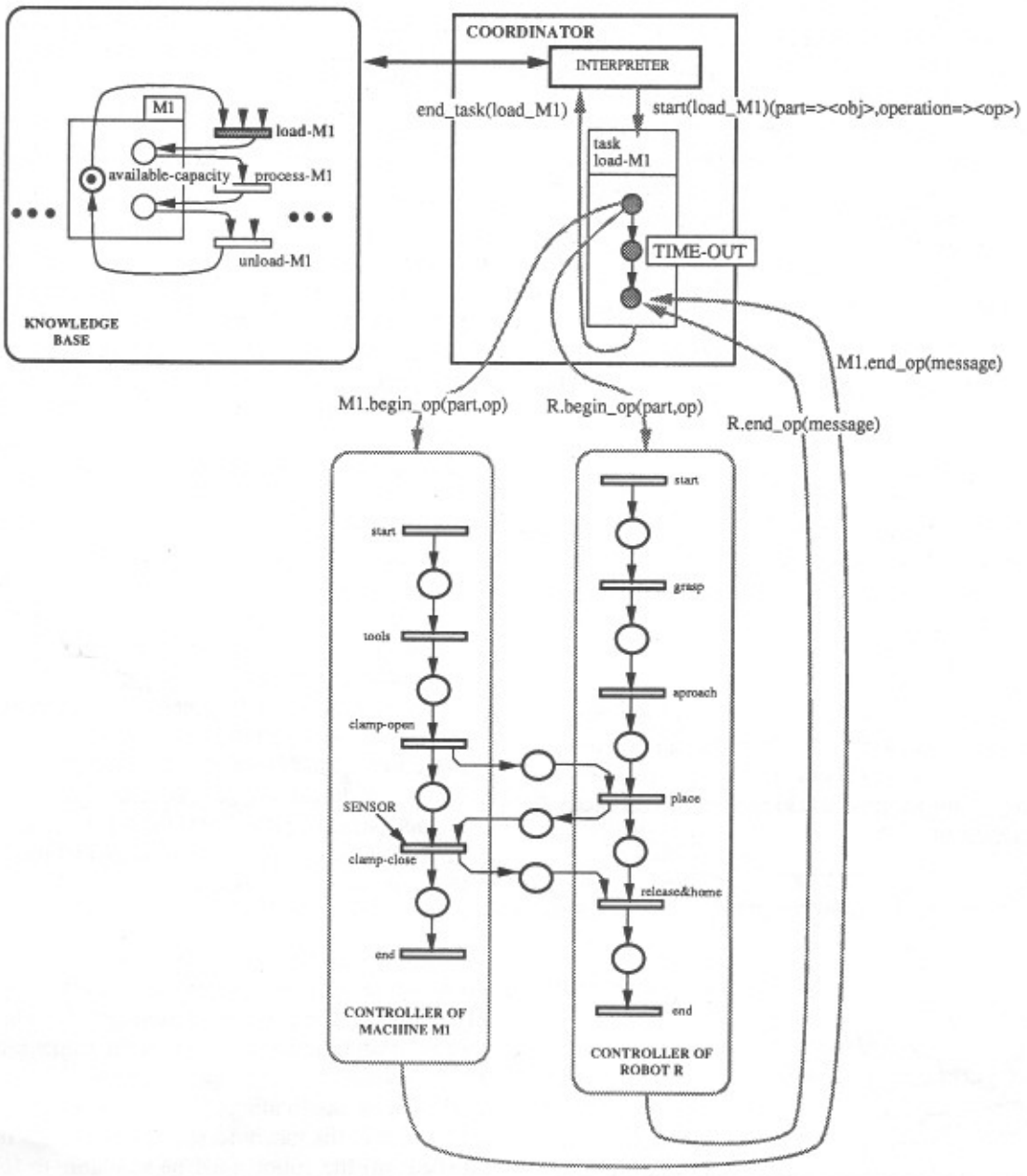

Fig. 8. Firing of transition **load–M1**.

finishing transition **load–M1** firing, a message is send by the task to the interpreter [**end_task(load–M1)** message in Fig. 8] and then the marking is updated: part **P1-01** will be loaded on machine **M1** and the robot will become free.

The exception handling operation will be illustrated using the same example. Let us suppose the following exceptional situation: the robot grasping operation fails and the approach to **M1** is made with no part. Both controllers, from machine and robot, reach a deadlock because no part presence is detected by the robot hand sensor. This fact implies that transition **clamp-close** from **M1** cannot be fired (see Fig. 8). Following a typical monitoring technique, a watchdog can be included in the corresponding active task. When the maximum time of an operation has been reached (*detection*), the associated task communicates the error to the exception handling module and finishes its execution. At that moment the following actions will be carried out by the exception handling module. First, orders will be sent to the local controllers to conduct the resources to a secure state. After that, an alarm signal will be sent to the production plant to notify the operator about the arisen error. This notification is a request for the operator to provide the adequate correcting actions. In this case, the operator must place the part correctly. Additionally, the exception handling module must be notified about the exception origin (*diagnosis*). The next action consists on recovering the previous marking to the exception which represents the cell actual state (robot free, machine **M1** free and part **P1-01** in the store). At this point, the interpreter will try to fire the transition again to continue in normal operation. This process is illustrated by Fig. 9.

7. CONCLUSIONS

This paper deals with the problem of software design for complex manufacturing systems coordination. A control software architecture, covering the planning, scheduling, and coordination functions, has been proposed. The architecture is structured in four software modules sharing a common knowledge base.

A knowledge representation schema has been developed to be used in the global knowledge base. This schema integrates frames, HLPNs and it follows the object-oriented programming paradigm. Manufacturing system's models are rapidly prototyped within this schema. The specification of entities' behaviour, concurrency, entity synchronization, model analysis, . . . are facilitated by the underlying Petri net. This one enforces the precision of models and allows qualitative and quantitative analysis and simulation.

This paper is mainly focused on the coordination and monitoring functions. These functions correspond to the *coordinator* module of the architecture, which is also composed of a set of comunicating software modules. System coordination is materialized by making the Petri net based coordination model evolve, which in turn makes the production system evolve. This is a closed-loop control where the actions are the orders sent to the controllers and the feedback is the composition of the activity end, exception, sensor and alarm signals. The coordinator module allows specific coordinations realized with techniques other than those based on Petri nets. Decision problems are not

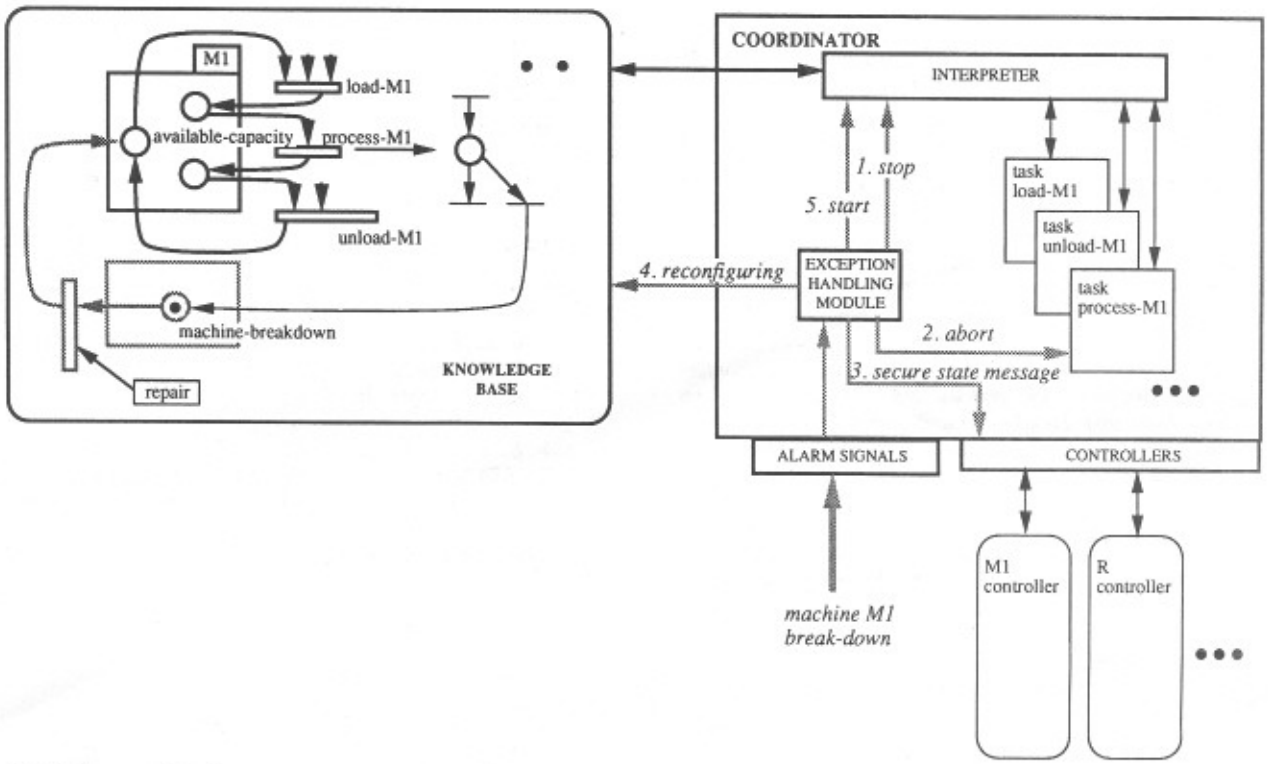

Fig. 9. Abnormal function.

solved by the coordinator but by a specialized module called the *dispatcher*.

The proposed working architecture allows the use of the same KRON model during system modelling, simulation and real system control phases. This approach avoids errors and decreases costs in the implementation phase and provides flexibility because control strategies can be easily changed by updating the model.

REFERENCES

1. Alla, H., Ladet, P., Martínez, J., Silva, M.: Modelling and validation of complex-systems by coloured Petri nets: application to a flexible manufacturing system. In *Advances in Petri Nets* 1984, Number 188 in Lecture Notes in Computer Science. Berlin, Springer Verlag. 1985, pp. 15–31.
2. Baldassari, G., Bruno, G.: An environment for object-oriented conceptual programming based on PROT nets. In *Advances in Petri Nets* 1988, Number 340 in Lecture Notes in Computer Science. Berlin, Springer Verlag. 1988, pp. 1–19.
3. Banares, J. A., Muro, P. R., Villarroel, J. L.: Taking advantages of temporal redundancy in high level Petri nets implementations. In *Application and Theory of Petri Nets* 1993, Ajmone Marsan, M. (Ed.). Number 691 in Lecture Notes in Computer Science. Berlin, Springer Verlag. 1993, pp. 32–48.
4. Ben Ahmed, S., Moalla, M., Courvoisier, M., Valette, R.: Flexible manufacturing production system modelling using object Petri nets and their analysis. In *Proceedings of IMACS Symposium MCTS*. Lille, France. 1991, pp. 553–560.
5. Bruno, G., Balsamo, A.: Petri net-based object-oriented modelling of distributed systems. In *Proceedings of the ACM Object-Oriented Programming Systems, Languages and Applications, OOPSLA* '86, Portland, OR. 1986, pp. 284–293.
6. Castelain, E., Gentina, J. C.: Petri-nets and artificial intelligence in the context of simulation and modelling of manufacturing systems. In *Proceedings of IMACS International Symposium on System Modelling and Simulation, SMS* '88, Cetraro. 1988, pp. 28–33.
7. Colom, J. M., Silva, M., Villarroel, J. L.: On software implementation of Petri nets and colored Petri nets using high-level concurrent languages. In *Proceedings of 7th European Workshop on Application and Theory of Petri Nets*, Oxford. 1986, pp. 207–241.
8. Gentina, J. C., Corbeel, D.: Hierarchical control of flexible manufacturing systems (fms). In *Proceedings of IEEE International Conference on Robotics and Automation*, Raleigh, NC. 1987, pp. 1166–1173.
9. IEEE: *Proceedings of* 1987 *IEEE International Conference on Robotics and Automation, Invited Sessions on Petri Nets and Flexible Manufacturing*. Raleigh, NC. 1987.
10. Jensen, K., Rozenberg, G. (Eds): *High-level Petri Nets*. Berlin, Springer-Verlag. 1991.
11. Martínez, J., Alla, H., Silva, M.: Petri nets for the specification of FMSs. In *Modelling and Design of Flexible Manufacturing Systems*. Amsterdam, Elsevier. 1986, pp. 389–406.
12. Martínez, J., Muro, P., Silva, M.: Modelling, validation and software implementation of production systems using high level Petri nets. In *Proceedings of IEEE International Conference on Robotics and Automation*. Raleigh, NC. 1987, pp. 1180–1185.
13. Martínez, J., Muro, P. R., Silva, M., Smith, S. F., Villarroel, J. L.: Mergina artificial intelligence techniques and Petri nets for real time scheduling and control of production systems. In *Artificial Intelligence in Scientific Computation: towards Second Generation Systems*. S. C. Baltzer AG, IMACS. 1989, pp. 307–313.
14. Micovsky, A., Sesera, L., Veishab, M., Albert, M.: Tora: a Petri net based tool for rapid prototyping of fms control systems and its application to assembly. *Computers Ind.* **15**: 279–292, 1990.
15. Muro-Medrano, P. R.: Aplicación de técnicas de inteligencia artificial al diseño de sistemas informáticos de control de sistemas de producción. PhD thesis, Departamento de Ingeniería Eléctrica e Informática, University of Zaragoza, 1990.
16. Muro-Medrano, P. R., Soriano, J., Pujol, J., Banares, J. A., Villarroel, J. L.: Petri nets in knowledge based control models for manufacturing systems simulation. In *Proceedings of IMACS/IFAC, Second International Symposium on Mathematical and Intelligent Models in System Simulation*, Belgium. 1993, pp. 299–303.
17. Sahraoui, A., Atabakhche, H., Courvoisier, M., Valette, R.: Joining petri nets and knowledge based systems for monitoring purposes. In *Proceedings of IEEE International Conference on Robotics and Automation*, Raleigh, NC. 1987, pp. 1160–1165.
18. Sibertin-Blane, C.: High-level Petri nets with data structures. In *Proceedings of Workshop on Applications and Theory of Petri Nets*. 1985.
19. Silva, M., Valette, R.: Petri nets and flexible manufacturing. In *Advances in Petri Nets* 1989, Rozenberg, G. (Ed.). Number 424 in Lecture Notes in Computer Science. Berlin, Springer Verlag. 1990, pp. 375–417.
20. Valette, R., Cardoso, J., Atabakhche, H., Courvoisier, M., Lemaire, T.: Petri nets and production rules for decision level in fms control. In *Proceedings of IMACS 12th World Congress on Scientific Computation*, Paris, Vol. 3. 1988, pp. 522–524.
21. Valette, R., Cardoso, J., Dubois, D.: Monitoring manufacturing systems by means of Petri nets with imprecise markings. In *Proceedings of IEEE International Symposium on Intelligent Control*, Albany, NY. 1989, pp. 233–238.
22. Villarroel, J. L.: Integración informática del control de sistemas flexibles de fabricación. PhD thesis, Departamento de Ingeniería Eléctrica e Informática, University of Zaragoza, 1990.
23. Villarroel, J. L., Silva, M., Muro, P. R.: Petri nets in a knowledge representation schema for the coordination of plant elements. In *IFAC Symposium on Intelligent Components and Instruments for Control Applications (SICICA* '92), Málaga. 1992.
24. Villarroel-Salcedo, J. L., Fernández-Lladó, J. A., Sainz-Martín, J., Muro-Medrano, P. R.: Entorno de diseño de software de control en CIM. In *Sometido a la revista Informática y Automática*. 1992.