A Programming Language for Discrete Event Production Systems Based on Production Flow Schema and Mark Flow Graph[†]

Paulo Eigi MIYAGI*, Kensuke HASEGAWA* and Kouji TAKAHASHI*

The authors propose a Mark Flow Graph (MFG) based programming language and programming procedure for control systems for practical discrete event production systems (DEPS).

Based on the hierarchical structure of the system, the programming procedure is a top-down approach from the conceptual level to the detail level of the DEPS. The macro representation of the system described through the production flow schema (PFS) is gradually broken down to generate the MFG which represents the system specifications in detail. The textual description (MFG/PFS language) is derived from this MFG/PFS model. In this language, DEPS activities are described through modules which are organized in a tree structure corresponding to the hierarchical structure of DEPS activities.

Programming tests using this language for MFG controllers and MFG simulators have confirmed the effectiveness and validity of this language for synthesis, updating and debugging of complex systems.

Key Words: discrete event production system (DEPS), mark flow graph (MFG), production flow schema (PFS), MFG/PFS language

1. Introduction

Actualizing CIM (computer integrated manufacturing) is the long term goal of the present industrial society. To implement this system, first, it is necessary to provide effective tools for process modeling, signal processing and control algorithm development. Currently, process modeling at the

conceptual level (high level) is based on flowcharts, time diagrams, state machine diagrams, etc.: the signal processing and control algorithms at the execution level (low level) are generally developed with relay ladder diagrams or procedural languages written for programmable logic controllers and microcomputer-based controllers. However, these techniques are unsuitable for representing large and complex systems with operation features such as parallelism and concurrency. Furthermore, the increasing use of robots and other automated machines has generated control software written in different levels and/or forms in prejudice of the system's flexibility and maintainability because it is, in practice, impossible for one person to understand all the control specifications from such control programs.

To solve this type of problem, motivated by the potential that Petri nets¹⁾ have for modeling parallel and concurrent processes, Mark Flow Graph (MFG)²⁾, C-net³⁾, GRAFCET⁴⁾ and other techniques derived from Petri nets^{5), 6)} have been successfully introduced as an effective tool for describing control specifications and realizing the control in a clear and uniform manner. However, despite the advantages offered by these tools, the synthesis, correction, updating, etc. of the system model and programming of the controllers are not simple tasks. Depending on the size and complexity of the system, these models can become very difficult to understand and treat.

To deal with the problems related to the description of systems through techniques derived from Petri nets (e.g. MFG, C-net, GRAFCET, etc.), the authors have investigated a synthesis and description procedure for discrete event production

[†] Presented at The 13th System Symposium, Japan (1987-11)

* Faculty of Engineering, Tokyo Institute of Technology, Meguro-ku, Tokyo
(Received July 20, 1987)
(Revised October 22, 1987)

systems (DEPS) based on production flow schema (PFS) and Mark Flow Graph (MFG).⁷⁾ In this interpretation and synthesis procedure, a top-down approach from a conceptual level is applied to the hierarchical structure of the system.

Based on these studies, a programming language and a programming procedure for practical DEPS control systems are proposed in this paper. This programming language is the textual description for a MFG/PFS-based system's representation in which the parallel and asynchronous activities are described hierarchically according to a top-down approach. First, basic concepts related to DEPS and MFG are revised. Then, the structure and syntax of the programming language are explained. Through an application example the programming procedure, a language translator and the evaluation of the results are presented.

2. Discrete Event Production System and Mark Flow Graph

2.1 Discrete event production system and production process model

Consider, for instance, a manufacturing system composed of several sub-systems such as conveyors, machine tools, robots and assembly cells. Each sub-system executes production operations, i.e. physical transformations such as machining, assembling, and/or transfer operations such as loading. Depending on the functional capability of the sub-system, production operations of corresponding levels are executed. For example, while conveyors execute only simple forward/backward movements, assembly cells can execute a set of different and reprogrammable operations.

The production process in this system is characterized by the flow of "items" (material, work pieces, objects, parts, etc.) which passes in ordered form through the sub-systems and receive appropriate production operations. From this viewpoint, the overall production process can be decomposed into a set of distinct activities (events) and conditions mutually interrelated in a complex form. By an "activity" we mean a single production operation of the production process which is executed by a sub-system. By a "condition" we mean a

state in the production process such as the assembly stage, machine operation mode, number of "items" to be processed, etc.

Such systems are called discrete event production systems (DEPS) due to the discrete nature of their elements (activities and conditions). To specify and control production processes in a DEPS, we must be able to represent the type of activities and where we want the activities to be carried out. For example, the assembly activity should select the machines and loading tools to be used, and the assembly sequence. In specifying the production processes to be performed, we make assumptions about what kind of activities are available and how they are going to be carried out. In other words, we have in mind a picture of how the process that we are specifying will be executed. This picture of how things are done is referred to as the production process model.

2.2 Production flow schema and Mark Flow Graph

Considering the nature of DEPS, the authors have previously proposed a production process modeling tool for DEPS called production flow schema (PFS)⁷⁾. The PFS is a graphical model used as a tool to identify types of activities, conditions, and their mutual interrelation; i.e. the PFS model describes explicitly the concept of the production process to be carried out in the DEPS. PFS is effective for a macro representation of the production process and for achieving a top-down interpretation down to the concrete lower level activities using MFG.

MFG is a graphical technique (derived from Petri nets) proposed to represent discrete event systems, i.e. MFG is an uninterpreted tool for modeling systems which are characterized by the occurrence of events and changing conditions. In the MFG, the guarantee of "safeness" and the additional capability of input/output signals from/to the machines are considered, and the representation of the activity contents and control strategies in detail are possible. Therefore, features of DEPS such as ordering, parallelism, asynchronism, concurrency and conflict can be concretely described through the MFG.

A design procedure for DEPS control systems based on the PFS and MFG is as follows. First, the conceptual level activities of the DEPS are represented through a PFS model considering the system behavior specification corresponding to the production process. Then, the MFG describing the activities is deduced based on the activity specification and required control strategies. This procedure is repeated up to an appropriate level corresponding to the control level of the equipment responsible for the activity execution. During the synthesis process, the procedure generates hybrid models composed of elements of PFS and MFG corresponding to the hierarchical nature of the activities. Such a model is called a MFG/PFS model.

3. MFG/PFS Language

To program the DEPS control system, a textual description is introduced which reflects the hybrid model resulting from the control system design procedure in section 2. This notation is called the MFG/PFS programming language. This language is used to program the MFG engine which is the core of controllers⁸⁾ and simulators⁹⁾ for DEPS based on the execution of an MFG model.

3.1 Structure of a program written in MFG/PFS language

The structure of a program written in this language is based on the MFG/PFS representation of the system. The program is composed of modules corresponding to the activity elements of the PFS model; each module can be composed of other sub-modules corresponding to the sub-activities.

Therefore, the program has a hierarchical structure which is represented by a tree structure in which the "root" node corresponds to a module defined by "/main". The nodes corresponding to the conceptual activities of the system are represented as "descendants" of the "root" node. Fig. 1 presents an example of activities of an MFG/PFS model and the corresponding organization.

The "root" activity is denoted by "/main [...]". In general, activity modules are defined by "/ACT_NAME [...<statements>...]", where "/ACT_NAME" is the name of the activity module, and

Fig. 1 Hierarchical structure of the activities represented by the PFS model and the corresponding tree structure

"[" and "]" are the delimiters of the activity content analogous to the graphical symbol of activities in PFS or MFG models. The name of activity modules always starts with "/".

Based on the tree structure, a "path" is defined to identify each activity module. A "path", such as "/a2/b5/c2", indicates that /c2 is a sub-activity of /b5 which is a sub-activity of /a2 which is a sub-activity of the "/main" activity. Therefore, the "path" of an activity module is the sequence of activity module names between the "/main" module and the activity module to be identified. As the "path" always starts from "/main", it is omitted. As a special case, the path of "/main" is defined as "/".

The content of each module is defined through

```

/<name of the activity module>
[
  <definition of device name>
  <definition of box type>
  <definition of external signal gate arc>
  <correspondence between input port and
    external signal gate arc>
  <definition of box function>
  <definition of directed arc connection>
  <definition of signal gate arc>
  <definition of initial marking>
  ...
  /<name of the sub-activity module>
  [
    ...
  ]
]

```

Fig. 2 Structure and statements of the activity module in the MFG/PFS language

textual statements which describe the MFG of the activity (Fig. 2). These statements describe the MFG/PFS focusing on events (primitive events or sub-activities), i.e. the statements define the pre-conditions and post-conditions of the events in accordance with the control system specification. The statements may include the definition of other sub-activity modules if necessary.

3.2 Language statements

The statements presented in Fig. 2 are explained in this section. The reserved terms and their meaning are presented in Table 1. The function and syntax of the statements are explained as follows, using the MFG/PFS model of the activity "/ACT-1" in Fig. 3 as an example.

(1) Definition of the device name

This statement defines the devices that will be used in the system to be programmed, assuming

Table 1 Reserved terms of the MFG/PFS language

Term	Remarks
main	Main module (root module)
device	Activity device
s_box	Single activity box
m_box	Multiple activity box
g_box	Grouping activity box
d_box	Dispersing activity box
u_box	Unit distributing box
c_box	Capacity distributing box
t_box	Time box (for simulation purposes)
e_gate	External source of gate signal
connect	Directed arc connection
gate	Signal gate arc connection
ti	Input transition of an activity
to	Output transition of an activity
marking	Initial marking

Fig. 3 Example of an activity described by MFG/PFS model

that the existing devices are pre-defined by the operating system, i.e. devices can be hardware entities such as input/output ports or software entities such as files.

Syntax: device D1 (), D2 (), ...;

Description: D1, D2, ... are the names of the devices.

Example: device inport (), machine ();

(2) Definition of the box type

This statement defines the boxes to be used in the model.

Syntax: BOX_TYPE B1 (P1), B2 (P2), ...;

Description: BOX_TYPE is the box type defined by one of the following terms: s_box, m_box, g_box, d_box, u_box, c_box, t_box. Each one of these terms is related to the nature of the element (activity or distributing) represented by the box.

B1, B2, ... are the names of the boxes.

P1, P2, ... are parameters that depend on the box type.

The parameter for m_box and c_box is the maximum number of marks that can enter the box.

The parameter for g_box is the number of marks necessary to form one group.

The parameter for d_box is the number or marks generated from the entrance of a mark in the box.

The parameter for t_box is the number of units of time.

The parameter for s_box and u_box can be omitted.

Example: m_box mach (5), buf1 (2);

u_box buf2;

(3) Definition of the external gate signal arc

This statement defines the external gates that will be used in the model.

Syntax: e_gate G1, G2, ...;

Description: G1, G2, ... are the names of the external gate signal arcs.

Example: e_gate start;

(4) Correspondence between the input port and the external gate

This statement defines concretely the physical input port device used as an external gate element of the model.

Syntax: GATE=INPUT_DEVICE (F1, F2, ...);

Description: GATE is the name of an external

gate signal source pre-defined by e_gate.

INPUT_DEVICE is the signal input device name pre-defined by the <device name definition> statement.

F 1, F 2, ... are the functions or parameters of the input device.

Example: start=inport (1);

(5) Related box function

This statement defines the function to be executed by a device when a mark enters a specified box. This function is an extension of the output arc signal concept.

Syntax: BOX=DEVICE_NAME (F 1, F 2, ...);

Description: BOX is the name of a box pre-defined by the <box type definition> statement.

DEVICE_NAME is the device name pre-defined by the <device name definition> statement.

F 1, F 2, ... specifies the parameters and type of function to be executed by the device.

Example: mach=machine (on);

(6) Definition of the directed arc connection

This statement defines the connections between the boxes and the activity elements of the model.

Syntax: connect ACT_NAME: B 1, B 2, ...;

Description: ACT_NAME can denote the name of a sub-activity module or a primitive event such as "ti" or "to".

B 1, B 2, ... are the names of boxes previously defined. The box names necessarily include the explicit description of the "path" that specifies the activity module in which the box is defined. When the box is defined in the current module, the "path" can be omitted.

Boxes between "(" and ")" denote boxes connected in a sub-activity through a secondary flow of "items" (an example is given in section 4.2).

A signal "-" preceding the box name denotes a box connected to the input side of the activity, otherwise the box is connected to the output side.

Example: connect ti: mach, buf 1;

connect /PROC: -buf 1, buf 2,

-/mach;

connect to: -mach, -buf 2;

(7) Definition of the gate signal arc

This statement defines the gate arcs of the transitions of the model.

Syntax: gate TR_NAME: C 1, C 2, ...;

Description: TR_NAME is the name of a transition that denotes a primitive event such as "ti" or "to".

C 1, C 2, ... are the names of boxes and/or external gates previously defined. The box names and/or external gate names necessarily include the explicit description of the "path" that specifies the activity module where the box and/or external gate is defined. When the box and/or external gate is defined in the current module, the "path" can be omitted.

A signal "-" preceding the box and/or external gate denotes an inhibitive gate signal, otherwise it is a permissive gate signal.

A signal "#" preceding the multiple, grouping, dispersing or capacity boxes indicates that the number of marks into these boxes is equal to the its maximum capacity.

Example: gate ti: start, -#buf 1, /mach;

(8) Definition of initial marking

This statement defines the initial marking of the model.

Syntax: marking B 1 (P 1), B 2 (P 2), ...;

Description: B 1, B 2, ... are the names of boxes previously defined. The box names necessarily include the explicit description of the "path" that specifies the activity module in which the box is defined. When the box is defined in the current module, the "path" can be omitted.

P 1, P 2, ... are the number of marks to be placed in the corresponding box. When, by definition, the maximum capacity of the boxes is equal to 1 (s_box, u_box and t_box), the parentheses and the parameters can be omitted.

Example: marking mach (1), buf 2;

(9) Comments

Comments in the program must be delimited by "/"* and "*"/.

This language is based on the MFG/PFS model in which both "activities" corresponding to macro events and "transitions" corresponding to primitive events exist. In relation to the directed arc connection there is no distinction between "activities" and "transitions"; i.e. they are considered as

“events”. If the content of an “activity” is not detailed, this empty “activity” is treated as a “primitive event”.

In contrast, gate signal arcs can be connected only in primitive events. In the case of “activities”, gate signal arcs can be defined when the activity content is detailed, i.e. the gates can be connected to the “transitions” from which the activity is composed. An “activity” necessarily includes an activity start transition and an activity end transition.

4. Application Example

In this section the object is to show the basic procedures of programming through a simple example.

4.1 System specification

The production line consists of two production stages: A and B as shown in Fig. 4.

Robot 1 (“rbt 1”) carries the work pieces to the input of machine A (“buf_i”) which has a capacity of 2. Robot 1 does not start the carrying operation if the input of A is full.

Machine A divides the received work pieces into two parts and processes them separately. The processed work pieces are placed at the output of A (“buf_o”) which has a capacity of 2. The work pieces at the output of A are carried to buffer 1 (“buf_1”) with capacity of 12 by robot 1. Robot 1 does not start the carrying operation if buffer 1 is full.

B is a batch process machine; i.e. machine B processes groups of 12 work pieces. Work pieces are loaded from buffer 1 by robot 2 (“rbt 2”) one by one. The batch process B starts when 12 pieces have been received. The processed work pieces are removed one by one by robot 2. After that, the batch process B can accept new work pieces.

Fig. 4 System lay-out

Fig. 5 PFS model of the system described in Fig. 4 (step 1)

4.2 System description by MFG/PFS language

In this section, the steps for programming using the MFG/PFS language are presented.

step 1:

—Based on the system specification, the PFS of the system is formed (Fig. 5).

step 2:

—Identify and recognize the significant sub-activity modules.

—Substitute the distributing elements by the corresponding MFG elements.

—Represent the shared resources.

—Represent the required control mechanisms.

step 3:

—Based on the resulting MFG/PFS model of the activity, the activity module program is written in the MFG/PFS language.

step 4:

—If there is an activity to be described in more detail, go to step 2. This indicates an increment in the depth level of the activity description.

The resulting activity models and their description in MFG/PFS language are shown in Fig. 6 (a)~(d). Fig. 7 illustrates the general structure of the resulting control program.

4.3 MFG/PFS language translator

The control programs written in MFG/PFS language must be transformed into an appropriate set of data to become the input for the controllers and/or simulators which execute the MFG. A MFG/PFS language translator has been developed considering the specification of the existing MFG engines.

The control programs written in MFG/PFS language are input to the translator. The translation process outputs the data files with the description of the MFG in an appropriate form to program the

Fig. 6

Fig. 7 General structure of the program written in MFG/PFS language

Fig. 8 Input and output of the MFG/PFS "translator"

existing MFG engines (Fig. 8).

4.4 Evaluation of the resulting description

From the viewpoint of description capability, the MFG/PFS language is not restricted by the scale and features of the DEPS under consideration. However, in respect to the execution of the control program, the performance and storage capacity of the MFG engine are constraint factors. The MFG engines used in the tests can manage about 1000 boxes and transitions respectively.

Besides the example above, the MFG/PFS language has been applied to different systems. In these examples, a production system composed of about 50 machines, buffers and junction/branching elements was the subject of the programming tests. The MFG which represents this system is composed of nearly 100 boxes and 100 transitions. In such systems, the production process described in MFG/PFS language has been programmed for existing MFG controllers and simulators via the developed translator.

In comparison with the conventional tabular description of box/transition connection, the programming using the MFG/PFS language method was effective in synthesizing/correcting/updating the control program and in understanding the overall system, because of the modular approach adopted to represent the hierarchical structure of the DEPS. Furthermore, errors in editing of the program were less frequent and were easier to detect.

These tests confirm that complex activity specifications are transformed into a set of simple activities through a series of steps corresponding to the details of the activity, making synthesis, updating and debugging of complex systems easier.

By extending the pre-processing functions of the "translator", this language can accept macro definitions of variables and activity modules as well. Thus, the use of standard activity specifications allows users to develop their own libraries to facilitate the documentation, analysis and development of new activity programs.

5. Concluding Remarks

In this paper, after discussing the control characteristics of DEPS and production process model, the MFG/PFS language based on MFG/PFS modeling is proposed and a translator developed to apply the MFG/PFS language to program MFG engines. An application example is presented to illustrate the programming procedure, and the effectiveness and validity of the proposed description is evaluated.

The MFG/PFS language proposed here can be considered a basic specification of software tools in which the user develops the control algorithms to provide the desired degree of automation and performance for the discrete event production system involved.

References

- 1) J.L. Peterson: Petri Net Theory and the Modeling of Systems, Prentice-Hall (1981)
- 2) K. Hasegawa, K. Takahashi, R. Masuda and H. Ohno: Proposal of Mark Flow Graph for Discrete System Control, Trans. of SICE, **20-2**, 122/129 (1984) (in Japanese)
- 3) T. Murata, N. Komoda and K. Matsumoto: A Petri-Net Based Factory Automation Controller for Flexible and Maintainable Control Specifications, Trans. of SICE, **20-9**, 844/851 (1984) (in Japanese)
- 4) E. Ono: GRAFCET Features and Concept of TSX 7 Series, Automation, **32-7**, 53/63 (1987) (in Japanese)
- 5) D. Dubois and K.E. Stecke: Using Petri Nets to Represent Production Processes, Proc. 22nd IEEE Conf. Decision and Control, 1062/1067 (1983)
- 6) C.L. Beck and B.H. Krogh: Models for Simulation and Discrete Control of Manufacturing Systems, Proc. of IEEE Int. Conf. on Robotics and Automation, 305/317 (1986)
- 7) K. Hasegawa, K. Takahashi and P.E. Miyagi: Application of the Mark Flow Graph to Represent Discrete Event Production Systems and System Control, Trans. of SICE, **24-1**, 69/75 (1988)
- 8) For e.g. K. Hasegawa and K. Takahashi: Programmable Controller Based on Mark Flow Graph, Proc. of 21st SICE Annual Conf., 109/110 (1982) (in Japanese)
- 9) For e.g. K. Hasegawa, K. Takahashi, R. Masuda, K. Tange and T. Suzuki: Simulation of Discrete Production Systems Based on Mark Flow Graph, 9th Int. Conf. System Science, Poland (1986)