A Programming Environment for Factory Automation

Masayuki Takata

Interdisciplinary Course on Advanced Science and Technology
The University of Tokyo

E-mail: takata@ai.rcast.u-tokyo.ac.jp
Research Center for Advanced Science and Technology
The University of Tokyo
4-6-1, Komaba, Meguro-ku, Tokyo 153, Japan

Abstract

This article describes an implementation of the Manufacturing Application Programming Language Environment (MAPLE) designed by the author. In this implementation, the MAPLE consists of two parts.

One of them is the system named data-pool, which is intended to act as a glue logic and to coordinate multiple application processes by means of messages sent via the inter-process communication over the network. Using the data-pool system, as all the information is exchanged by the inter-process messages, the user can easily program data sharing and task interlocking among multiple application processes, without any knowledge about the inter-process communication nor the networking. Furthermore, this system includes event notification and condition monitoring features to eliminate needs of data polling among application processes. Using these feature, all the behaviors among the application processes are concentrated within the data-pool system, and each application processes achieve high modularity and maintenanceablity.

Another is the Factory Automation Programming Language Processor, which controls equipment and monitors status in the manufacturing work-cell, running with the data-pool system. The language specification is designed to ease describing the task done in the manufacturing work-cells.

1 Introduction

In the activities of the "Survey and Study Committee for Standardization of Factory Automation Programming Language" in Japan, the programming language reported by the author in the MAPLE '90 Symposium has been continuously evaluated, and its language specification is refined according to the recomendation. Here, the fruit of the refinement is described. Please note that the following design and the specification are the private opinions of the author, and are not the output of the FAPL committee introduced above.

1.1 The Model of the Manufacturing Work Cell

The manufacturing work-cell consists of devices (such as NC machines, robots, work-piece conveyers, operation panels, sensors, actuators and so on) and a work-cell controller, which are connected together and enabled to communicate via signal lines. The work-cell also has communication channels to communicate with something resides outside of the work-cell, such as area computers, other work-cells, and operators.

The manufacturing work-cell is requested to have following functions;

- Manufacturing planning,
- Task / device selection,
- Giving directions to the operator,
- Device / machine control,
- Data acquisition / analysis,
- Process monitoring,
- Communication.

In the production area, the manufacturing work-cells as described above are installed and connected by some transporting systems and a communication system. Work-pieces and production data are distributed on demand requests from work-cells, and the results of the process done in work-cells are returned for the following requests. This imaginary field, which resides outside the work-cells and consists of storages for the work-pieces in process and the process / control data, is named as **coordination field**, as this field connects many work-cells by means of work-piece and information. Using the concept of the coordination field, we can easily design autonomous production control system.

1.2 Software System for FA Controlling & Monitoring

The typical manufacturing work-cell is supporsed to be as shown in the Figure 1.

In the manufacturing work-cells, the work-cell controlling and monitoring software systems should have abilities to realize followings.

- Work-cell model database which contains information about the characteristics, configurations and operations of the work-cell itself.
- Autonomous task selection which selects a valuable task using the work-cell model database.
- Task decomposing which decomposes the selected task into much more simple tasks and schedules those

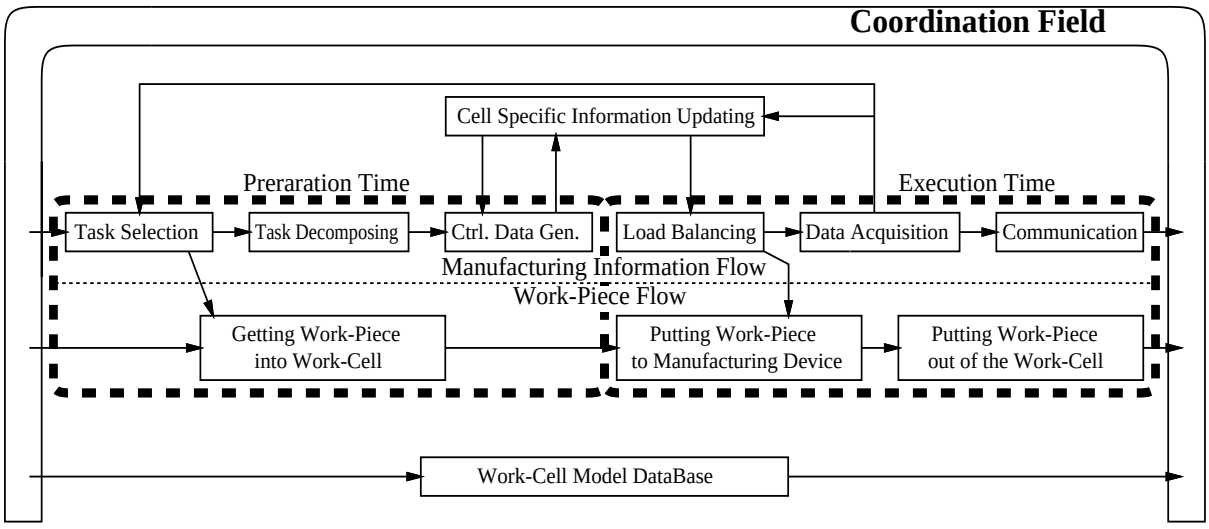

Figure 1: The model of manufacturing work-cell

divided tasks using precedence relations among them.

- Device control data generation which generates device control data for the virtual manufacturing devices.
- Dynamic task dispatching which balances load of devices within the work-cell and makes the utilization rate of devices much higher.
- Work-cell monitor which checks the operating status continuously and takes proper actions for the events.
- Information updating which informs the results of the tasks done.

To achieve objectives listed above, the data to be handled in the work-cell controller software systems consists of followings.

- Data representing physical bodies to be treated by the work-cell, such as the work-pieces.
- Data representing physical bodies prepared in the work-cell to be used in machining or assembling work-pieces, such as equipment, devices, tools, storages, controllers and so on.
- Data representing knowledge to be used or generated in the work-piece treatment, such as process data, task selection rules, device control methods and so on.
- Data representing knowledge to be used for efficient use of knowledge, such as how to select proper rule set.

In order to make the work-cell control systems autonomous, it is important to use these various data freely and timely. Moreover, these data should not have any contradiction in any instant. As these data should be represented in the work-cell controller software system, the FA programming language is requested to be able to handle these data.

1.3 Two Important Element of FA System – Coordination & Execution

In general, the required features of the FA Programming Language Environment are categorized into two major domains; one is to make all work-cell controllers cooperative and communicative, and another is to make each of the work-cell controllers clever and autonomous. The former is realized by means of the coordination field, i.e. by the environment, and the latter is realized by application programs written by users, i.e. using the programming language.

In the following sections, these two elements of the FA Programming Language Environment are described.

2 The Data-Pool System

The **data-pool** system has been developed to support application programs by means of data sharing, event notification and condition monitoring. Using the inter-process communication over the network, the data-pool system works as the infrastructure of the autonomous distributed work-cell control systems, and it also ease development and maintenance of event driven application. The data-pool system is thought to be one implementation of the coordination field described above.

The data-pool system is based on the client-sever model of the transaction processing, and basically consists of one server and multiple clients as many as users wish. The data-pool server system works as a process running on the work-cell controller operating system, and application programs including the

data-pool clients are operated as other processes which communicate with the data-pool server process using inter-process communication vehicle provided by the operating system running on the work-cell controller.

As shown in the Figure 2 below, the data-pool server consists of two major parts; the communication interface subsystem and the data management subsystem. The data management subsystem also consists of two parts; the data change monitor subsystem and the data storage subsystem. The data storage subsystem manages association between the name and its data provided by the client. The access and / or update request from its client is processed by the communication interface subsystem, and the data retrieved from the data storage subsystem is manipulated according to the request.

If the data-pool server has multiple clients over the network system, those clients can share all the data by means of accessing the same server. When some data in the data storage subsystem are changed by means of update request from one of clients, the data change monitor subsystem sends the notification messages to the clients registered in advance. Moreover, some relational expressions representing conditions can be registered within the data-pool server, and the data change monitor subsystem sends the notification messages when the condition is met. Using these feature, the data-pool clients need not poll the status of their interested data or relations amoung data.

2.1 The Usage

The most basic element of the data-pool system is the pair of the **name** and the **value**. This is the atomic unit of the information kept in the data-pool system.

The **name** is something like variable identifier in programming language, and each of them can have a value. The name is a sequence of some identifiers, separated by a period, such as `abc.ijk.xyz`. Using this format, users can denote data structure by the name sequence. For example, users can use the name such as:

```
<item_identifier> . <aspect_identifier>
```

to represent certain aspect of certain item. Using this notation, the application programmers can implement arbitrary data structures using names. In the elements of one structure, their names contain same identifier sequence in its leading part, and the trailing part of their name differs from each other. The leading common part is called **stem** and the trailing part is called **variant**.

Each name may have some **attribute**s. The attributes denote optional settings or functions of corresponding names, and the data-pool system changes its behavior according to the values of attributes.

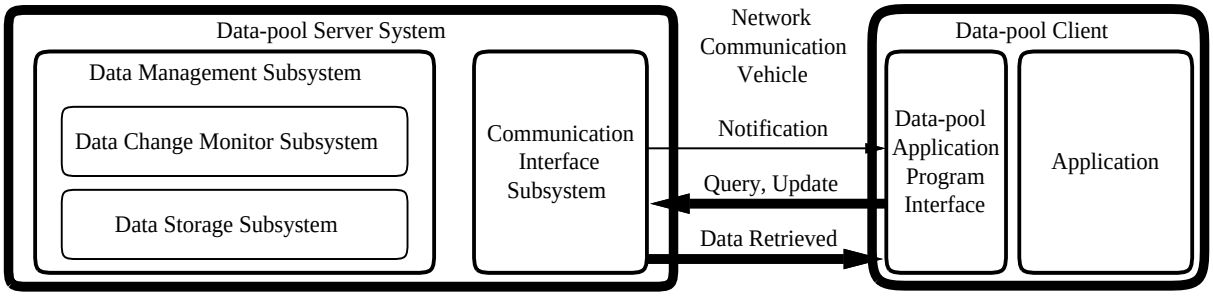

Figure 2: The data-pool system

As the value of the name, application programs may specify one of followings; integer, floating point real, character string, expression and link. As the name itself is not typed, users may bind different types of data in turns. If a client accesses the name bounded to an expression value, the expression is evaluated and the result is returned to the client as the value of the name. Using the link type value, users can access one value with more than two name. This is equivalent to the function of the "symbolic link" found in many of the tree structured file systems, and users can use values of this type as pointers.

2.1.1 Data Sharing

The most simple usage of the data-pool system is for data sharing. Originally, the data-pool system was designed as the data management subsystem of the FA programming language system described in the successive section. As it is very important to share some data within multiple application processes, the data-pool system is prepared as the separate process, in order to provide common data for those related processes.

As there are some names which are deeply related to each other, those values should be updated simultaneously to keep those values consistent. In the data-pool system, there are many application program interface (API) used to change values of multiple names with only one transaction. For the names which values are updated by multiple clients, there is another API to do mutual access control.

2.1.2 Data Change Notification

In order to eliminate the needs of data polling, the feature of data change notification is used. The clients, which want to receive the change notification of the certain name's value, can register the name of the client itself to the interesting name. The clients may register other client's name as the notification destination, the user can implement a kind of dispatcher which dispatches some processes to the events of the data change.

On the time when the data-pool server system receives data update request, the system searches for the clients registered as the notification destination, and then notify the fact of change to all the registered clients. In this way, the application programs are free from polling to find the change of status.

2.1.3 Automatic Update of Dependent Data & Conditon Monitoring

Some clients may need to know the value of name being a certain constant value, or the value of name satisfies a certain condition. The data-pool system can be set to send a notification only if certain condition is met.

The each name of the data-pool system can have a dependence list as the name's attribute. If the name listed in the dependence list is updated, the value of the name is updated to the result of a certain expression, which is also registered as the value of another attribute.

Using this mechanism, if the values of more than one names listed in the dependence list vary, the registered expression is evaluated and the result is assigned to the name itself as its value. If this new value differs from the former value, the data change notification is sent to its subscribers.

The condition monitoring can be done by registering conditional expressions. With this mechanism, user can implement multi-way branching by comparing value of a name to the constants. Usually, only one of them holds true, and the corresponding application program is notified. If the branching is implemented as above, users can add another selection later without changing any existing application

program, but only adding new condition expressions comparing a name and a constant. This flexibility is implemented by the data-pool system and the registered condition expressions.

2.1.4 Using Automatic Process Invocation

In the case of the data-pool system having the process invocation capability, the system invokes an application program and then sends a message to the process just invoked, when some names are changed or some conditions met and when the notification receiver is not ready to run.

Assuming that users make it a rule to set the execution result or final status to the name in the data-pool system at the end of each process execution, this status updating causes the following tasks to start their process, just like falling dominos.

As the data-pool system can link the application programs coded and compiled separately using the shared data and the notification messages, those application programs obtain high modularity and readability.

2.2 Support Utilities

Some application programs are supplied as the utility programs of the data-pool system. These clients have very universal functionalities and adapt wide range of the user applications.

2.2.1 Independent Requestor

This utility let the application programmer access the data-pool server from the operating system command line. This requester can emit all of transactions defined in the data-pool system, and parses the resulting message and displays it on the display screen.

The independent requester has very limited user interface, but can request all the operations available in the data-pool server.

2.2.2 Data-Pool Display Subsystem

This display system has a character mapped display screen, and it can display the values of the names in the arbitrary data-pool system on its screen. All the control messages sent to this subsystem are passed via a certain data-pool server, and using the change notification feature of the data-pool system, the data-pool display utility can display the real-time values of the names specified in advance.

Some panels can be shown on the display in turn, and the application programmers can define multiple fields in a panel. Each field has its origin point, width and depth, and foreground / background colors as its attributes. As these parameters are also defined in the data-pool system, the size or location of the field can be dynamically changed. Therefore, simple animation can be displayed on the screen, and this subsystem is enough to be used as a status monitor display panel.

2.2.3 Data-Pool Inspector Subsystem

As the data-pool display subsystem is only for displaying, there should be the tool to manipulate the contents of the data-pool server interactively. The data-pool inspector is the tool to retrieve and to update the values of names and attributes in the single data-pool server.

On the screen of the data-pool inspector subsystem, the names or attributes selected by the user is listed vertically. On the right of them, its current value is displayed. In order to update those data, user places the cursor on the current value to be changed and overwrites the new value on it.

After the overwriting has been completed, user should do the "commit" operation. This operation causes a transaction to update the values. The data-pool system has two kinds of update transaction. One simply updates the value and has no concern for their old value. Another transaction first compares the values of selected names or attributes to the data sent from the client. If all the values are identical to the data from the client, then the values of the names are updated. Otherwise, the contents of the data-pool remain unchanged and the current value of the selected names are sent back to the client. Using this "secure" transactions, the data-pool system realizes semaphore operations.

2.3 Petri-net Based Graphic Programming Systems based on The Data-Pool

Using the features of the event notification and the condition monitoring, users can describe relation among places shown in the Petri-net diagram. In the Petri-net diagram, each **place** describing the task are associated to the **transition** which describes the firing condition of the following places. In order to fit Petri-net diagram to manufacturing control applications, some extensions of the Petri-net are announced.

The Sequential Function Chart (SFC) is one of such diagrams, which is added external signals as the firing conditions. In this diagram, the transition is fired if all of its upstream places are fired and all of its incoming external signals have logical true value. Once the transition is fired, the upstream places of the transition are now not fired and its downstream places are fired.

This relation is easily translated to the scheme of the condition monitoring in the data-pool system, assuming that the values of the names are updated by associated processes to the names, when those processes have completed their tasks, or when they have checked the status of external signals. The value of the condition expression is evaluated, at each time when one of names being referred in the condition expression is updated, that is when one of the preceding processes completes. The registered condition is met only if the value of condition expression becomes logical true, and then the notification message is sent to the data-pool client, that is the trigger to start the successive processes.

The condition expressions registered to the data-pool system can be made from SFC diagram automatically. Assuming the tool which has graphic interface to compose SFC diagrams and is able to generate condition expressions for the data-pool system, the application programmers can define the relations among the application processes using interactive graphic interface, and the generated condition expressions can be directly loaded to the data-pool system. Making glue logic among the application processes becomes quite simple and easy to maintain.

2.4 Application Program Interface

The Application Program Interface (API) is provided as a function library package. Using this library, application programmers need not know about the network communication programming at all. So, the application programs become simple and easy to read.

The data managed by the data-pool system is represented by a union type of the C language. This type is named **DataCell**, and the library package also contains a complete set of functions and macros to handle this data type.

The data-pool API consists of three types of the interface routines, which are;

- Network control and message processing API,
- DataCell structure handling API,
- Data-Pool accessing API.

The last type of the API includes following categories of API;

- Retrieving or updating values of names or that of name's attributes,
- Controlling the change notification messages,
- Copying the values of name or attributes without getting them back to the client,
- Querying system status, such as list of all names in the data-pool,
- Saving or restoring the whole contents of the data-pool.

These API require the data-pool identifier as its first argument, in order to allow accessing multiple data-pool systems simultaneously from one application program.

2.5 Message Protocol

All the operations relating the data-pool system are requested by a client via transaction message. Then, requested operation is carried out and the requested information is returned via the transaction.

As this protocol is completely independent from the network communication vehicle used by the package, any kind of networks can be used as its vehicle. Moreover, the protocol is completely hidden from the application programmers, the transplantation work of the application program is quite simple.

All request transaction from the data-pool client is compressed within single message, and is sent to the data-pool server. At the network interface subsystem in the data-pool server, all the requests from multiple clients are serialized and served in first come first served basis. So, all APIs are atomic operation for the data-pool server.

3 The Language Processor

As the language described below is not named yet, it is referred as "FAPLE" here for the convenience.

The FAPLE language processor is a kind of object oriented language interpreter system. This language is designed assuming the existence of the data-pool system, and it owes the data-pool system the global data management, as well as inter-interpreter communication, process synchronization, mutual execution and condition monitoring.

As the process of the FAPLE language interpreter can be started from the data-pool system, the action of a certain condition can be composed in this programming language. In this case, if the condition met, the data-pool server invokes the language interpreter process and send a message to the interpreter process just created.

3.1 Required Characteristics

There are many requirements for the programming language for the manufacturing work-cell controller as follows;

- Work-piece in process, manufacturing equipment and the task done in the work-cell can be abstracted or represented as virtual objects.

- It is easy to compose the definitions or the actions for the exceptions or the event occurred.
- The actions taken for the exceptions are bound to the exceptions dynamically.
- The concurrent processes and relations among them can be expressed clearly and explicitly.
- The program can be written from the view point of the work-piece processed.
- Manufacturing application programs themselves can be managed dynamically.
- The written programs can be self-descriptive and easy to decrease the maintenance documents.

Furthermore, followings are assumed to enhance its usability;

- Assuming production engineers as the typical programmers using this language.
- Defining the statement to control equipment not so rigid, in order to ease extending opereations users can specify.
- Defining the program structure and the control structures, such as the concurrency or the event handling, rigidly.
- Defining atomic virtual operation set for each classes of machines controlled, and prepare the control methods for each virtual operations.
- Making the presentation of the program familiar to ordinary users.
- Making language specification not to force the implemented system slow and complicated.

3.2 Granularity of the "Object Oriented Data Processing"

Using the data-pool system, the application processes are executed in the message sending basis. This is one implementation of the object oriented data processing, but the author used massage sending mechanism also in the processing of the application processes.

The message based execution is not always good in the statement level, because of its large over-head. But it is very effective for the information hiding purposes, and especially for the virtual manufacturing device (VMD) control. So the author selected the data abstraction rather than the execution efficiency.

But this problem is still left open, and the further study is seriously required.

3.3 Syntax & Semantics

3.3.1 Message Sending

The message sending is expressed as:

```
<receiver_object> . <message_selector> ( <argument_list> )
```

where the argument list is the comma separated list of the expressions, and the list is optional.

The FAPLE language processor searches for the appropriate method to the combination of the message receiver and the message selector, and then executes the method found to evaluate the value of the message sending expression. The method searching is done by the language processor at the evaluation time, not at the compilation time.

Though all the processes executed in the language processor are expressed as above, some frequently used operations are expressed using unary or binary operators for the simplicity. But also in this case, the expressions with the operators are once translated internally to the message sending expression, and then evaluated with method searching.

3.3.2 Synchronous Data Updating

In the manufacturing data processing, multiple data should be updated synchronously, in order to keep consistency among multiple data.

In order to express the need of synchronous data updating, following structure is prepared;

```
Atomic <statement_list> EndAtomic
```

The assignment done in this structure is deferred until the structure completed, and then they are updated at the same time. The data is not updated while the statements inside is under execution, so the program:

```
Atomic A := B; B := A EndAtomic
```

swaps contents of variables `A` and `B`.

3.3.3 Exclusive Execution

The semaphore operation is also implemented as the variation of the synchronous data updating. The binary semaphore is implemented by a name in the data-pool and usually it has a value "unbound", which shows no one owns the semaphore. When one process wants to own the semaphore, it emits a transaction which assigns the language interpreter process identifier to the name only when the name has the value "unbound". This operation is safe from the deadlock state and is convenient because it is very easy to know which process is owning the semaphore.

Using this protocol, a structure for mutual execution is expressed as follows;

```
Exclusive <name> Then <statement_list> EndExclusive
```

The specified name is used as the name of semaphore, and the following statements are executed if the language interpreter process succeeds to own the semaphore.

3.3.4 Exception Handling

The exceptions mentioned here includes the signals coming from the outside of the work-cell controller, the conditions detected within the data-pool system, and the exceptions raised by the raise statement or the language processor itself. The language processor takes registered action responding to the exception occurred.

In the FAPLE language system, every process can register a statement for each exception, which is executed when the exception occurs. When the occurrence of some event is expected, the exception control statement should be coded with following syntax prior to the occurrence.

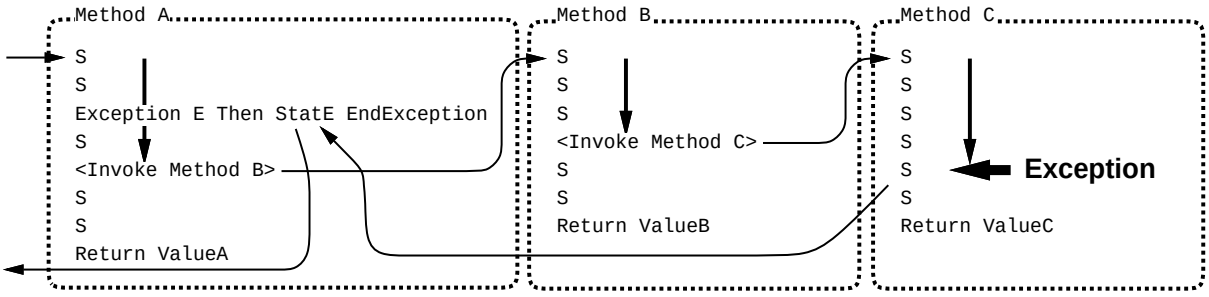

Figure 3: Control flow of the exception handling

```
Exception <identifier> Then <statement> EndException
```

If the exception occurs in the period from the execution of the exception statement until the completion of the method in which this statement is included, the method with the exception statement and all the methods invoked as its children are aborted, and the specified statement is executed instead. This operation sequence is illustrated in Figure 3.

The binding of the exception and its handling statement is kept dynamically, and old handling statement is pushed down if the new exception statement is executed while the former is still valid.

3.3.5 Daemon Control Statement

The daemon is implemented by the combination of the condition monitoring by the data-pool system and the automatic language interpreter process invocation. Daemon is defined by the statement as follows;

```
Daemon <name> As <statement> On <condition> EndDaemon
```

The specified condition is registered to the data-pool system, and its value is evaluated every time its referring names are changed and is assigned as the value of the specified name. If the value becomes logical true, the notification message is sent to the newly created language interpreter process, in order to make the specified statement being evaluated.

3.4 Class Library Hierarchy

The class library is the key of this language system to define its ability to control devices in the work-cell or to make the control system intelligent. Generally speaking, the classes to be prepared in this language processing system are divided into some categories listed bellow.

- The classes to keep the language processing system operational by means of defining system behaviors, the operating system interfaces, and the fundamental data structures the system expecting to be, such as class, method, variable, identifier, method dictionary, simple (unstructured) data types, string, set, array, vector, matrix, e.t.c.
- The classes to represent materials, parts, work-pieces in process, and final products.
- The classes to describe and to control VMD which represents abstracted devices, such as robots, NC machines, work-piece vehicles, storage systems, device control systems, e.t.c.
- The classes to represent physical manufacturing devices which describes the devices themselves and translates their abstracted operation commands for VMD into their appropriate commands according to each device protocols.
- The classes to represent the knowledge on the manufacturing processes, such as process description, process selection rule, e.t.c.

These classes should be defined and be arranged in the tree structure. As the system expects that the classes far from the root of the tree structure describe more specific and detailed objects, the arranging order in this class hierarchy is very significant.

In the case of the device controlling classes, there should be methods for some purposes such as real-time control, run-time simulation for scheduling, automatic control file generation, and execution simulation for program verification. If the single class should have all the methods belonging to all these categories, the number of the method may be too many to manage. In those cases, it is thought effective that making the class describing the device specifications and the classes containing methods for each categories separately and arranging them as the former near the root and others as its descendants.

4 Conclusion

In this article, one implementation of the MAPLE is described. The author believes the effectiveness of the coordination space, which is the glue of tasks to be done in the manufacturing work-cells, defined as the methods and the processes.

There are many application programs and operating systems which have many useful tools, but the succeeded systems among them have sophisticated mechanism to integrate prepared tools.

The author would like to emphasise that the smart glue mechanism is the very thing to make the programming system powerful and easy to be programmed, and also hopes that the data-pool system is the good candidate of such glue logics.

References

- M. Takata, M. Hasegawa, H. Matsuka, "A Unified Environment for Production Operation," *Proc. of Symp. on Manufacturing Application Programming Language Environments*, pp. 85-94, 1990.
- M. Takata, "Object Oriented Production Control Language," *Trans. of the Japan Society of System Engineering*, vol. 12, no. 2, pp. 13-30, 1988 (in Japanese).
- M. Takata, "FA Software Development on Object Oriented Paradigm," *Journal of the Japan Society of Precision Engineering*, vol. 58, no. 10, pp. 27-29, 1992 (in Japanese).
- M. Takata, "Object Oriented Language for Production Specification: OLPS, Language Specification and User's guide," IBM Research, Tokyo Research Lab., TRL Research Report, TR87-1012, 1986.
- A. Goldberg and D. Robson, "Smalltalk-80: The Language and Its Implementation," Addison Wesley, 1983.
- D. R. Barstow, H. E. Shrobe and E. Sandewall (Ed.), "Interactive Programming Environments," McGraw-Hill, 1986.
- K. Hasegawa and K. Takahashi, "MFG (Mark Flow Graph)-based Programmable Controller and its Application to FMS," *Proc. of Int. Conf. on Systems Engineering*, pp. 306-311, 1984.
- K. Hasegawa and H. Ohno, "On Programming of Conventional Programmable Controller by Using Mark Flow Graph," *Proc. of Int. Symp. on Circuits and Systems*, pp. 933-936, 1985.
- R. Masuda, and K. Hasegawa, "Mark Flow Graph and its Application to Complex Sequential Control System," *Proc. of 13th Int. Conf. on System Science*, pp. 194-203, 1980.
- T. Murata, N. Komoda, and K. Matsumoto, "A Petri-Net Based FA Controller for Flexible and Maintainable Control Specification," *Proc. of IECON Conf. on Industrial Electronics*, pp. 362-366, 1984.
- J. L. Peterson, "Petri Net Theory and the Modelling of Systems," Prentice-Hall Inc., 1981.