An efficient development method for plant control software using algebraic specification and software components

***Masakazu Takahashi*†, *Satoru Takahashi*††, *and Kazuhiko Tsuda*††**

†*Shimane University, Nishikawatsu 1060, Matsue, Shimane, 690-8504 Japan*
††*Tsukuba University, Otsuka 3-29-1, bunkyo-ku , Tokyo, 112-0012 Japan*

Summary
This paper proposes a method to efficiently develop Plant Control Software (PCS) using software components. PCSs are typically developed in individual order basis, and the conventional component-based development methods have difficulty of selecting appropriate software components based on design specifications. The proposed method addresses this problem by selecting software components using algebraic design specification. We have developed a prototype of Integrated PCS Development Environment (IPDE). This integrated environment supports processes from requirement definition through software implementation by gradually detailing algebraic design specification. In the IPDE, an initial requirement specification can be generated only by defining characteristics of targeted PCS using its domain model. Using this IPDE, even less experienced PCS developers can easily develop PCSs. Furthermore, the IPDE improves development efficiency and quality (especially adaptability and reliability) of PCSs. As a result of applying the IPDE to the actual PCS developments, the PCSs have been developed successfully using software components, except input and output parts which depend on hardware specification of control equipment. The reused rate of the source code is 65[%], and the reduced rate of development time is 58[%].

Key words:
Plant, control software, algebraic specification, software components, integrated development environment.

1. Introduction

A plant is mechanical facility for manufacturing products or for processing materials. Each plant has its own manufacturing or processing method depending on the intended purposes of customers. Accordingly, each Plant Control Software (PCS), which is attached to plants, need to have its specific functions. For this reason, PCSs are developed individually for the targeted plants, and their functions or performance vary depending on the experiences of PCS developers. Especially, the inconformity in PCSs causes the problems such as lower reliability, longer development period, and larger cost.

To solve these problems and to develop PCSs keeping a certain level of quality, development methods must consistently support the processes from requirement definition through implementation. These problems are typically addressed with the methods such as CASE[1)] or Domain Model[2)]. However, these methods impose heavy load on PCS developers to be familiar with software development methodologies and models.

In the area of software development, attention is being given to software component-based development methods. In this paper, we propose a PCS development method which applies software components[15)]. The followings are the outline of the proposed method:

(i) Define the requirements for a PCS, using software components for PCS prototype development.
(ii) Based on the requirement definitions, generate requirement specifications described in algebraic specification (Z language).
(iii) By taking an advantage of Z language which allows hierarchical description, detail the requirement specification hierarchically and create design specifications for implementation.
(iv) Based on the detailed design specification, select software components used for PCS development. In this selection, the lists of function categories and input/output data shall be used.
(v) Construct the targeted PCS by combining the selected software components. Then customize the PCS for specific requirements, if necessary.

We have developed a prototype of Integrated PCS Development Environment (IPDE) which implements the above stated method. As a result of applying the IPDE to several PCS developments, the PCSs have been developed consistently from requirement definition through implementation, and the development efficiency and the quality of the PCSs have been improved. The reused rate of the source code on the software component-based PCS developments is 65[%], and reduced rate of development time is 58[%].

This paper is organized as follows: Chapter 2 discusses the problems and the solutions on PCS developments. Chapter 3 explains how to create requirement specifications for PCSs. Chapter 4 describes the method to create preliminary and detailed designs. Chapter 5 describes the method to select appropriate

Manuscript revised January 26, 2006.

software components. Chapter 6 evaluates the application of IPDE on actual PCS developments. And Chapter 7 concludes the discussion and mentions about future work.

2. Problems and solution on PCS development

The following factors make it difficult to develop PCSs using software components.

- Requirement specifications contain ambiguity because they are described in natural language.
- Without guidelines or reference information, it is difficult to fully define additional information used for implementation design (preliminary and detailed designs) of PCSs.
- There is a gap between software components and PCS requirement specifications because the granularity (function range to be covered) of software components are too small.

To develop PCSs using software components based on requirement specification, design details in preceding processes must be transmitted to subsequent processes accurately. This requires a language to describe design details precisely. We use Z language which allows formal specification description. Formal languages[4),5),6)] like Z generally require the knowledge of advanced mathematics and software engineering. In this proposed method, however, the knowledge of Z language is not required because it is used only for transmitting information between processes (refer to phase 1 to 3 described below). This enables wider application of this development method. Moreover, the standardized interfaces between the processes enable accurate transmission of design details. Z language also allows hierarchical specification description. In our proposed method, requirement specifications in Z language are hierarchically detailed into preliminary and detailed designs. This detailing process is repeated until it reaches to granularity of software components. This achieves seamless development environment.

To address the above mentioned three problems, this method divides development processes into the following three phases.

Phase 1: Describe requirement specification in Z language.

PCSs require similar functions even if the control targets or the control methods are different.[1)] Utilizing this characteristic, describe the functions of requirement specification in Z language in advance, then additionally describe the inherent part of each PCS to complete the PCS requirement specification.

Phase 2: Create design specification based on guidelines.

Design PCS implementation based on the requirement specification in Z language. Firstly, divide a plant into sub-plants which represent the actual control units, then classify them into two modules: sequence control and feedback control. Next, divide the functions of each module according to the predetermined design guidelines and additionally define the information required for implementation. Then output the results as PCS implementation specification in Z language.

Phase 3: Construct the target PCS by combining software components.

Based on the PCS implementation specification, search for an applicable software component for each function unit[13)]. When no applicable software component is found, either divides the function unit until the applicable software component is found or develops new programs for the function unit.

Fig.1 shows the outline of the proposed PCS development method.

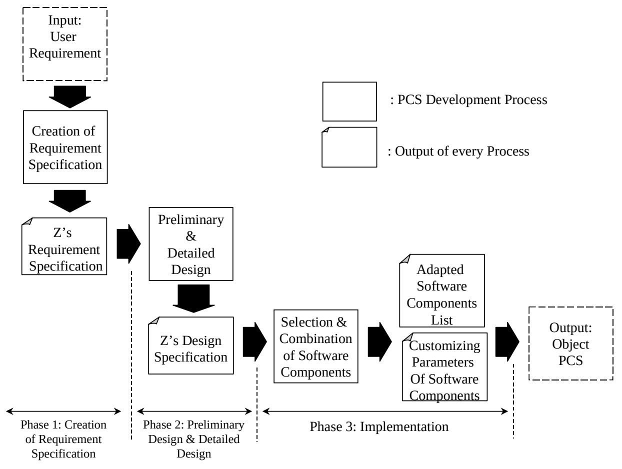

Input: User Requirement
: PCS Development Process
: Output of every Process
Creation of Requirement Specification
Z's Requirement Specification
Preliminary & Detailed Design
Z's Design Specification
Selection & Combination of Software Components
Adapted Software Components List
Customizing Parameters Of Software Components
Output: Object PCS
Phase 1: Creation of Requirement Specification
Phase 2: Preliminary Design & Detailed Design
Phase 3: Implementation

Fig.1 Outline of proposed PCS development method

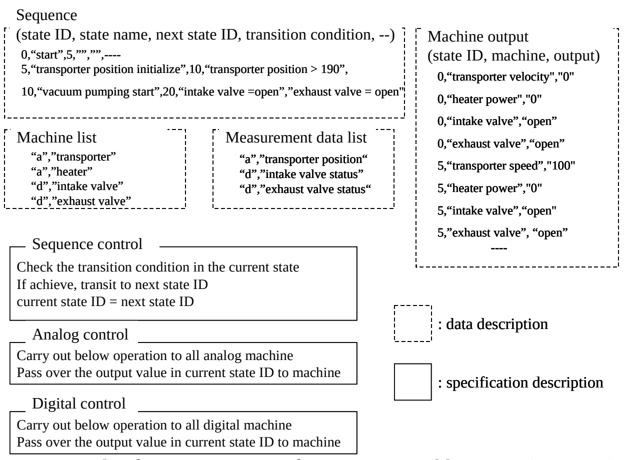
3. Problems and solution on PCS development

PCSs are normally composed with three modules: sequence control, feedback control, and input/output. The input/output interfaces of control equipments are not determined at requirement specification phase. At this phase, functions, performance, and sequences of PCS need to be defined. This chapter discusses how to describe requirement specification of sequence control and feedback control modules in Z language.

3.1 Requirement Specification of Sequence control

Module in Z Language

Fig.2 shows a natural language version of requirement specification for PCS sequence control module. This specification consists of four sections which describe: sequences, values to be output to machine on each process, and machines in plant and its input/output (sensor input and operational volume output). Now we discuss the standardization of data structure to formalize requirement specification of sequence control module.



Sequence
(state ID, state name, next state ID, transition condition, --)
0,“start”,5,””,””,----
5,“transporter position initialize”,10,“transporter position > 190”,
10,“vacuum pumping start”,20,“intake valve =open”,”exhaust valve = open"

Machine output
(state ID, machine, output)
0,“transporter velocity","0"
0,“heater power","0"
0,“intake valve”,“open”
0,“exhaust valve”,“open”
5,“transporter speed”,"100"
5,“heater power”,"0"
5,“intake valve”,“open"
5,”exhaust valve”, “open”

Machine list
“a”,”transporter”
“a”,”heater”
“d”,”intake valve”
“d”,”exhaust valve”

Measurement data list
“a”,”transporter position“
“d”,”intake valve status”
“d”,”exhaust valve status“

Sequence control
Check the transition condition in the current state
If achieve, transit to next state ID
current state ID = next state ID

Analog control
Carry out below operation to all analog machine
Pass over the output value in current state ID to machine

Digital control
Carry out below operation to all digital machine
Pass over the output value in current state ID to machine

: data description
: specification description

Fig.2 Sample of requirement specification in natural language (sequence)

At first, we consider the data structure of sequence control module. The sequences of plant are described as state transition tables. The state of plant means "the unit to uniquely distinguish the plant by the combination of hardware components' states, such as location of machine or On/Off of switches". The transition means "to transit the state of plant from A to B by changing the states of hardware components". In here, transition criteria are defined as "conditions to trigger a transition" such as "if the power of the heater is On" or "if the ambient temperature is higher than 300[K]". Generally, the relation between state and transition criteria is one-to-many because more than one criterion needs to be satisfied to trigger one state transition.

At second, we consider the data structure of output module. According to the above definitions, the output to each machine is defined for each state. The relation between a state and machines is one-to-many because more than one machine needs to be controlled by sequence control. Since a machine has multiple control output ports (for control signals), the relation between a machine and control output ports is also one-to-many. Again, a machine and sensors attached to it are one-to-many relation. The data structure of sequence control module according to the above is shown in Fig.3.

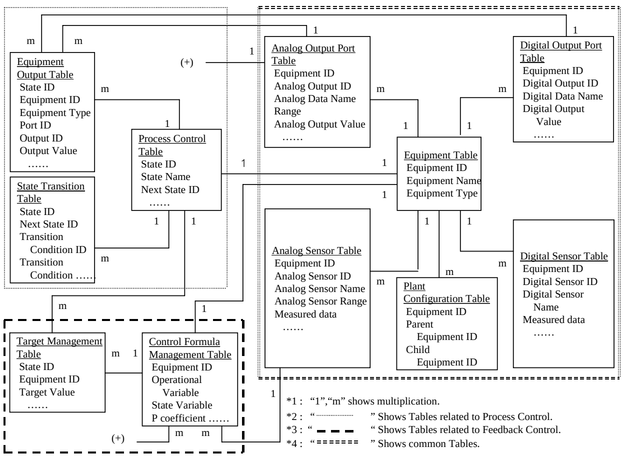


Fig.3 PCS data structure – extracted –

```
//Data Definition Part
define data: machine data
//Machine Table Data  ( Machine Name )
“Transporter"
        ………………..
define data: measured data
//Sensor Data List (Machine Name, Measured Data Name, Data Type, Data Unit )
“Transporter",“Transporter Position”,“Analog","mm"
“Transporter",“Transporter Power Unit“,“Digital","on=1/off=0"
        ………………..
define data: output port data
//Output port List ( Machine Name, Output Port Name, Data Type, Data Unit )
“Transporter",“Transporter Motor RPM“,“Analog",“RPM”
“Transporter",“Transporter Power Unit“,“Digital","on=1/off=0"
        ………………..
define data: sequence control output data
 //Sequence Output Data ( State ID,Machine Name, Output Port, Output Value )
0,“Transporter",“Transporter Motor RPM",0.0
0,“Transporter",“Transporter Power Unit",0
        ………………..
//Function Definition Part
define function: analog output
//Specification of Analog Output Management
“Get  Output Port Name and Output Value that is shown at Machine ID and State ID."
“Output Output Value through Output Prot that is shown Machine ID, State ID and Output Port Name.”
```

Fig.4 Sample of requirement specification in Z (sequence)

```
//Data Definition Part

State ID={0, 5, 10….}                                      //Definition of State ID

Machine Name={Transporter, Pump,…..}                       //Definition of Machine Name

Machine ID={1001, 1002,…..}                                //Definition of Machine ID

Output Port Name={Transporter Motor RPM, ….}               //Definition of Output Port

Output Port ID={101, 102,…..}                              //Definition of Output Port ID

Analog Output =(State ID x Machine ID x Output Port ID)    //Definition of
          |->Output Value                                  //Analog Output Data Type

Analog output={(0, 1001, 101)|->0.0, (0, 1002, 102)|-> 0.0,…..} //Definition of Analog Output


//Function Definition Part
-- Analog Output

//Definition of Analog Variables
State ID?, Analog Machine ID?, Analog Output?, Output Value!

--------------------------------
//Prediction of Analog Output Management

State ID?=0 and Analog Machine ID?=1001 and Output Port ID?=101
                    =>Output Value! = Analog Output

          ………………………………………………………………………..
--------------------------------
```

Fig.5 Requirement specification using template in Z (sequence)

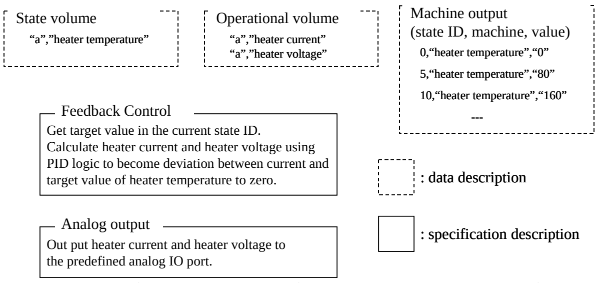
At third, we consider the functions of sequence control module. The required functions are as follows:

(i) analog and digital measurement functions to read the values from sensors attached to machines,

(ii) process management function to verify if transition criteria are satisfied and to decide the state after the transition, using current state ID as a key, and

(iii) analog and digital data control functions which determine the values to be output to each machine, using current state ID as a key.

Fig.4 shows an example of these functions described in Z language. The analog sensor value in data definition section is the value measured from sensor and it is variable. The other values are constants since they are determined uniquely depending on the process or the machine configuration. The schema such as sequence control or analog control section is fixed for every plant. Therefore, if a template in Z language is described in advance, PCS requirement specification of sequence control module can be created only by changing the sensor value and the constants. Fig.5 shows the resulted output using this tool.



Get target value in the current state ID.
Calculate heater current and heater voltage using PID logic to become deviation between current and target value of heater temperature to zero.

Out put heater current and heater voltage to the predefined analog IO port.

Fig.6 Sample of requirement specification in natural language (feedback)

```
//Definitions of state Variables and Operational Variables of Each Machine for Feedback Control

define data: machine data
//Machine Table Data(Machine Name)
“Heater”
          ……….
define data: measured data
//State Variables(Machine Name, Sensor Name, Data Type, Data Unit)
“Heater”,“Heater Temperature”,“analog”, “degree”
          ……….
define data: output port data
//Operational Variables(Machine Name, Output Port Name, Data Type, Data Unit)
“Heater”,“Heater Current”, “Analog”, “Ampere”
“Heater”,“Heater Voltage“, “Analog”, ”Volt”
          ……….
define data: feedback output data
//Target Value (State ID, Machine Name, Output Port Name,Target Value)
0, “Heater”, “Heater Temperature”, 30.0
          ……….

//Description of Control Method
define function: Feedback Control Output
//Specification of Feedback Control
“Get Target Value that is shown at Machine ID, State ID and Output Port Name”
“Calculate Heater Current and Voltage in order to make Difference between State Value and Target Value”
“Output Heater Current and Heater Voltage through the Output Port
                              that is shown at Machine ID and State ID”
```

Fig.7 Sample of requirement specification in Z (feedback)

3.2 Requirement Specification of Feedback Control Module in Z Language

At the phase of creating requirement specification for PCSs, it is important to efficiently define state variables, operational variables, and outline of control method. Fig.6 shows the requirement specification of feedback control module, which is created using software components. This specification consists of two sections: the definitions of state variables and operational variables of each machine for feedback control, and the description of control method. At this phase, it is difficult to further clarify the specification since the detailed control method is not determined yet. Therefore, this specification is described as schema in Z language as it is. Fig.7 shows the result. Fig.8 shows the resulted requirement specification of feedback control module using this tool.

4. Preliminary and detailed design method for PCS

This chapter describes a method to create PCS preliminary and detailed designs based on the PCS requirement specification in Z language. At this phase, the additional information required for PCS development are defined and added to the requirement specification. By referring one of the representative software development standards, "NASDA Software Development Standard (below, NASDA Standard. Now, NASDA is called JAXA.)"[7], we discuss what kind of information need to be additionally defined and in which sequence. NASDA Standard divides development in five phases: requirement specification, preliminary design, interface specification, detailed design, and database file specification. Table 1 shows the outline of NASDA Standard.

The development standard of our proposed method ("the proposed standard") differs from NASDA Standard in the following two ways.

Table 1 Outline of NASDA standard for software development

Development phase	Developed document	Contents which have to decide in document
Requirement definition	Requirement specification	System operational procedure, required functions, size, processing time
Preliminary design	Preliminary design specification	Functional configuration, state transition, data and control flow, global variables, functional design, outline of input/output data, assignment of size and processing time
	Interface design	Input/output data, input/output interface
Detailed design	Detailed design specification	Module configuration, detailed data and control flow, module design, detailed input/output
	Database design specification	Outline of data file, data model

- The proposed standard combines preliminary design and interface specification into one document, while NASDA Standard creates them separately at the preliminary design phase.

- The proposed standard creates database definition at requirement specification phase, while NASDA Standard creates it at the detailed design phase.

The development standard of our proposed method differs from NASDA Standard in the following two ways.

- The proposed standard combines preliminary design and interface specification into one document, while NASDA Standard creates them separately at the preliminary design phase.
- The proposed standard creates database definition at requirement specification phase, while NASDA Standard creates it at the detailed design phase.

In the proposed standard, the following two tasks are carried out at preliminary design phase.

[Preliminary 1]:
Clarify motion functions described in requirement specification.

[Preliminary 2]:
Clarify input/output data of each function.

The following two tasks are carried out at detailed design phase.

[Detail 1]:
Divide functions described in preliminary design.

[Detail 2]:
Design input/output data physically.

The following sections discuss methods to create preliminary and detailed designs.

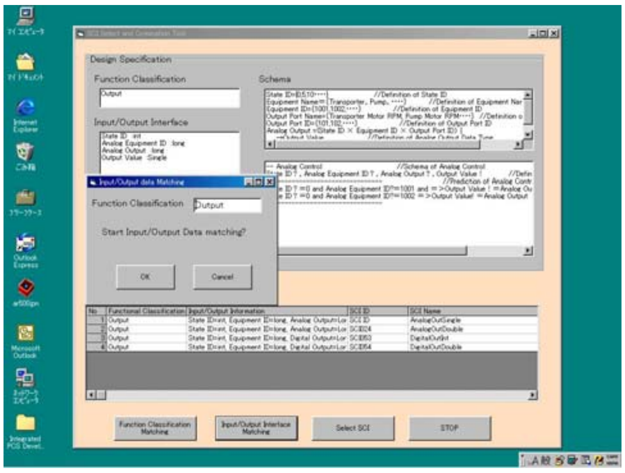
4.1 Preliminary and Detailed Design Tools for Sequence control Module

This section discusses a method to create preliminary and detailed designs of sequence control module.

The sequence control module performs motion control output during the sequence predefined for a plant. The contents of the sequence, such as states, transition criteria, and motion control output, are already described in requirement specification phase, which means no preliminary design task is required for sequence control module. In detailed design phase, the following tasks should be carried out according to the above stated definitions.

[Detail 1]:
Divide a plant into sub-plants, which are the actual units to be controlled. Then divide the sequences (states, transition criteria, and motion control output of entire plant) and assign them to each sub-plant accordingly. The dependent relations must be assigned between the sub-plants and the entire plant (for example, Sub-plant A1 belongs to Plant A). Fig.9 shows sequence data input screen using SCDT (explained below).

[Detail 2]:
Describe the valid digits and the port output format of output values. Those should be clarified through the preceding design work.


Design Specification
Function Classification
Output
Input/Output Interface
Schema
Input/Output data Matching
Start Input/Output Data matching?
OK
Cancel
Function Classification Matching
Input/Output Interface Matching
Select SCI
STOP

```
State ID : int
Analog Equipment ID : long
Analog Output : long
Output Value : Single
```

```
State ID={0,5,10・・・}                          //Definition of State ID
Equipment Name=={Transporter, Pump, ・・・}      //Definition of Equipment Nar
Equipment ID={1001,1002,・・・}                 //Definition of Equipment ID
Output Port Name={Transporter Motor RPM, Pump Motor RPM・・・}  //Definition o
Output Port ID={101,102,・・・}                 //Definition of Output Port ID
Analog Output ≔State ID × Equipment ID × Output Port ID) {
```

```
-- Analog Control                         //Schema of Analog Control
State ID ? , Analog Equipment ID ? , Analog Output ? , Output Value !     //Defin
                                          //Prediction of Analog Contr
State ID ? =0 and Analog Equipment ID?=1001 and =>Output Value ! =Analog Ou
State ID ? =0 and Analog Equipment ID?=1002 =>Output Value! =Analog Output
```

No	Functional Classification	Input/Output Information	SCI ID	SCI Name
1	Output	State ID=int, Equipment ID=long, Analog Output=Lor	SCI ID	AnalogOutSingle
2	Output	State ID=int, Equipment ID=long, Analog Output=Lor	SCID24	AnalogOutDouble
3	Output	State ID=int, Equipment ID=long, Digital Output=Lor	SCID53	DigitalOutInt
4	Output	State ID=int, Equipment ID=long, Digital Output=Lor	SCID54	DigitalOutDouble

Fig.9 Sequence input screen using SCDT

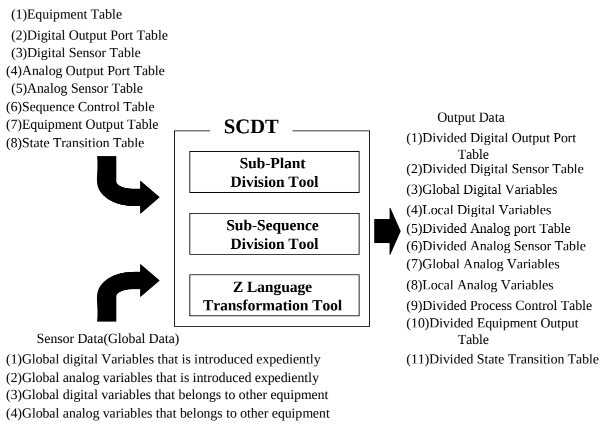

(1)Equipment Table
(2)Digital Output Port Table
(3)Digital Sensor Table
(4)Analog Output Port Table
(5)Analog Sensor Table
(6)Sequence Control Table
(7)Equipment Output Table
(8)State Transition Table
SCDT
Sub-Plant Division Tool
Sub-Sequence Division Tool
Z Language Transformation Tool
Output Data
(1)Divided Digital Output Port Table
(2)Divided Digital Sensor Table
(3)Global Digital Variables
(4)Local Digital Variables
(5)Divided Analog port Table
(6)Divided Analog Sensor Table
(7)Global Analog Variables
(8)Local Analog Variables
(9)Divided Process Control Table
(10)Divided Equipment Output Table
(11)Divided State Transition Table
Sensor Data(Global Data)
(1)Global digital Variables that is introduced expediently
(2)Global analog variables that is introduced expediently
(3)Global digital variables that belongs to other equipment
(4)Global analog variables that belongs to other equipment

Fig.10 Outline Of SCDT

We have developed Sequence control Design Tool (SCDT) based on the above tasks. Fig.10 shows the outline of SCDT. SCDT divides a plant into sub-plants and assigns dependent relations to them, using machine configuration table (included in PCS requirement specification in Z language). Based on the defined sub-plant configuration, SCDT divides states, transition criteria, and motion control output table for each sub-plant. Finally, SCDT converts the created design into Z language and output it as PCS implementation specification.

4.2 Preliminary and Detailed Design Tools for Feedback Control Module

This section discusses a method to create preliminary and detailed designs of feedback control module.

Feedback control module calculates motion controls (operational volume) and output them to machines. The operational volume is calculated based on the sensor values (state volume) of the machines and its targeted control value[8]. In most cases, a formula to calculate motion controls is not determined at requirement definition phase. Therefore, the specification is determined at preliminary and detailed design phases. The following tasks should be carried out according to the above stated definitions.

[Preliminary 1]:

Define the functional outline of motion control formula by describing calculation method with control block diagrams such as proportional, differential, and integral. Fig.11 shows motion control formula input screen using FCDT (explained below).

[Preliminary 2]:

Describe additional information regarding operational volume and state volume which are clarified through the preceding design work.

[Detail 1]:

Describe detailed functions required for PCS development, such as delays, filters, and saturation, and add them to the control block diagrams created in [Preliminary 1]. Then define data transmission between block diagrams.

[Detail 2]:

Describe the valid digits and the port output format of output values. Those should be clarified through the preceding design work.

We have developed Feedback Control Design Tool (FCDT) based on the above tasks. Fig.12 shows the outline of FCDT. Based on input/output data of formal requirement specification and motion control output formulas, FCDT describes motion control formulas using feedback control components (block diagrams). The block diagrams become more sophisticated by detailing them repeatedly. Finally, FCDT converts the created design into Z language and output it as PCS implementation specification.

5. Software Components Selection Method

In our proposed method, PCSs are developed by combining software components selected from detailed specification in Z language. The well-known software components selection methods are Faceted Classification[9] and Specification Matching[10), 11)].

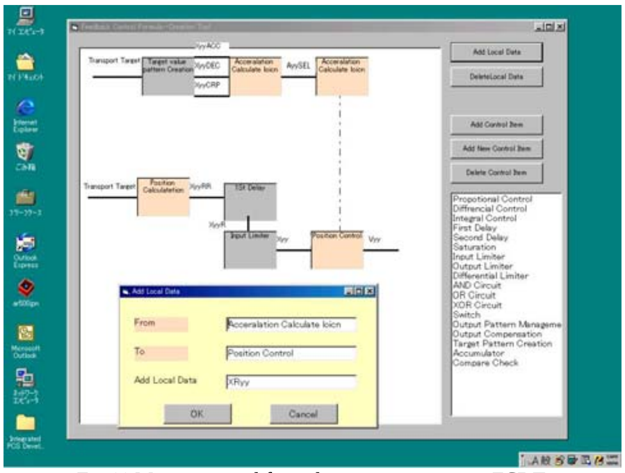


Fig.11 Motion control formula input screen using FCDT

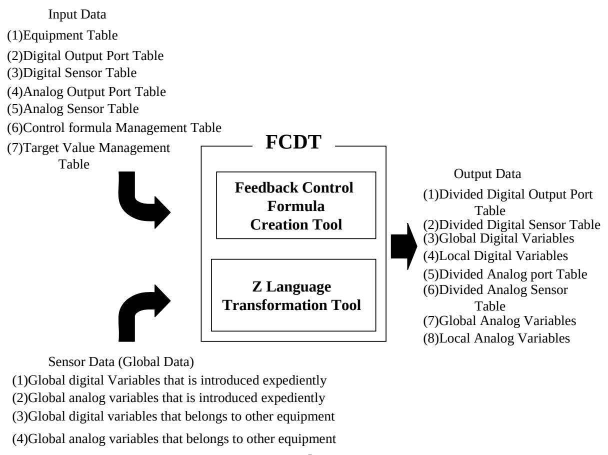

Input Data
(1)Equipment Table
(2)Digital Output Port Table
(3)Digital Sensor Table
(4)Analog Output Port Table
(5)Analog Sensor Table
(6)Control formula Management Table
(7)Target Value Management Table
FCDT
Feedback Control Formula Creation Tool
Z Language Transformation Tool
Output Data
(1)Divided Digital Output Port Table
(2)Divided Digital Sensor Table
(3)Global Digital Variables
(4)Local Digital Variables
(5)Divided Analog port Table
(6)Divided Analog Sensor Table
(7)Global Analog Variables
(8)Local Analog Variables
Sensor Data (Global Data)
(1)Global digital Variables that is introduced expediently
(2)Global analog variables that is introduced expediently
(3)Global digital variables that belongs to other equipment
(4)Global analog variables that belongs to other equipment

Fig.12 Outline FCDT

Our method selects and combines software components based on:

- input/output interface specification of software components, and
- function classification added by users.

5.1 Software components selection algorithm

This section explains about component lists (shown on Table 2) which manage software components (shown on Figure 13), and then describes the algorithm for selecting software components.

Table 2 Outline of software component list

Functio -nal classify -cation	Input/ Output IF	Component		Functional explanation
		ID	Name	
OS	IN; Count: int	OS1	OsCycle	Accumulate clocks counts, and create defined control cycle.
OS	IN; Seq: char OUT; ET: real	OS2	OsExe	Execute software components according to the predefined
---	---	---	---	---

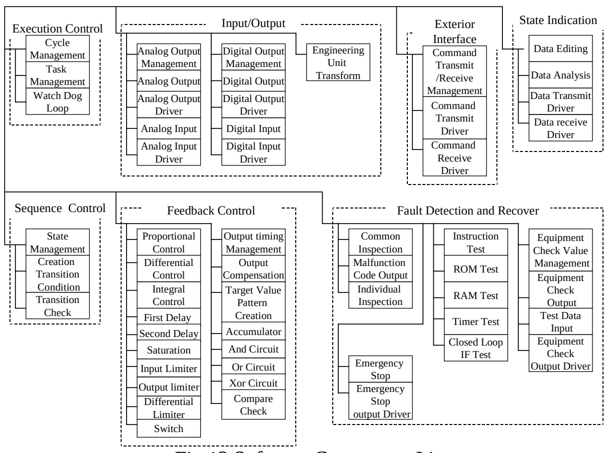
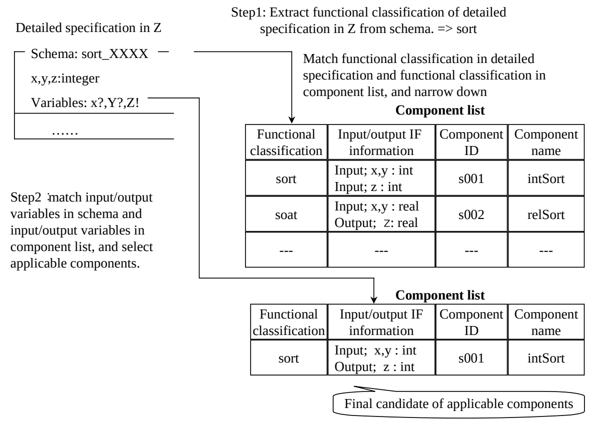


Fig.13 Software Component List


Detailed specification in Z
Schema: sort_XXXX
x,y,z:integer
Variables: x?,Y?,Z!
……
Step1: Extract functional classification of detailed specification in Z from schema. => sort
Match functional classification in detailed specification and functional classification in component list, and narrow down
Step2 :match input/output variables in schema and input/output variables in component list, and select applicable components.
Final candidate of applicable components

Component list

Functional classification	Input/output IF information	Component ID	Component name
sort	Input; x,y : int Input; z : int	s001	intSort
soat	Input; x,y : real Output; z: real	s002	relSort
---	---	---	---

Component list

Functional classification	Input/output IF information	Component ID	Component name
sort	Input; x,y : int Output; z : int	s001	intSort

Fig.14 Software components selection method

Component lists contain input/output interface information and function classification information. The input/output interface information is composed using argument information of software components provided by the authors. The input/output data and their data types are listed in the interface information column. The function classification column contains the functional outlines of software components. Users can select this information from classification candidates and add it to the component list when registering software components to the list.

software components are selected in the following steps (see Fig.14):

[Step 1]:

Decide best matching function classification for each schema in PCS detailed specification, and match them with function classification on the component list. Then every software components which belongs to the matched function classification are listed.

[Step 2]:

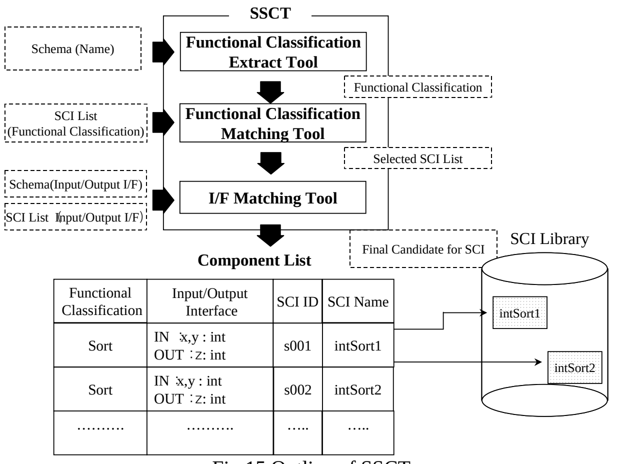
Narrow down the candidates of software components by matching input/output interface on the component list and on the detailed specification. This matching uses information such as number of variables or data types.

[Step 3]:

When more than one software components is found, all of them are listed and a user selects an appropriate one considering performance and data area size. When no software component is found, the schema part of detailed specification must be reviewed.

5.2 Software component selection and combination Tool

Fig.15 shows the outline of software Component Selection and Combination Tool (SSCT).


SSCT
Schema (Name)
Functional Classification Extract Tool
Functional Classification
SCI List (Functional Classification)
Functional Classification Matching Tool
Selected SCI List
Schema(Input/Output I/F)
SCI List (Input/Output I/F)
I/F Matching Tool
Final Candidate for SCI
SCI Library
intSort1
intSort2
Component List

Functional Classification	Input/Output Interface	SCI ID	SCI Name
Sort	IN :x,y : int OUT :z: int	s001	intSort1
Sort	IN :x,y : int OUT :z: int	s002	intSort2
……….	……….	…..	…..

Fig.15 Outline of SSCT

SSCT selects an appropriate function classification from component list using schema in detailed specifications. We have established the naming rules for schema which enable users to distinguish unknown functions. We also created the thesaurus of function classification to improve an efficiency of software component selection. This thesaurus enables users to select similar components even if no component matched exactly. SSCT then selects an appropriate component by matching interfaces. When more than one candidate is found, SSCT lists all of them and allows user to select one.

6. Application and evaluation of Integrated PCS development environment

This chapter describes and evaluates the results of applying Integrated PCS Development Environment (IPDE), which combines SCDT, FCDT, and SSCT, to actual PCS developments.

We have applied the IPDE to five plant developments (Plat A to E). Table 3 shows the number of applied software components, Table 4 shows the reused rate of source codes, and Table 5 shows the development time [hours] of each development. The reused rate and the development period are defined as below[12), 14)]:

Table 3 Number of existing and newly developed software components

Plant Name	Components of sequence control type sub-plants [pieces]		Components of Feedback control type sub-plants [pieces]		Components of other parts [pieces]		RR_ CM [%]
	Reus -ed	New -ly	Reus -ed	New -ly	Reus -ed	New -ly	
A	34	31	49	11	8	5	66
B	24	22	27	8	8	3	64
C	17	16	124	29	8	3	76
D	17	15	51	13	8	5	69
E	44	40	34	15	8	7	58

Table 4 reused source code [LOC] of each development

Plant Name	LOC of sequence control type sub-plants [LOC]		LOC of Feedback control type sub-plants [LOC]		LOC of other parts [pieces]		RR_ CD [%]
	Reus -ed	New -ly	Reus -ed	New -ly	Reus -ed	New -ly	
A	6907	3917	2418	897	449	105	67
B	7108	4554	1574	731	449	81	63
C	4159	2740	6106	3101	449	465	63
D	5016	2501	2346	1297	449	282	65
E	8688	5061	3604	1528	449	303	65

Table 5 development time [hours] of each development

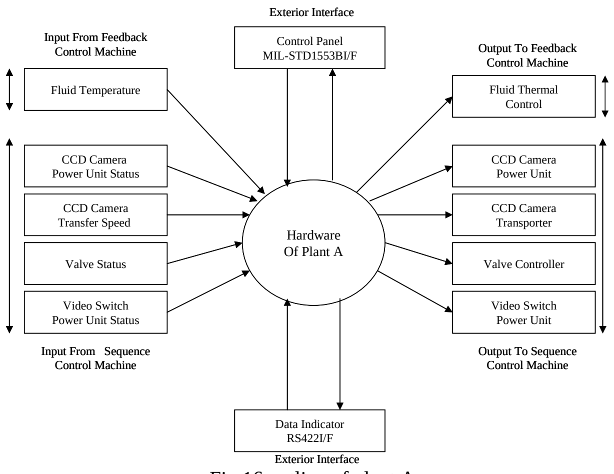
Plant name	Development time in established method	Development time in proposed method	RR_TM [%]
A	548	222	60
B	720	323	55
C	396	163	59
D	387	163	58
E	751	315	58

$$RR\_CM = 100 \times \frac{Nrcm}{Nrcm + Nncm} \quad (1)$$

$$RR\_CD = 100 \times \frac{Nrloc}{Nrloc + Nnloc} \quad (2)$$

$$RR\_DT = 100 \times \left(1 - \frac{DTppm}{DTesm}\right) \quad (3)$$

RR_CM:	Reused rate in Components.
Nrcm:	Number of reused components
Nncm:	Number of newly developed components
RR_CD:	Reused rate in LOC (Lines Of Codes)
Nrloc:	Number of reused LOC
Nnloc:	Number of newly developed LOC
RR_DT:	Reduced rate in Development time
DTppm:	Development time (requirement definition, preliminary design, detailed design, programming, verification) in proposed method
DTesm:	Development time (requirement definition, preliminary design, detailed design, programming, verification) in established method


Exterior Interface
Control Panel MIL-STD1553BI/F
Input From Feedback Control Machine
Output To Feedback Control Machine
Fluid Temperature
Fluid Thermal Control
CCD Camera Power Unit Status
CCD Camera Power Unit
CCD Camera Transfer Speed
CCD Camera Transporter
Valve Status
Valve Controller
Video Switch Power Unit Status
Video Switch Power Unit
Hardware Of Plant A
Input From Sequence Control Machine
Output To Sequence Control Machine
Data Indicator RS422I/F
Exterior Interface
Fig.16 outline of plant A

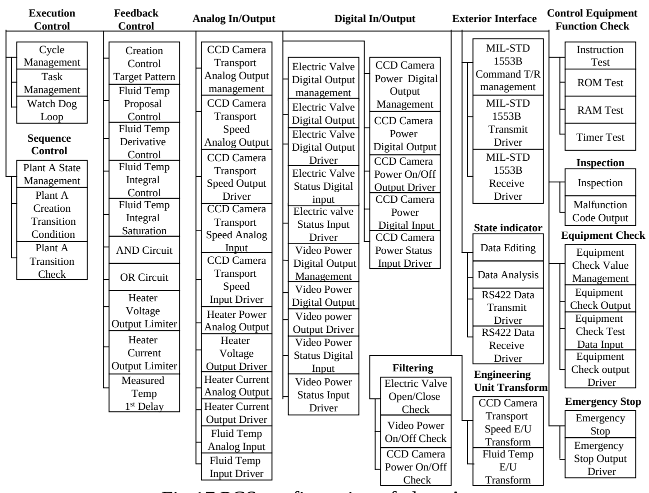

Execution Control
Cycle Management
Task Management
Watch Dog Loop
Sequence Control
Plant A State Management
Plant A Creation Transition Condition
Plant A Transition Check
Feedback Control
Creation Control Target Pattern
Fluid Temp Proposal Control
Fluid Temp Derivative Control
Fluid Temp Integral Control
Fluid Temp Integral Saturation
AND Circuit
OR Circuit
Heater Voltage Output Limiter
Heater Current Output Limiter
Measured Temp 1st Delay
Analog In/Output
CCD Camera Transport Analog Output management
CCD Camera Transport Speed Analog Output
CCD Camera Transport Speed Output Driver
CCD Camera Transport Speed Analog Input
CCD Camera Transport Speed Input Driver
Heater Power Analog Output
Heater Voltage Output Driver
Heater Current Analog Output
Heater Current Output Driver
Fluid Temp Analog Input
Fluid Temp Input Driver
Digital In/Output
Electric Valve Digital Output management
Electric Valve Digital Output
Electric Valve Digital Output Driver
Electric Valve Status Digital input
Electric valve Status Input Driver
Video Power Digital Output Management
Video Power Digital Output
Video power Output Driver
Video Power Status Digital Input
Video Power Status Input Driver
CCD Camera Power Digital Output Management
CCD Camera Power Digital Output
CCD Camera Power On/Off Output Driver
CCD Camera Power Digital Input
CCD Camera Power Status Input Driver
Filtering
Electric Valve Open/Close Check
Video Power On/Off Check
CCD Camera Power On/Off Check
Exterior Interface
MIL-STD 1553B Command T/R management
MIL-STD 1553B Transmit Driver
MIL-STD 1553B Receive Driver
State indicator
Data Editing
Data Analysis
RS422 Data Transmit Driver
RS422 Data Receive Driver
Engineering Unit Transform
CCD Camera Transport Speed E/U Transform
Fluid Temp E/U Transform
Control Equipment Function Check
Instruction Test
ROM Test
RAM Test
Timer Test
Inspection
Inspection
Malfunction Code Output
Equipment Check
Equipment Check Value Management
Equipment Check Output
Equipment Check Test Data Input
Equipment Check output Driver
Emergency Stop
Emergency Stop
Emergency Stop Output Driver
Fig.17 PCS configuration of plant A

Fig.16 shows the hardware configuration of plant A, and Fig.17 shows the PCS configuration of plant A as an example.

6.1 Evaluation of Reused Rate and Lines Of Codes (LOC)

We evaluate the reused components, reused LOC and reduced time in each sub-plant. This is because each sub-plant is separately developed in PCS developments and the scales of PCSs are proportional to the number of sub-plants. Additionally, we discuss sequence control type (S-type) and feedback control type (F-type) of sub-plants separately.

S-type PCS is composed of sequence and measurement modules. The sequence module consists of sequence management, output value management, and output drivers. And the measurement module consists of sensor measurement, engineering value conversion, and input drivers. The functions such as sequence management, output value management, sensor measurement, and engineering value conversion are implemented with software components. Input/output drivers must be developed because they depend on input/output interface of control equipment. Sequence management contains sequence data, and output value management contains output value data, which are used for customization. One line of sequence management data represents one state as a set of a state, transition criteria, and a state after the transition. One line of output value management data represents values to be output to sub-plants in one state.

The number of software components in S-type PCS is calculated as below. Σ represents a sum total of all sub-plants.

$$\begin{aligned} TNsc &= \sum (Nsc + Nsco + (Nscm + Nsci) \times Nsmp) \\ &= \sum (3 + 2 \times Nsmp) \end{aligned} \qquad (4)$$

TNsc:	Total number of S-type software components
Nsc:	Number of S-type software components
Nsco:	Number of S-type output driver components
Nscm:	Number of S-type software components for measurement
Nsci:	Number of S-type input driver components for measurement
Nsmp:	Number of S-type measurement points

Accordingly, the total LOC for S-type PCS is calculated as below. The LOC for input/output drivers are mean values of Plant A to E.

$$\begin{aligned} TLs &= \sum ((Lslc + Lslo) + (Lslm + Lsli) \times Nsmp \\ &\qquad + (Lslp + Lsid) \times Nsls + Lsla) \\ &= \sum ((562 + 86) + (104 + 70) \times Nsmp + 2 \times Nsls + 51) \\ &= \sum (699 + 174 \times Nsmp + 2 \times Nsls) \end{aligned} \qquad (5)$$

TLs:	Total LOC for S-type software components
Lslc:	LOC for S-type software components
Lslo:	LOC for S-type output drivers
Lslm:	LOC for S-type output software components for measurement
Lsli:	LOC for S-type input drivers
Nsmp:	Number of S-type measurement points
Lslp:	LOC for sequence management data per one state
Lsid:	LOC for output measurement data per one state
Nsls:	Number of state transitions
Lsla:	LOC appended or modified

Based on (5), the calculated numbers of codes in Plant A to E are 11030 [LOC], 11911 [LOC], 7028 [LOC], 7388 [LOC], and 14242 [LOC] respectively. The errors are within 5 [%] between the calculated values and the actual values shown in Table 4. This is because that the total LOC for software components can be calculated accurately, and that the total LOC for newly developed portion is small and its estimation error is around 15 [%]. Therefore, the total LOC for S-type PCS is estimated based on (5).

F-type PCS is composed of modules such as proportional, differential, integral, delay, limiter, switch, output timing management, measurement, AND, OR, input drivers, and output drivers. The twenty types of modules such as proportional, differential, integral, delay, limiter, switch, AND, OR, output timing management, and measurement are implemented with software components. Input/output drivers must be developed because they depend on control methods and input/output interface of control equipment. F-type PCSs use similar set of software components, although the components are used in different order to accomplish appropriate control. F-type PCSs contain data (coefficients and time constants) used for customizing PID control, however, they can be ignored since the data volume is small as compared to the total LOC for software components and newly developed portion. The number of software components and LOC in F-type PCS are calculated as below. Σ represents a sum total of all sub-plants.

$$\begin{aligned} TNfc &= \sum (Nfc + Nfco + (Nfcm + Nfci) \times Nfmp) \\ &= \sum (21 + 2 \times Nfmp) \end{aligned} \qquad (6)$$

TNfc:	Total number of F-type software components
Nfc:	Number of F-type software components

Nfco:	Number of F-type output driver components
Nfcm:	Number of F-type software components for measurement
Nfci:	Number of F-type input driver components
Nfmp:	Number of F-type measurement points

$$TLf = \sum((Lflc + Lflo) + (Lflm + Lfli) \times Nfmp + Lfla)$$
$$= \sum((726 + 79) + (104 + 70) \times Nsmp + 81)$$
$$= \sum(886 + 174 \times Nfmp) \qquad (7)$$

TL f:	Total LOC for F-type software components
Lflc:	Number of codes for F-type software components
Lflo:	Number of codes for F-type output drivers
Lflm:	LOC for F-type software components for measurement
Lfli:	LOC for F-type input drivers
Nfmp:	Number of F-type measurement points
Lfla:	LOC appended or modified

Based on (7), the calculated numbers of codes in Plant A to E are 3192 [LOC], 2185 [LOC], 8766 [LOC], 3538 [LOC], and 4914 [LOC] respectively. The errors are within 5 [%] between the calculated values and the actual values shown in Table 4. This is because that the total LOC for software components can be calculated accurately, and that the total LOC for newly developed portion is small and its error estimation is around 15 [%].

The other parts are composed of cycle management, task management, control equipment operation check, machine operation check, abnormal interrupts, and emergency machine stop. The functions such as cycle management, task management, and control equipment operation check are implemented with software components. Machine operation check, abnormal interrupts, and emergency machine stop are the functions inherent in each plant. Because the LOC for these inherent functions cannot be determined, we estimate it as mean value of Plant A to E, which is 247 [LOC]. The total LOC for the other parts is calculated as below:

$$TLo = TLolc + Tloli = 449 + 247 = 696 \qquad (8)$$

Tlo:	Total LOC for other parts
TLolc:	Total LOC for the other software components
TLols:	Total LOC for the other inherent parts of a plant.

Based on Formula (5), (7) and (8), the total LOC for a PCS is calculated as below.

$$TLp = TLs + TLf + TLo$$
$$= \sum(699 + 174 \times Nsmp + 2.0 \times Nsls)$$
$$+ \sum(886 + 174 \times Nfmp) + 696 \qquad (9)$$

TLp:	Total LOC for PCS

Based on (9), the calculated total LOC in Plant A to E are 11068 [LOC], 7028 [LOC], 11911 [LOC], 7339 [LOC], and 14357 [LOC] respectively. The errors are within 10 [%] between the calculated values and the actual values shown in Table 4. This result indicates that (9) is applicable to estimate total LOC for PCSs.

Based on (2), the reused rate in Plant A to E are 67 [%], 64 [%], 63 [%], 65 [%], and 65 [%] respectively, and the average is 65 [%]. In here, input/output drivers are treated as new developments. The reused rate should be improved if we register frequently used drivers to the component list.
As the result of Table 5, reduced rate of development time in Plant A to E are 60[%], 59[%], 55[%], 58[%], and 58[%] respectively, and the average is 58[%]. Applying proposed method reduces PCS development time. This strengthens PCS developer’s competitiveness.

6.2 Evaluation of PCS development environment

6.2.1 Evaluation of Tools to Describe PCS Requirement Specification in Z Language

- Eliminating ambiguity using Z language

We have described requirement specifications in Z language. This reduces ambiguity in the descriptions and enables precise transmission of requirement specification information to preliminary and detailed design phase. As a result, garbling or back track of tasks has been reduced, and shorter development period have been achieved.

6.2.2 Evaluation of sequence control design tool

- Dividing plant into sub-plants

We have divided an entire plant into sub-plants using SCDT. This limits the range controlled by one component.

- Hierarchical state transition

Description of state transition (sequences) has been facilitated by defining them separately for each sub-plant. Also, description of coordinated sequences has been facilitated by defining state transitions hierarchically between plant and sub-plants. As a result, sequence control has been simplified and easier verification has been enabled.

6.2.3 Evaluation of feedback control design tool

● Dividing plant into sub-plants

FCDT also has prevented controls from being too complex and has enabled the development of reliable PCSs.

● Dividing motion control functions

FCDT divides schema in implementation specification repeatedly until it matches to functions registered on component list. This enables less experienced developers to determine how far the functions should be divided.

6.2.4 Evaluation of software component selection and combination tool

● Automatic extraction of appropriate components

SSCT extracts the candidates of software components automatically. This has extensively saved labor to search applicable components from software component libraries.

● Refine specification through repetition of selecting software components

When no applicable component is found, SSCT allows developers to divide functions further and to search for applicable components repeatedly. When no applicable component is found after the repetition, then new programs need to be developed. However, new programs are developed much easier since the functions and the input/output interfaces of the programs are clarified sufficiently after the repeated division.

7. Conclusion and future works

This paper has proposed a method for seamless PCS development from requirement definition through implementation. The results of applying IPDE to actual PCS developments have indicated that this method is highly effective for seamless PCS development. The reused rate of pre-defined software components was 65 [%]. Consequently, only inherent functions in each plant, such as input/output drivers and abnormal sequences, need to be newly developed. The reused rate shall be improved by increasing types of software components and registering developed programs to libraries as software components. The reduced rate of development time was 58[%]. Consequently, this proposed method enables us to develop PCS effectively.

In this method, we have adopted Z language to standardize the interfaces of specifications described in each development phase and to transmit information precisely between the phases. This has enabled to use different tools for each development phase, and the flexible PCS development environment has been accomplished.

As a future work, we will improve operability of PCS development environment. We will also reduce labor to manage and maintain the environment by implementing methods or tools to create component lists automatically based on information of black-box components.

Acknowledgments

This research is conducted as a part of "Preliminary Technology for Next Generation Transportation System Design", which is a delegation from New Energy and Industrial Technology Development Organization (NEDO).

References

[1] Matsumoto M. et. al. : Specifications reuse process modeling and CASE study-based Evaluations, COMPSAC91, Proc. of the 15’th annual international computer software & applications (1991).

[2] Natori M., Kagaya S. and Honiden S.: Reuse of Requirements Specification Based on Domain Analysis, Information Processing Society of Japan, No.37, Vol.3, 393/408(1996). (In Japanese)

[3] Takahashi M. and Tsuda K.: The Efficient Method of Plant Software Requirement Definition, Information Processing Society of Japan, No.42, Vol.3, pp.518-528（2001）. (In japanese)

[4] Norcliffe A. and Slater G.: Mathematics of Software Construction, Ellis Horwood（1991）.

[5] Kawakita M., Sakai M., Yamamoto S. and Agusa S.: A Model for Reuse Based on Formal Specifications, Information Processing Society of Japan, Vol.36, No.5, pp.1050-1058 (1995). (In Japanese)

[6] Kobayashi H., Kawata Y., Maekawa M. Kawasaki A., Yabu A. and Onogawa K.: Modeling External Objects of Process Control Systems in Executable Specifications, Information Processing Society of Japan, Vol.35, No.7, pp.1402-1409 (1994). (In Japanese)

[7] National Space Development Agency of Japan (NASDA, JAXA at present): NASDA Parts Application Standard（NDC-1-9-1）, NASDA(1996). (In Japanese)

[8] Teshima S., Inamori Y. and Agusa K: EPS ProgramModel for Embedded Real-Time Syatem abd EFS-Based CASE Tool Schetch, The Institute of Electronics, Information and Communication Engineers, Vol.80 D-I, No.8, pp.691-702 (1997). (In Japanese)

[9] Prito-Diatz, R: Implementing Faceted Classification for Software Reuse, Communications of ACM, Vol.34, No.5, pp.89-97 (1991).

[10] Jeng,J and Cheng B: Specification Matching for Sotware Reuse: A Foundation, In Proceedings of Software Engineering and Knowledge Engineering, Vol.2, No.4, pp.523-546 (1992).

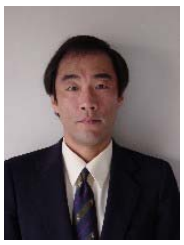
[11] Penix J. and Alexsander P.: Classification and retrieval of reusable componets using semantic feature, In proc. Of 9th knowledge-based software engineering conference, pp.131-138 (1995).

[12] Matsumoto M.: Software Modeling and Reuse, Kyoritsu publishing (1996). (In Japanese)

[13] Richard W.: Enabling Reuse-Based Software Development of Large Scale Systems, IEEE Transactions on Software Engineering, Vol.31, No.6, pp.495-510 (2005).
[14] Jose J. D., A Validation of the Component-Based Method for Software Size Estimation, IEEE Transactions on Software Engineering, Vol.26, No.10, pp.1006-1021 (2000).
[15] Robert B, Dae-Kyoo Kim, Sudipto Ghosh and Eunjee Song, A UML-Based Pattern Specification Technique, EEE Transactions on Software Engineering, Vol.30, No.3, pp.193-206 (2004).

Masakazu Takahashi received his BA in 1988 from Rikkyo University, Japan, his MA degree in 1998 and Ph.D. degree in 2001, both in Systems Management from University of Tsukuba, Japan. He was with Ishikawajima-Harima Heavy Industries Co., Ltd. during 1988 and 2005, and he was assigned to a subsidiary of IHI, and Galaxy Express Corporation during 2002 and 2005. He is an associate professor in department of mathematics and computer science, interdisciplinary faculty of science and engineering, Shimane University. He is a member of The Information Processing Society of Japan, The Society of Instrument and Control Engineers, and The Institute of Electrical Engineers of Japan.

Satoru Takahashi received the B.S. and M.S. from Tokyo University of Science in 1996 and 1998 respectively, and received M.B.A from in 2004. He is a Ph. D. candidate in Graduate School of Systems Management, University of Tsukuba. Since 1998, he has been with Mitsui Asset Trust and Banking Co., Ltd. He is a member of The Information Processing Society of Japan.



Kazuhiko Tsuda received his BA degree in 1986 and Ph. D. degree in 1994, both in engineering and from Tokushima University, Japan. He was with Mitsubishi Electric Corporation during 1986 and 1990, and with Sumitomo Metal Industries Ltd. during 1991 and 1998. He is an associate professor in Graduate School of Business Management, University of Tsukuba, Tokyo, Japan during 1998-2005, professor since 2005. His research interests include natural language processing, database, and human-computer interaction. He is a member of The Information Processing Society of Japan and The Institute of Electrical Engineers of Japan, Information and Communication Engineers.