Modelling a Flexible Manufacturing System *

Vladimír Janoušek Tomáš Vojnar

Department of Computer Science and Engineering, Technical University of Brno
Božetěchova 2, CZ–612 66 Brno, Czech Republic
e-mail: {janousek,vojnar}@dcse.fee.vutbr.cz

Abstract: *The paper describes a case study on applying object oriented Petri nets (OOPN) and the associated tool PNtalk to flexible manufacturing systems modelling. From a more general point of view, this case study is a part of a broader research on the applicability of OOPN/PNtalk to modelling various complex systems in order to get some feedback and improve both the general methodology of modelling by means of OOPNs as well as the supporting tool.*

Key words: object oriented Petri nets, flexible manufacturing systems, simulation and modelling.

1 Introduction

During several recent years a new formalism based on high-level Petri nets and the object-oriented paradigm has been developed at the Department of Computer Science and Engineering of the Faculty of Electrical Engineering and Computer Science of the Technical University in Brno. Object oriented Petri nets (OOPN), which combine advantages of a firm mathematical formalism and of the object-oriented technology, seem to be an advantageous tool for evaluating both correctness as well as performance of modelled systems.

Our current research is focused on modelling various kinds of systems by means of OOPN. This article is based on experience we have gained from a grant project which verified the supposed applicability of OOPNs and of the associated tool PNtalk to working with more complex systems[1]. As a particular application area we have chosen the area of flexible manufacturing systems (FMS). For this purpose we established a cooperation with the Department of Production Machines, Systems, and Robotics of the Faculty of Mechanical Engineering at TU in Brno, which supplied us with some general information from the given area and especially we got access to a particular FMS we could model.

The model of the given FMS was created mainly in order to get some experience with FMS modelling. Nevertheless, the model can also provide a basis for further reliable considerations about the modelled system. We also suppose that after some modifications it will allow us to perform stochastic simulation and obtain performance characteristics of the modelled FMS, to prototype and optimize the control of the system, or incidentally to realise some kind of an adaptive control based on real-time analysing the model of the FMS in states reflecting the current situation in the running physical system.

*A modified version of this article was published in the proceedings of MOSIS'98, Hostýn, Czech Republic, MARQ Ostrava, 1998.

[1]Before we modelled only simple and artificial systems created for OOPN/PNtalk features demonstration.

2 The Context of the Work

2.1 A Brief Description of OOPNs and PNtalk

This section contains a very brief description of the OOPN formalism and of the tool PNtalk, which is based on it. A more detailed description can be found e.g. in [1].

The *OOPN formalism* is characterized by a Smalltalk-based object orientation enriched by concurrency and polymorphic transition execution, which allow message sending, waiting for and accepting responses, creating new objects, and performing primitive computations.

An OOPN consists of Petri nets organized in classes. Every class consists of an object net describing the internal activity of its objects and of a set of method nets describing how these objects respond to messages. The inheritance of classes is defined by the inheritance of object nets, together with sets of method nets.

A running OOPN consists of objects, which are interconnected by references stored in places, and which communicate by message sending. Each object is an instance of some class and consists of an instance of its object net and currently running instances of its method nets, which have been created as responses to some message sendings.

The pure OOPN formalism has been enriched with a modified concept of subpages, which allow us to compose objects also in a static way, and with delay and firing time concepts, which are necessary for evaluating performance or real-time characteristics. Details of these extensions are described in [2].

The *PNtalk system* is a computer tool designed to support the use of OOPNs. In fact, it is an additional tool incorporated into the ParcPlace VisualWorks Smalltalk environment. This environment makes it possible to write Smalltalk classes, create graphical interfaces, communicate with databases etc. PNtalk itself allows us to draw particular nets of an OOPN, to compile them, and to process them. OOPNs can be processed either in an automatic way, when we obtain final results of simulation only, or in a step-by-step mode, which is advantageous for debugging models.

2.2 The Modelled Flexible Manufacturing System

From the general point of view, an FMS consists of a set of flexible machines, an automatic transport system, storage facilities, and a control system, which must be able to decide at runtime which operation has to be done on which machine [5]. Every kind of product is assigned a set of operations which it should go through. These operations can be ordered, or in the general case, partially ordered. There can be more machines capable to perform the same operation and they can have attached priorities, according to which they can be chosen for performing the given operation on the given part.

The FMS whose model we are going to discuss is a little less complicated than the mentioned general case. It is the system "CIM ER 104 Cell" produced by Eshed Robotec and installed in the laboratory of the Department of Production Machines, Systems, and Robotics at the Faculty of Mechanical Engineering, Technical University of Brno, Czech Republic.

This system consists of four stations, a conveyor, and a central control system. In the first station there is installed a CNC lathe, the second one contains a CNC cutter. The third station includes an assembly site, a check site, and a turner. The last station contains an automatic store. All stations are equipped with a robot and a buffer with two places. Parts are stored on carriers and those are transported on pallets by the conveyor. A pallet stops every time it is going round a station and it can be loaded or unloaded by the robot of this station. A robot moves carriers among the conveyor, the buffer of its station, and machines of this station. Every station is locally controlled by its controller. The whole system is controlled by the central

computer. It is supposed that there is a worker who loads semi-products into the store and unloads the final products from there.

3 A Model of the FMS Based on OOPNs

The main structure of the model reflects the physical structure of the FMS – it consists of the four stations, the conveyor, the worker, and the central control. Figure 1 depicts the (partial) object diagram of the FMS model – only aggregation and inheritance relations are included.

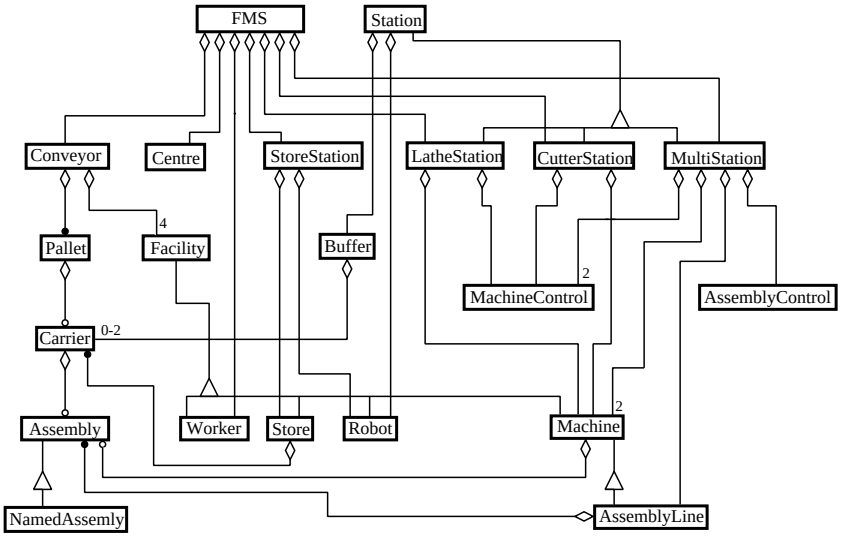

Figure 1: The (partial) object diagram of the model of the FMS

The three production stations are described by the classes `LatheStation`, `CutterStation`, and `MultiStation` derived from the same base class which is called `Station`. The class `Station` describes the common features of all of these stations. Namely, it implements a control strategy for exchanging carriers and assemblies between a station and the conveyor and it also provides its descendants with a robot and a buffer. A robot (an instance of the class `Robot`) is used for transporting carriers among the conveyor, the buffer of its station, and machines of this station. In the case of the multipurpose station, its robot is used for assembling parts in the assembly line, too. A two place buffer (an instance of the class `Buffer`) serves as a utility for physical storing carriers as well as a database describing its contents. The control mechanism for handling incoming and outgoing assemblies and carriers is based on requests of controls of particular machines present in a station and on the contents of the station buffer. This mechanism tries to reuse carriers and assemblies which are already in the station. Therefore it accepts a new assembly or a new carrier from the arriving pallet only, if it is needed and it is not present in the buffer. Similarly, a carrier or an assembly is sent away only, if an empty pallet arrives, and up to this moment no machine control has expressed any interest in this carrier or assembly.

Classes derived from `Station` add to the inherited properties the appropriate machines (instances of the classes `Machine` and `AssemblyLine`) and their control mechanisms (instances of the classes `MachineControl` and `AssemblyControl` – see figure 2). The task of the machine control systems is to accept requests for producing particular assemblies, to distribute requests for producing their source assemblies, to announce to station control units which assemblies and car-

riers are needed, and to control the movement of source assemblies into machines and of products back to the buffer. The difference between the classes `MachineControl` and `AssemblyControl` is that the assembly line control must arrange for getting more than one source assembly into the working area. The controls are included into the station classes in the form of subpages. This form has been chosen to allow sharing buffers and robots by particular machine and station control mechanisms and to allow their locking by simple removing references to them from the particular places.

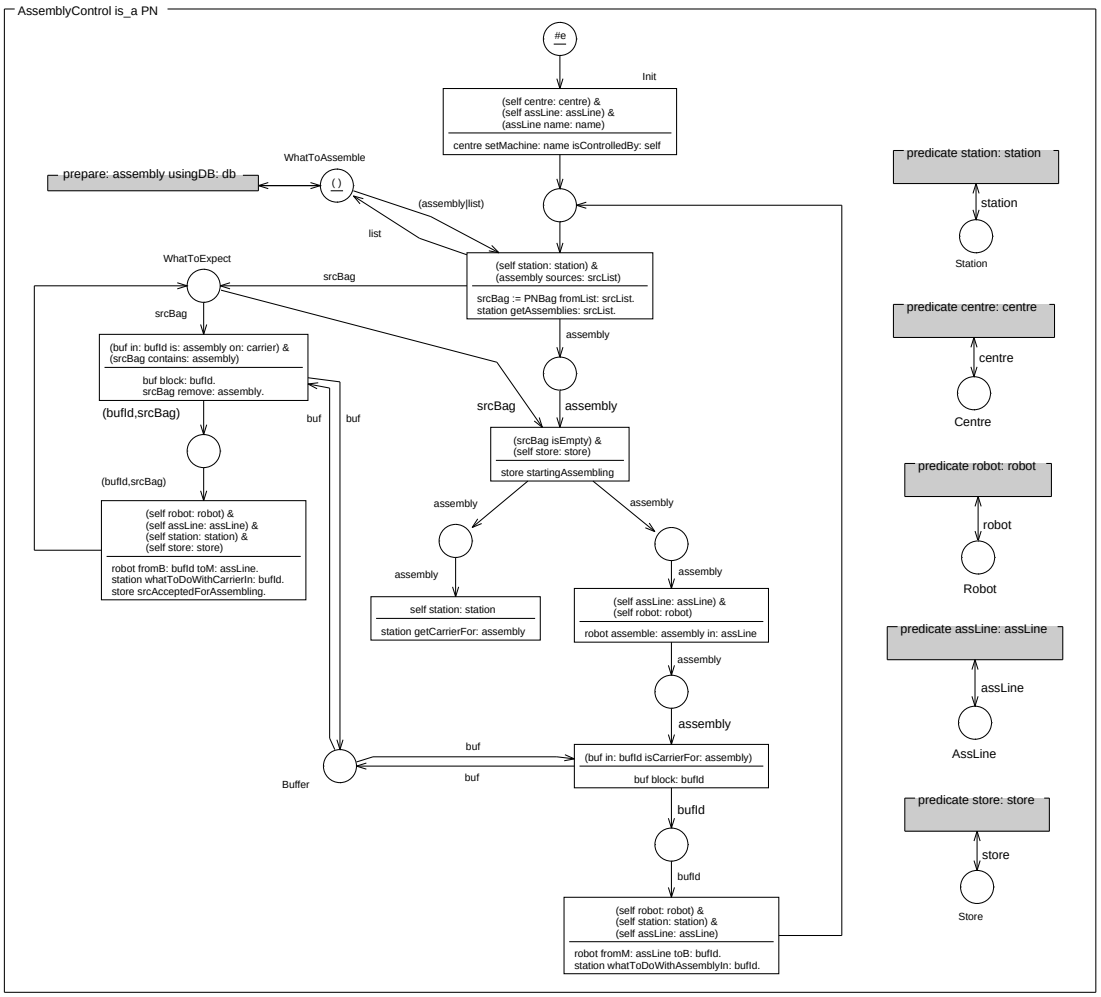

Figure 2: The object net of the class `AssemblyControl`

The fourth station containing the store is described by the class `StoreStation`, which is not a descendant of `Station`. This is because it is a little different from the other stations. `StoreStation` contains a store (an instance of the class `Store`) and a robot, but not a buffer – the buffer can be viewed as an extension (or a cache) of the store and we have left it out to simplify the model. The class `Store` represents the physical store as well as the database describing its contents. The class `StoreStation` contains also a control mechanism for moving carriers and assemblies to and from the store. The number of movements is optimized by first trying to reuse carriers and assemblies going on the conveyor back to the store rather than immediately releasing new items from the store.

The task of the conveyor (an instance of the class `Conveyor`) is easy. It simply moves pallets

among the terminals of the four stations. In front of every station terminal there is a queue of waiting pallets.

The worker, whose role is adding new carriers and semi-products into the store and removing final products from it, is represented by an instance of the class `Worker`. At the beginning he adds an initial number of carriers and semi-products into the store. Then he regularly adds new assemblies instead of the ones which have been consumed and removes produced final products. What is more, in the case of some critical situation in the store (something is missing, the store is full, etc.), the worker is signalled to try to solve this situation. However, his effort can be in vain and then the system can go in deadlock.

The last component of the model, we will discuss here, is the central control (an instance of the class `Centre`). The production control is based on two principles: a fixed maximum amount of work in progress and a fixed production mix. Those are implemented in such a way that the centre contains a circular list of final products, through which it keeps going launching the production of particular final products. However, it can start the production of a next product only, if it has already released all semi-products needed for producing the previous product from the store. The production of a single product is based on instructions which have the form of a tree of interconnected objects of the classes `Assembly` and `NamedAssembly`. The root of this tree is the final product we want to produce. Leaves of the tree are semi-products stored in the store. Every intermediate node represents an assembly produced by some machine. Every assembly contains information about the machine and the NC program which are used for producing it, about the carrier necessary for transporting it, and also about the source assemblies from which it is produced. So, the production of a particular product is launched in such a way, that we go through its instruction tree starting in the root and announce all the appropriate machine controls what they will have to produce and from what sources. This procedure begins by telling the store it will have to accept some final product and ends by telling the store it will have to release some semi-products. (This is, of course, done individually for every particular semi-product.) As soon as the first of the semi-products is released, the physical process of the production begins. But, we have to take into account that only a limited number of assemblies can be present in the FMS at the same time and therefore sometimes it is necessary to wait for some time, before a next semi-product can be released. The machine controls wait until the sources arrive and then they can produce the expected assembly and send it further. If the requests for producing arrive to stations faster than they can be granted, they are organized in queues. There is a separate queue of products for every particular source assembly. Thus it can be decided what to produce according to the available source assembly and the head of the queue attached to it. The production of a single product is then finished by storing it into the store.

At the end of this section we have to stress that we are using the term *assembly* not only for products consisting of several parts, but generally for all elements which can be transported or stored on carriers and processed by all kinds of machines. Further, two physically distinct assemblies of the same type are represented by two references to the same object of the class `Assembly`. So, we do not distinguish between logical descriptions of assemblies and their physical instances. This could be necessary, if we e.g. wanted to keep some records about quality of particular assemblies. However, we have not taken this into account.

4 Conclusions

The task of modelling the FMS has brought us some pieces of knowledge which will help us to improve the OOPN formalism, the PNtalk system, and the methodology of their usage.

It has turned out that OOPNs and PNtalk have been designed as a very general language

for modelling and programming very dynamic systems. This OOPN dynamics has not been exploited to a very large extent in the case of the FMS model – its structure does not change during a run of the model. Nevertheless, it seems that the OOPN message sending mechanism is relatively advantageous and natural for modelling a control system. And what is more, it allowed us to achieve one interesting result, namely to separate the specification of production routes from the description of the physical structure of the given FMS. Thus we can add new products, modify production routes, or remove possible products without any necessity to change the structure of the system. This result was achieved by exploiting the dynamism and generality of OOPNs and we can dare to suppose that we will be able to obtain similar nice results even when modelling different classes of systems.

Static hierarchical structuring of the model was allowed by the adopted concept of substitution transitions [2]. However, this concept has been used only inside classes `LatheStation`, `CutterStation`, and `AssemblyStation`, where controls of the particular machines have been specified as subpages of the control of the whole station. This structuring mechanism has been chosen because it allows simple resource allocation in the test-and-set manner by means of place shared by several nets as though they were a part of the same net. It would be possible to solve this situation also by using one plain net or by using the message sending mechanism. But, in the first case we would obtain a very large net and in the second one complex synchronising algorithms would be necessary.

As an example of the methodological pieces of knowledge we have gained, we can present e.g. our experience that it is advantageous to use calling predicates via `self` from transitions to access attributes of objects rather than to use dialog arcs. The semantics is the same but we avoid a necessity of introducing many crossing arc (see picture 2).

When developing the model of the FMS we encountered some weaknesses of PNtalk. They stem from the fact that this tool was developed mainly to illustrate problems arising in the area of the OOPN formalism, which was originally presented on a purely abstract level suitable for theoretical considerations only. Therefore, using PNtalk for modelling and simulation of real systems was an attempt which brought us some experience and suggestions for its future development.

In future PNtalk should be extended with facilities for designing object diagrams allowing a global view on modelled systems. It should be possible to comment particular classes and methods. The tool should allow us to highlight related Petri net areas by using different colours, line styles, etc. The editor should automatically offer typical constructions. The simulator should be made faster. Finally, facilities for checking properties in an analytical way should be provided.

Acknowledgment

This work was supported by a grant of the Fund for Sciences and Arts of the Technical University of Brno under the contract FU470006.

References

[1] Češka, M., Janoušek, V., Vojnar, T.: *PNtalk – A Computerized Tool for Object Oriented Petri Nets Modelling.* In: Computer Aided Systems Theory – EUROCAST'97, Lecture Notes in Computer Science 1333, Springer-Verlag, 1997.

[2] Vojnar, T.: *Hierarchical and Time Extensions of Pure Object Oriented Petri Nets.* In: Proceedings of ASIS'97, Krnov, MARQ Ostrava, 1997.

[3] PNtalk project. http://www.fee.vutbr.cz/~janousek/pntalk/

[4] Petri net research group at TU Brno.
http://www.fee.vutbr.cz/UIVT/research/petrinet/

[5] Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: *Modelling with Generalized Stochastic Petri Nets*, Wiley Series in Parallel Computing, 1995.