Paper

An Efficient Integrated Development Environment for Plant Control Software

Masakazu Takahashi* Member
Kazuhiko Tsuda** Member

This paper proposes an efficient environment to develop plant control software (PCS) that controls the space plants for satellites or space stations. Generally, PCSs for space plants have been individually developed because each plant operates in different way and have strict restrictions on types, performances and admeasurements of the hardware to be used. Because of this individual development, the qualities of PCSs heavily depend on the developers' abilities and experiences, and the development period tends to get longer. To solve this problem, we propose an efficient environment that make possible to develop high quality PCSs (1) by developing software components based on domain analyses of PCSs, and (2) by conveying design information accurately throughout the development processes. We have developed a prototype of this environment and applied to the actual developments of PCSs. As a result, we are able to convey precise design information throughout the development processes and have succeeded to develop PCSs that are fully conformable to our requirements. We are also able to predict the program size and the processing time of the PCSs within 10[%] errors at the initial phase of development. Moreover, we have developed most parts of the PCSs using software components. The reuse rate of software components is 65.5[%], and development time is cut down to 42.1[%] in comparison with the developments from scratch.

Keywords:
Software Components, Domain Analysis, Plant, Motion Control Software

1. Introduction

A "plant" is machinery and equipment to manufacture mechanical or chemical products. This paper focuses on space plants for satellites or space stations, and proposes an efficient method to develop motion control software for space plants that are used for material experiments and for controlling environmental equipment. Since space plants need to operate in ultimate environment in outer space, there are various constraints on its hardware [4] and software [5]. And the operation of each plant is different. For these reasons, plant control software (PCS) has been developed individually for each plant. The functionality and the performance of a developed PCS highly depend on the developers' experiences and techniques, and there have been troubles such as protracted development period and increased development cost. To solve these problems and to develop PCSs efficiently, we need to standardize PCSs and their development processes and to enable the development of high quality PCSs without depending on developers' experiences and techniques. Various methods are proposed to address these problems such as Domain Analysis [6], CASE [3] and Framework [1]. However, these methods impose a heavy burden on developers to deeply understand the methodologies, tools and models. The outline of the proposed method is described below:

(1) Develop software components for deciding requirement specification and for implementing PCS, by analyzing PCS domain.
(2) Create the requirement specification, using software components for creation of PCS prototype.
(3) Translate requirement specification to Z language, using similarity of PCS's functions.
(4) By taking an advantage of Z language that allows hierarchical description, develop the requirement specification hierarchically and create design specifications for implementation.
(5) Based on the design specification, select software components for implementation of PCS. In this selection, the lists of function categories and input/output data shall be used.
(6) 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. The IPDE also allows us to predict the program size and the processing time of PCSs within 10[%] errors. The reuse rate of the source code on the software component-based PCS developments is 65.5[%], and the development time is cut down to 42.1[%] in com-

* Galaxy Express Corporation
3-5-1 Mukodaichou, Nishitokyo-shi, Tokyo 188-8555
** Gradudate School of Systems Management, The University of Tsukuba, Tokyo
3-29-1 Otsuka, Bunkyo-ku, Tokyo 112-0012

parison with the developments from scratch. As a result, we have confirmed that IPDE opens the way to efficiently develop highly conformable PCSs.

2. Problems on PCS Development and Solutions

2.1 Actual Situation of PCS Development

There are two kinds of methods to develop PCSs that control plant motion: one uses PLC (Programmable Logic Controller) and another uses PC (including Distributed Control System). IEC61131-3 is a development standard for PCSs that operate on PLC. It is determined based on experiences in development of plant motion control software. However, IEC61131-3 has difficulties such as: (1) it is an implementation language for PCSs and does not support upstream processes of PCS development, and (2) more standardization of feedback control function is required. Furthermore, control devices which implement PCSs must satisfy NASDA standards [4] to be used in peculiar environment like outer space. At the present, no PLC satisfies these requirements and therefore we need to develop PCSs on PC in order to use them in outer space.

2.2 Problems on Development and Solutions

The following factors make it difficult to develop PCSs.

(1) It is difficult to create PCS requirement specification with fully functions.

(2) It is difficult to transmit contents of requirement specification to contents of basic and detailed design.

(3) It is difficult to develop PCS adapting the result of detailed design.

The factor (1) is caused by the fact that PCS is typically developed in individual order basis that is contacted with customer. To eliminate this factor, the developers of a PCS need to fully clarify the requirements for the PCS. The factor (2) and (3) are caused by the ambiguity of design information. It is necessary to develop PCS that design details in preceding processes must be transmitted to subsequent processes accurately. We use Z language [8] [9] [10] which allows formal specification description. In this proposed method, Z language is used only for transmitting information between processes (refer to phase 1 to 3 described below). Z language also allows hierarchical specification description. In this proposed method, requirement specifications in Z language are hierarchically developed into basic and detailed designs. This developing process is repeated until it reaches to granularity of software components for implementation. Based on the detailed design, the software components are selected and combined to implement PCS. To address the above mentioned three problems, this method divides development processes into the following three phases (Fig.1).

Phase1: Creation of Requirement Specification Create requirement specification with fully functions. PCSs require same and similar functions even if the control targets or the control methods are different. [7] Utilizing these characteristics, create fully function list of PCS, then create requirement specification selecting required functions.

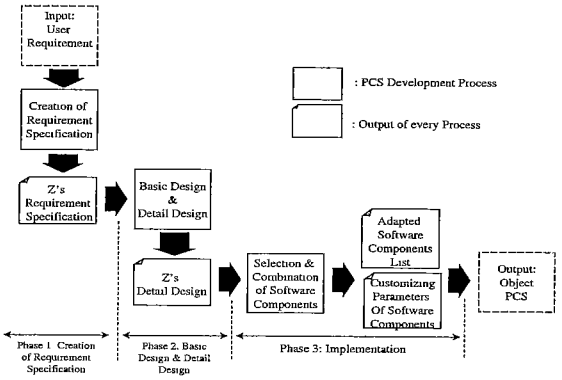


Fig. 1. Outline of the proposed PCS development method.

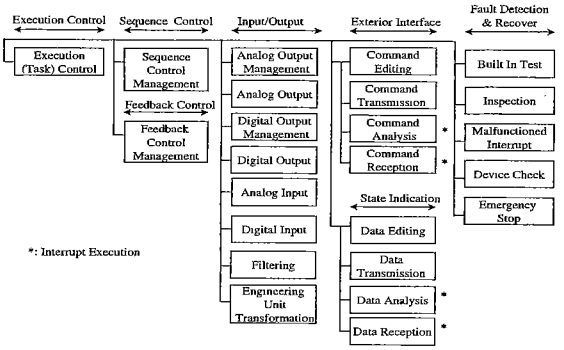


Fig. 2. List of SCR.

Phase2: Basic and Detailed Design Transmit design information accurately. Contents of requirement specification are transmitted to contents of basic design specification by using Z language. And contents of basic design are transmitted to detailed design accurately as it. As a result, detailed design specification is created adapting to the requirement specification.

Phase3: Implementation Implement PCS adapting to detailed design specification. Based on the PCS detailed design specification, applicable software components (for implementation) are selected. Combine selected software components into PCS.

3. Creating Requirement Specification

3.1 Outline of How to Create PCS Requirement Specification

At the phase of creating requirement specification, it is important to decide the functions that individual PCS has. At this phase, it isn't necessary to decide detail of the function. In the proposed method, we develop software components that simulate outline of functions. The prototype of PCS is developed by using these software components. Work the prototype and check the behavior of it. When the behavior of the prototype confirms to user requirement, we create the entire requirement specification of the PCS by combining the requirement specification templates and the customizing parameters for the software components that are prepared in advance. The following subsections

Table 1. list of PCS functions -Extructed-.

PCS parts	function group	detailed function	subject PCSs			
			semiconductor experiment facility on satellite	Thermal control device on space station	strage tank pressure control device	- - -
sequence control group	sequence control	sequence control	X	X	X	- - -
feedback control group	feedback control	feedback control	X	X	X	- - -
common functions	execution control	execution control	X	X	X	- - -
	input/ output	analog output	X	X	X	- - -
		analog input	X	X	X	
		digital output	X	X	X	
		digital input	X	X	X	
		- - -	- - -	- - -	- - -	
	exterior interface	command send		X		- - -
		command receive	X	X	X	
		sent command creation		X		
		received command analysis	X	X	X	
	state indication	data send	X	X	X	- - -
		data receive		X	X	
		sent data edit	X	X	X	
		received data analysis		X	X	
	fault detection and recover	inspection	X	X	X	- - -
		control device check	X	X	X	
		machine check	X		X	
		- - -	- - -	- - -	- - -	

outline the processes to develop the software components and to create the requirement specification.

3.1.1 Outline of Software Components Development We need to clarify the functions of the PCS in order to develop the software components used for creating PCS prototype (hereafter, SCR: Software Components for Requirement). We firstly compared and examined more than twenty PCSs, including the ones for semiconductor production, thermal control, storage tank and chemical production. Those plants have 20-40 machines to be controlled by the PCSs. The control software is written in high-level language such as C or C++, and the Line Of Codes (LOC) is between 7000 and 20000. We conducted the clustering of these PCSs by focusing on the targets of PCS control, and we found that most of them are composed of two groups: sequence control and feedback control. Then we conducted further clustering on each of these groups to categorize their functions and found that the PCSs consist of seven major functions as shown in Table1, which are sequence control, feedback control, execution control, input/output, exterior interface, state indication and fault detection and recovery. [7]

Then we created SCRs that correspond to the functions in one-to-one or one-to-many relation. The common function parts among all PCSs are developed as source codes, and the peculiar parts for each PCS (such as signal types and operation timing) are developed as parameters. The SCRs can be customized by giving these parameters externally. Furthermore, we prepared two types of requirement specification templates: the one written in natural language (N requirement specification) and the one written in Z language (Z requirement specification). The contents of both templates are the same, except the languages used. We use the N requirement specification to investigate the contents and we use the Z requirement specification as basis for designing PCSs. The external parameters are variables so we can create PCS requirement specifications by entering actual values into these parameters. Fig.2 shows the configuration of a SCR developed with the method above. Fig.3 shows a part of the data structure of the SCRs. The parameters to customize the SCR are stored in this structure.

3.1.2 Outline of Requirement Specification Creation We create PCS requirement specifications in the following three steps (see Fig.4):

(1) Extract information that needs to realize PCS

We determine the configurations of the plant (hardware) and the PCS (software) based on the design drawings and the operation scenario of the plant. Then we determine the information required to create PCS prototype (such as plant operation sequence, output information, and PCS control cycle).

(2) Construct PCS prototype using the design information

Next, we combine the SCRs and construct the PCS prototype based on the extracted information.

(3) Confirm PCS requirement through checking behavior

Finally we operate the developed PCS prototype and check if its behavior conforms to the customer's requirements. When it does not, we go back to the step (1) and modify the prototype. This process is repeated until the behavior of the PCS and the customer requirements correspond. We collect the N requirement specification templates and the customization parameters that are used to create the prototype, create the N requirement specification for the PCS, and check its content. When the content is adequate, we collect the Z requirement specification templates and the customization parameters, and create the Z requirement specification for the PCS.

To effectively perform the procedure above, we have developed a tool to support the development of PCS prototype. Fig.5 shows a sample screen of the tool. This

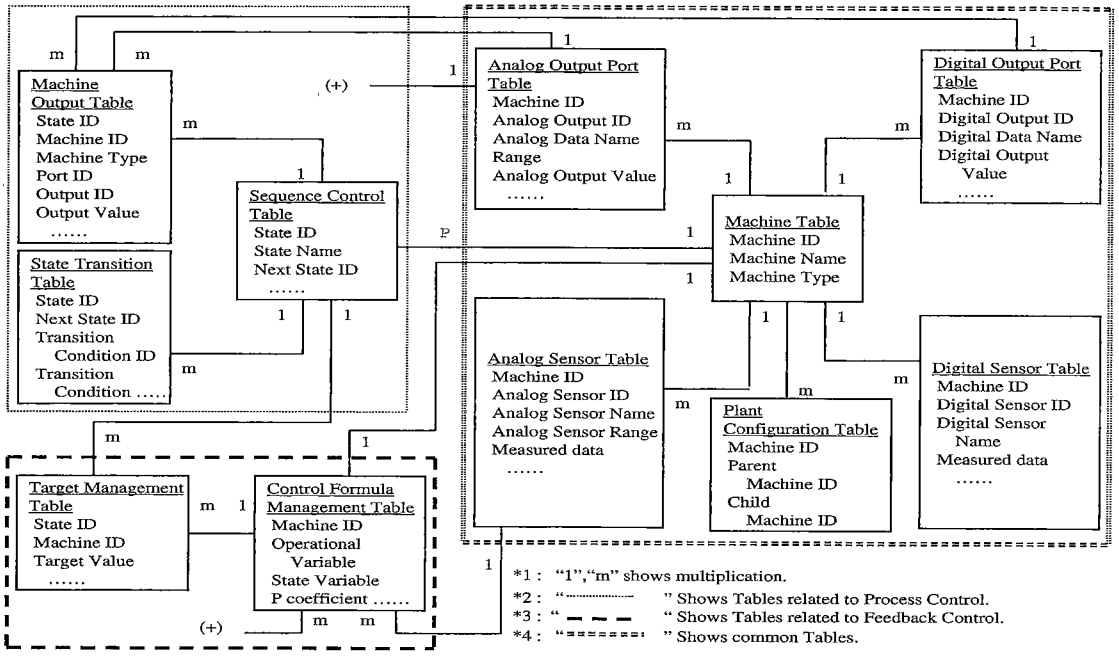


Fig. 3. Configuration of PCS data - Extracted -.

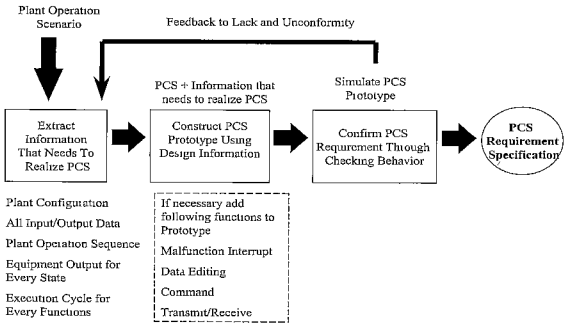


Fig. 4. Tools of creating PCS requirement specification.

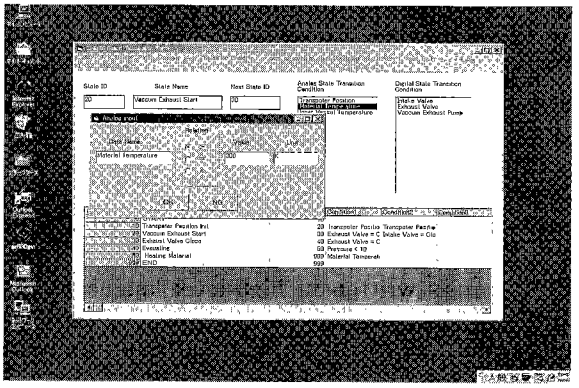


Fig. 5. Sample screen of Tools of creating PCS requirement specification.

screen is to enter the operation scenario of the sequence control group in the form of state transition table. [7]

As mentioned earlier, PCSs consist of two groups: sequence control and feedback control. In general, sequence control group of PCS is designed by defining the motion sequence of the plant and feedback control group of PCS is designed by defining the feedback control formula. So there are some differences of PCS designing method. Therefore this paper discusses how to describe requirement specification of sequence control and feedback control groups in Z language separately.

3.2 Requirement Specification of Sequence Control Group in Z Language This subsection describes the creation method for Z requirement specifications of the sequence control group. Creation of requirement specification for sequence control needs all SCRs except feedback control. Fig.6 shows the N requirement specification and Fig.7 shows the Z requirement specification of analog output management SCR.

3.2.1 Creating N requirement specification of sequence control group The N requirement specification is composed of the data definition and function definition parts.

The data definition part consist of four parts: machine data definition, measured data definition, output port definition and sequence control output definition. The machine data definition part defines the machine name to be controlled (line 4 in Fig.6). The measured data definition part consists four definitions: the machine name that the measured data belongs to, the measured data name, data type and data unit (line 8 in Fig.6). The output port data definition part defines the machine name that the output port belongs to, the output port name, the data type and the data unit (line 13 in Fig.6). The sequence control output data definition part defines the state ID (unique ID to identify a particular step in the plant operation), the machine name, the output port name and the output value (line 18 in Fig.6).

The function definition part defines the functions of analog output management SCR. The example on lines 24-25 in Fig.6, firstly obtains output port name and out-

```
//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                                  "//" line means notes
//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. 6. Requirement specification of sequence control in the natural language.

```
//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

                                                         *1: "?" means input data.
//Function Definition Part                               *2: "!" means output data.
-- Analog Output                                         *3: "//" means notes
//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. 7. Example of requirement specification of sequence control module in Z language.

```
//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                                 "//" line means notes
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. 8. Requirement specification of feedback control in natural language.

```
//Variable Definition Part
State ID = < 0, 5, 10.... >                              //Definition of State ID
Machine Name = { Heater, .... }                          //Definition of Analog Machine Name
Machine ID = {2001, ....}                                //Definition Analog Machine ID
State Value = {Heater Temperature, ....}                 //Definition of State Value
State Value Sensor ID = {201,....}                       //Definition of State Value Sensor ID
Operational Value =  Heater Current, Heater Voltage, ....} //Definition of Operational Value Name
Operational Value output  Port ID = {301, 302, ....}     //Definition of Operational output Port ID
Target Value= (State ID x Machine ID x State Value Sensor ID) |-> State Value
Target Value= {(0,2001,201) |-> 30, (5,2001,201) |-> 40, ....}
//Process Description Part
-- Feedback Control                                      //Schema of Feedback Control
//Input/Output Data Definition Part
State ID?,  Machine ID?, Target Value?, Heater Temperature?, Heater Current!, Heater Voltage!
---------------------------------                        *1: "?" means input data.
                                                         *2: "!" means output data.
//Process Definition Part                                *3: "//" means notes
//Specification of feedback control
State ID?=0 and Machine ID?=2001 and State Value sensor ID=201 => State Value = Target Value
Calculate Heater Current and Heater Voltage with PID Control in order to make Difference between State Value
and Target to Zero
---------------------------------
```

Fig. 9. Example of requirement specification of feedback control module in Z language.

put value that are shown at machine ID and state ID. Next, the output values is output through the output port that is shown at machine ID, state ID and the output port name.

3.2.2 Creating Z requirement specification of sequence control group The Z requirement specification consists of the data definition and function definition parts.

The data definition part of Z requirement specification is created by translating the N requirement specification from natural language to Z language. However, we need to break down the data table into individual data items because Z language cannot handle the data in table format (lines 7-8 in Fig.7).

The function definition part is a Z language version of the N requirement specification. The example on 16-17 in Fig.7 describes the function that derives the analog output value from the input variables. On lines 16-17 in Fig.7, "State ID? = ## ,Machine ID = ## and output port! = ## => output value ! = Analog output" is the template of the Z requirement specification for sequence control SCR, and ## indicate the parameters that are obtained from the design information of PCS prototype.

3.3 Requirement Specification of Feedback Control Group in Z Language Requirement specification of feedback control group is created in the similar way of sequence control group. Creation of requirement specification for feedback control needs all SCRs except sequence control SCR. At the phase of creating requirement specification for feedback control group, it is important to efficiently define state variables, operational variables and outline of control method.

3.3.1 Creating N Requirement Specification of Feedback Control Group Fig.8 shows the N requirement specification of feedback control group, which is created by using software components. This specification consists of two sections: definitions of state variables and operational variables of each machine for feedback control, and description of control method.

The state variable and the operational variables required for feedback control are specified in this section. The description of control method describes the functional outline of the feedback control by setting PID coefficients in standard PID control formula (lines 19-25 in Fig.8). This is because the detailed feedback control logic is not determined yet when creating the require-

ment specification. This simplification is possible because it does not need to contain the detailed feedback control logic in this phase.

3.3.2 Creating Z Requirement Specification of Feedback Control Group 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.9). The requirement specification consists of the variable definition and process description parts.

The variable definition part is described on lines 1-10 in Fig.9. The variable definition part is Z language version of "definitions of state variables and operational variables of each machine for feedback control" of N requirement specification. Some design information is appended on it.

The process description part is described on lines 11-21 in Fig.9. The process description part consists of the input/output data definition part and the process definition part. The input/output data definition part is described on lines 13-14 in Fig.9. This part indicates whether the defined data is used as input or output in the process definition part. The process definition part is described on lines 16-20 in Fig.9. This describes the outline of feedback control just like the one in N requirement specification. For example, "Get Target Value that is shown at Machine ID, State ID and Output Port Name" on line 22 in Fig.8 is developed to "State ID? = 0 and Machine ID? = 2001 and Control Value Sensor ID? = 201 => control Value! = Target Value" on line 18 in Fig.9.

4. Basic and Detailed Design Methods

4.1 Outline of Basic and Detailed Design This subsection describes a method to develop the implementation design (it consists of both basic and detailed design) of PCS based on the Z requirement specification. In our proposed implementation design method, we create design specifications of PCS by hierarchically developing the Z requirement specification. It is developed until the granularity equals to the software components used for implementation (see section 5 for the details). The proposed method uses the following characteristics of Z language:

(1) Specification can be described hierarchically.
(2) Variables can be defined accurately.

In the proposed method, a boundary between basic and detailed design phases is not clarified, and it is difficult to determine what tasks should be carried out in each phase. Therefore, we determine the tasks by comparing our method with NASDA Software Development Standard (see Table2).[5] The determined tasks are as follows.

In the basic design phase, the following tasks are required:

[**Basic 1**] Clarify the functions described in Z requirement specification, and
[**Basic 2**] Clarify the input/output data of each function.

In the detailed design phase, the following tasks are required:

[**Detail 1**] Divide the functions described in basic design, and
[**Detail 2**] Design the actual input/output data.

According to NASDA Standard, we need to develop both interface and database specifications. In the proposed implementation design method, the interface specification is described in the input/output part of implementation design specification, and the database specification is described in the data structure shown in Fig.3. Thus, our method includes every information required for designing PCS and is applicable for PCS implementation design.

Note that the detailed level of Z requirement specification is different between the sequence control group and the feedback control group, which means the different design information must be defined at the implementation design phase. Therefore, we develop the implementation design methods separately for sequence control group and for feedback control group.

4.2 Basic and Detailed Design Method for Sequence Control Group This subsection discusses a method to create basic and detailed designs of sequence control group. The sequence control group performs motion control output during the sequence predefined for a plant.

Firstly, we describe the method to create the basic design. In the basic design phase, we need to clarify the functions and the input/output data that are described in the Z requirement specification. We already have defined the outline of the sequence such as states, transition criteria, and motion control output in the Z requirement specification when we created the PCS prototype. Therefore, there is no task to be carried out in the basic design phase of sequence control group.

Secondly, we describe the method to create the detailed design of the sequence control group. In the 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 sequence (states, transition criteria, and motion control output) of entire plant and assign it to each sub-plant accordingly. The dependent relations must be assigned between the sub-plants and the entire plant.
[**Detail 2**] Describe the valid digits and the hardware interface format of output values. Those should be clarified through the preceding design work.

We have developed Sequence Control Design Tool (SCDT) to execute the above tasks efficiently. Fig.10 shows the outline of SCDT and Fig.11 shows sample of SCDT screen. SCDT is composed of three subsystems: sub-plant division tool, sub-sequence division tool, and Z language transformation tool. The sub-plant division tool is used to divide a plant into sub-plants in [Detail 1]. Dividing control target into multiple sets of actual functions make us easy to design and understand the entire control target. The sub-sequence division tool is used to divide the sequence of entire plant into sub-sequences of

Table 2. NASDA's software development standard.

Requirement Definition	Basic Design		Detail Design		Implementation	Verification
Requirement Specification	Basic Design Specification	Software IF Specification	Detail Design	Database File Specification		Test specification
User Scenario (Operational Sequence) Concreate Sequence	Function Configuration Performance Error Detection and Recovery	Not Applicable	Module Configuration Detail Control Flow Detail Data Flow Global Data	Not Applicable	Not Applicable	Combined Module Test and Hardware-Software Combined Test
Function Contents Performance Output Sequence Error Detection and Recovery Execution Cycle Algorism	Function Configuration Performance Error Detection and Recovery Concrete Sequence	Not Applicable	Module Design Input/Output Data Local Data Interrupt Signal Response Time Sequence(Control, Algorism, Error Handling)	Not Applicable	Sorce Code	Single Module Test and Combined Module Test
Resource Requirement Memory Size Execution Time	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Hardware-Software Combined Test
IF Requirement Input data Output data	Input/Output Data Explanation Unit and Precision	IF Specification IF Name, IF Protocol IF Data, Hardware IF	Not Applicable	Not Applicable	Not Applicable	Hardware-Software Combined Test
Databse Requirement Record Attribute	Upper/Lower Limit Input/Output Cycle Input/Output Data Global/Local Data	Not Applicable	Not Applicable	File/Access Type Record/File Name Data Type, Digit Initial Value	Database	Combined Module Test

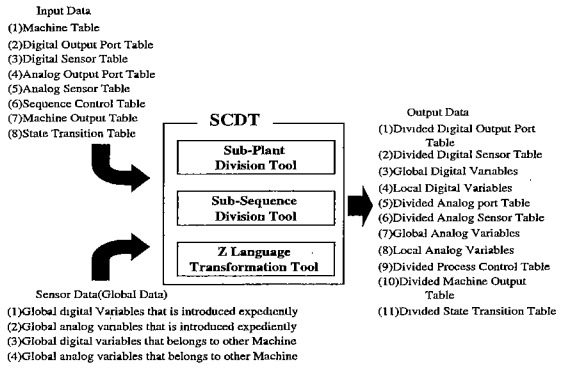


Fig. 10. Outline of SCDT.

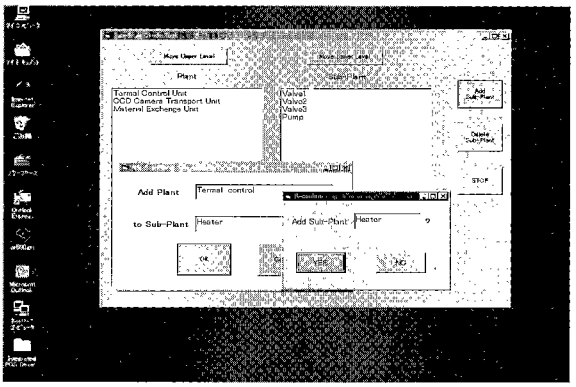


Fig. 11. Sample screen of SCDT.

each subsystem. The sub-sequences are created by dividing the state transition table for the entire plant into the ones for the sub-plants. Then we state the collaborative action between the sub-sequences. In this way, we are able to limit the range handled by each control system and simplify its logic. This should raise the efficiency of PCS development. The Z language transformation tool translates the design information (sub-sequences, or state transition tables, for sub-plants) described in natural language and the collaborative actions between the sub-sequences into Z requirement specifications. Fig.11 shows a sample screen where a "Heater" is added to the temperature control system.

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.

4.3 Basic and Detailed Design Method for Feedback Control Group This subsection discusses a method to create implementation designs of feedback control group. Feedback control group calculates motion control values (operational volume) and output them to machines. The operational volume is calculated based on the sensor values (state volume) of the machines and its target value [11]. Following tasks should be carried out according to the above stated definitions.

[**Basic 1**] Define the functional outline of feedback control formula by describing calculation method with control block diagrams such as proportional, differential and integral.

[**Basic 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

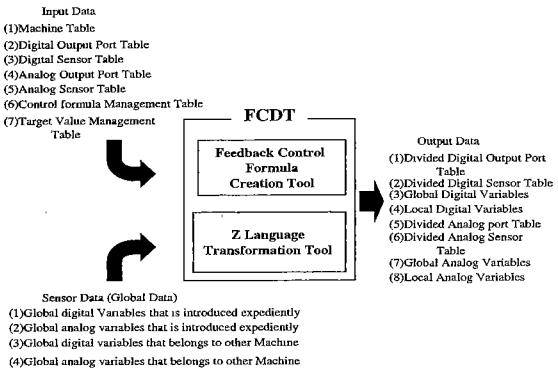


Fig. 12. Outline of FCDT.

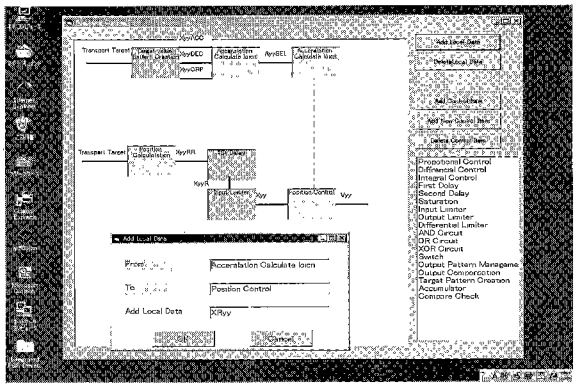


Fig. 13. Sample screen of FCDT.

PCS development, such as delays, filters and saturation, and add them to the control block diagrams created in [Basic 1]. Then define data transmission between block diagrams.

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

We have developed Feedback Control Design Tool (FCDT) to execute the above tasks efficiently. FCDT is composed of two subsystems: feedback control formula creation tool and Z language transformation tool (Fig.12). The feedback control formula creation tool is used to create feedback control formulas. This tool sorts the symbols that correspond to proportional , differential and integral control, and defines the input/output data that is sent and received between the symbols. These symbols can be describe hierarchically and then the data consistency is checked between upper and lower layers. The Z language transformation tool is used to translate symbols to a feedback control formula in Z language. As for the symbols used to create feedback control formulas, we prepare the detailed design specifications for them in Z language. We integrate these detailed design specifications for symbols and append the data to be sent and received between the symbols. This procedure is how we create the detailed design specification for the feedback control group in Z language. Fig.13 shows a sample screen to define the feedback control formula by defining XRyy data that is passed from the symbol "Acceleration Calculation" to the symbol "Position Control".

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.

5. Software Components Selection Method

We have developed software components for implementation (SCI) based on the results of domain analyses. As described in chapter 3 and 4, each function corresponds to SCR in one-to-one relation, and each SCR corresponds to basic and detailed design specifications

Table 3. Parameter of SCI - Extracted -.

SCI Name	SCI Type	Parameter	Note
Propotional control	Parameter	Propotional coefficient	
Differential control	Parameter	Differential coefficient	
Integral control	Parameter	Integral coefficient	
Saturation	Parameter	Saturation value	Cut down integral value by the saturation value
Target value pattern creation	Parameter	target value	Manage target value in order to fix process
.....	...	.....	.....

in one-to-one or one-to-many relation. We have developed SCIs to correspond to detailed design specifications in one-to-one or one-to-many relation. Therefore, each function corresponds to SCI in one-to-one or one-to-many relation, which enables us to implement PCS using SCI in each functional unit. In our proposed method, select SCI applying to detailed design specification in Z language. And PCS are developed by combining SCIs.

5.1 Components for PCS Implementation

At first, investigate software components for implementation (hereafter, SCI) of actual PCS. Fig.2 shows full functions that are needed for PCS. Furthermore clarify the scope of customization for divided functions and decide the parameters to customize the functions. Based on the result, SCI is developed. Fig.14 shows sample code of SCI, Fig.15 shows the configuration of SCI, and Table3 shows customizing parameter that is used for SCI. We have developed the SCI in C language. The reason we adopt C language is to describe the bit handling of input/output interfaces and its timing.

Secondly, explain about component list (shown in Fig.16) that manages SCI. Component list contains input/output interface information and functional classification information. The input/output interface information is composed using argument information of SCI provided by the authors. The input/output data and their data types are listed in the interface information column. The functional classification column contains the functional outlines of SCI. Users can select this information from classification candidates and add it to

```
{int i;  FILE *fp;  transitable buff;  char buff1[100]; stateNum = 0;
 fp=fopen(inputTransCondPath,"r");
 fseek( fp, 0L, SEEK_SET );
 for(i=0;i>=0;i++)
  {
   if(EOF==fscanf( fp, "%d %s %d %s %s %s %s %s %s %s %s %s %d",
   &buff.curCondNum,&buff.curCondNam,&buff.nxtCondNum,
   &buff.condition[0],&buff.condition[1], ..... , &buff.condition[9],
   &buff.condNum)) break;
   fgets(buff1,100,fp);
   stateNum = stateNum + 1;
   curtranstable[i].curCondNum =buff.curCondNum;
   strcpy(curtranstable[i].curCondNam,buff.curCondNam);
   curtranstable[i].nxtCondNum = buff.nxtCondNum;
   strcpy(curtranstable[i].condition[0],buff.condition[0]);
   strcpy(curtranstable[i].condition[1],buff.condition[1]);
      .....
   strcpy(curtranstable[i].condition[9],buff.condition[9]);
   curtranstable[i].condNum = buff.condNum;
  }
 fclose(fp);
 return 0;
}
```

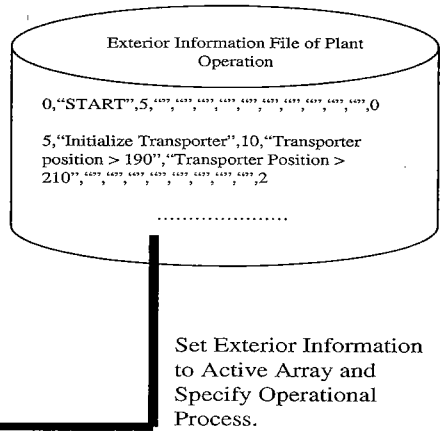


Fig. 14. Sample code of SCI.

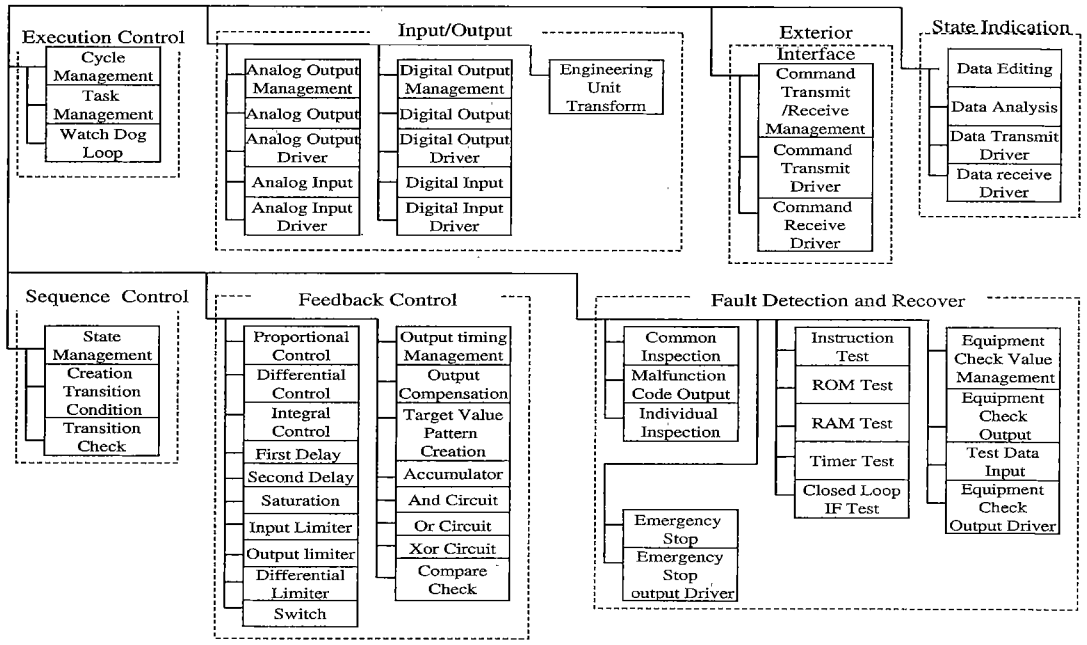


Fig. 15. Configuration of SCI.

the component list when registering SCI to the list.

5.2 SCI Selection and Combination Method

The well-known software components selection methods are Faceted Classification [12] and Specification Matching [13]. These methods can select software components that belongs to various domain. But they need a lot of time to prepare search data. In our proposed method, we adopt simplified search method restricting to PCS domain. SCI are selected in the following steps (see Fig.16):

[**Step 1**] Decide best matching functional classification for each schema in PCS detailed design specification, and match them with functional classification on the component list. Then every SCI, which belongs to the matched functional classification, are listed.

[**Step 2**] Narrow down the candidates of SCI by matching input/output interface on the component list and on the detailed design specification . This matching uses information such as number of variables or data types.

[**Step 3**] When more than one SCI is found, all of them are listed, and user selects an appropriate one considering performance and data area size. When no SCI is found, the schema part of detailed design

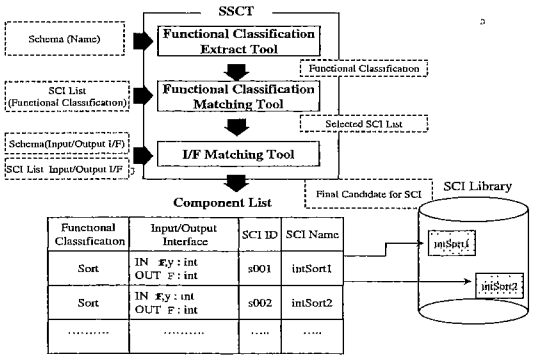


Fig. 16. Outline of SSCT.

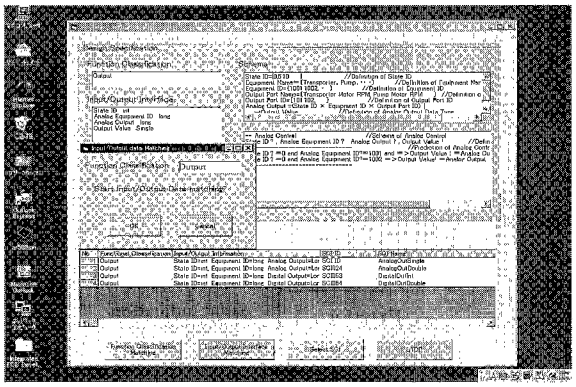


Fig. 17. Sample screen of SSCT.

specification must be reviewed.

We have developed SCI Selection and Combination Tool (SSCT) to execute the above tasks efficiently. Fig.16 shows the outline of SSCT. Fig.17 shows the sample screen of SSCT. Fig.17 shows selecting SCI corresponding to functional classification "output".

6. Application and Evaluation of Integrated PCS Development Environment

6.1 Outline of Experimental Systems We evaluate effectiveness of SCR, SCI, SCDT, FCDT and SSCT (hereafter, IPDE: Integrated PCS Development Environment). The following sections describe the evaluation items for IPDE and the case examples used for the evaluation.

6.1.1 Evaluation Items for IPDE There are two evaluation items for IPDE: (a) the conformity to the requirements for the PCS and (b) the efficiency of the PCS development. We evaluate the item (a) in both software and hardware. The conformity of software is evaluated based on (a-1) whether every function described in the requirement specification is implemented in the PCS. The conformity of hardware is evaluated based on (a-2) whether the program size of the developed PCS is smaller than the memory capacity and (a-3) whether the processing time of the developed PCS is shorter than the required control cycle. This is because we need to develop PCSs that conform the requirements in the restricted environment (usable CPU is slow and allowed memory size is small because of the license that hardware can use in the space) [4]. We evaluate the item (b) based on (b-1) Code Reuse Rate (CRR) and (b-2) Time Reduce Rate (TRR) of PCS development. The CRR is calculated in Formula (1). LOC in the formula stands for Lines Of Codes. Higher CRR means greater number of SCI is applied to the PCS development and is evaluated as improvement of development efficiency.

$$\text{CRR} = \text{LOC of SCI} \;/\; \text{Total LOC of PCS} \cdots\cdots (1)$$

The TRR is calculated in Formula (2). DT in the formula stands for Development Time. Lower TRR means shorter development time and is evaluated as improvement of efficiency. DT is determined as the time spent between the creation of requirement specification and the completion of the unit test. The tests after the unit test are not included because these tests are conducted by implementing the PCS to the actual plant and by operating it, which means that the difference of development methods should not affect the development time.

$$\text{TRR} = \text{DT by Proposal Method} \;/\; \text{DT by Newly Development} \cdots\cdots\cdots (2)$$

6.1.2 Case Examples Used for Evaluation of IPDE The outline of the plants A-E as follows: The plants A-E contain the sequence control and the feedback control. All of them are rather small-scale space plants and are controlled with one 16-bits CPU.

The plant A is the thermal control plant in the space station laboratory. This plant controls the temperature by circulating heating fluid in the pipes set around in the laboratory. This plant operates in the control cycle of 125 [milli-second (msec)]. The plants B and C are the experimental plants for semiconductor manufacturing in an earth satellite. Each plant has a chamber for experiments and manufactures semiconductors by inserting experimental materials into the chamber and heating them. The both plants operate in the control cycle of 100 [msec]. The plant D is the connection device with motors to connect and release two machines. This plant operates in the control cycle of 250 [msec]. The plant E is the plant to generate homogeneous chemical products under weightless conditions and has one chamber. It manufactures chemical products by heating, cooling and drying the materials in the chamber. This plant operates in the control cycle of 1 [second]. For example, Fig.18 shows the PCS configuration of the Plant A.

6.2 Evaluation of IPDE

6.2.1 (a-1) Evaluation of the Conformity to PCS Requirements This item evaluates whether every function is implemented in the PCS. As described in chapter 3, 4 and 5, there are one-to-one or one-to-many correspondences between each function and SCR, SCR and requirement specification, requirement specification and basic/detailed design, and detailed design and SCI. Moreover, Z language ensures that the design information is conveyed between them without omission. This allows us to trace design information from each function to SCIs, which enables us to implement every function in the PCS. As every function is extracted

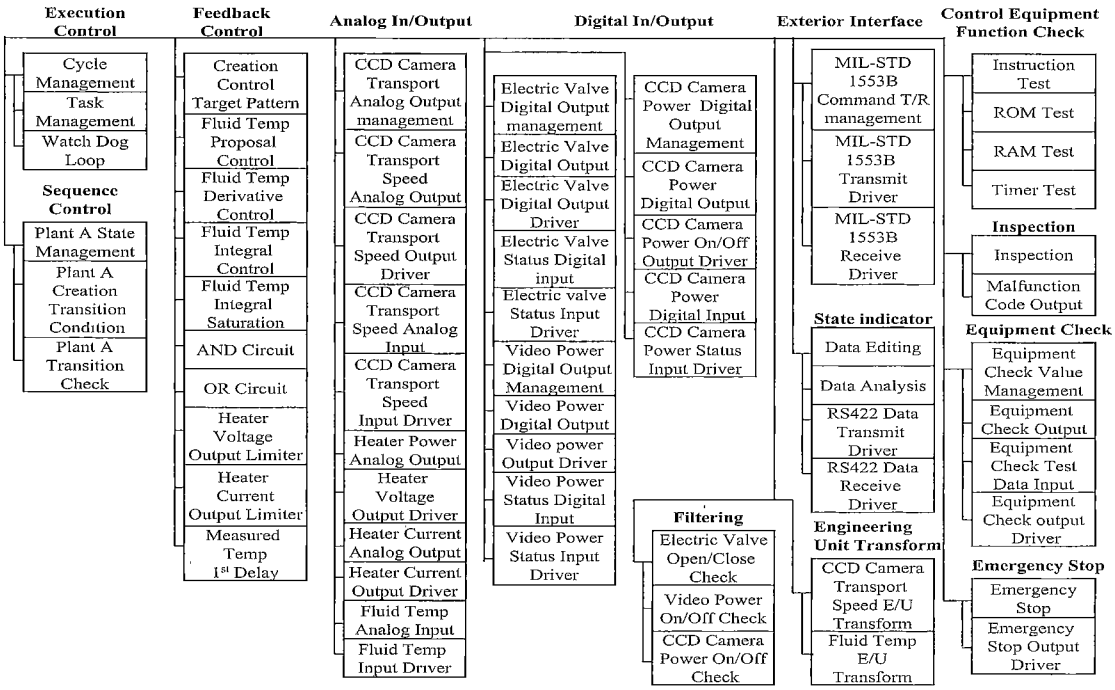


Fig. 18. PCS Configuration.

Table 4. List of SCI program size.

SCI Group	SCI Name	Size[byte]	SCI Group	SCI Name	Size[byte]
Execution Control	Cycle Management	207	input/output	Analog Output Management	259
	Task Management	172		Analog Output	151
	Watch Dog Loop	48		Analog Output Driver	87(Average)
Sequence Control	State Management	774		Analog Input	148
	Creation Transition creation	520		Analog Input Driver	102(Average)
	Transition Check	234		Digital Output Management	227
Feedback Control	Proportional Control	121		Digital Output	131
	Differential Control	515		Digital Output Driver	95(Average)
	Integral Control	859		Digital Input	109
	First Delay	51		Digital Input Driver	92(Average)
	Second Delay	76	Exterior Interface	Engineering Unit Transform	530
	Saturation	261		Command Transmit/Receive Management	692
	Input Limiter	177		Command Transmit Driver	152(Average)
	Output Limiter	152	Fault Detection and Recover	Command Receive Driver	181(Average)
	Differential Limiter	220		Common Inspection	654
	Switch	103		Malfunction Code Output	194
	Output timing Management	1045		Individual Inspection	318
	Output Comparison	135		Emergency Stop	336
	Target Value Pattern Creation	891		Emergency Stop Driver	120(Average)
	Accumulator	157		Instruction Test	145
	And Circuit	32		ROM Test	90
	Or Circuit	37		RAM Test	128
	Xor Circuit	50		Timer Test	60
	Compare Check	122		Close Loop Test	80
State indication	Data Editing	651		Machine Check Value Management	127
	Data Analysis	447		Machine Check Output	218
	Data Transmit Driver	232(Average)		Machine Check Test Data Input	192
	Data Receive Driver	313(average)		Machine Check Out Driver	97(Average)

through the creation of the prototype using SCR, we are able to implement the PCS that conforms the requirement specification by combining SCIs with functional units.

6.2.2 (a-2) Evaluation of PCS Program Size Prediction The evaluation item (a-1) determines the unique SCI to be used for each function of the PCS. The content and program size of SCI is determinate as shown in Table4 (as the program sizes of drivers is inherent in each plant, we use the average program sizes calculated from existing plants. This is because the detailed prediction is not required at the creation of requirement specifications). If the compiler does not compress the memory, it is possible to predict the program size of the PCS by summing up the program size of every SCIs used. The each group of PCS, sequence control group (S-Gr), feedback control group (F-Gr) and other parts (fault detection and recovery), is developed inde-

Table 5. Comparison between estimated and actual program size.

Plant Name	S-gr Number of						F-Gr Number of						Other Part [byte]	Estimated PGM Size [byte]	Actual PGM Size [byte]	Estimated /Actual [%]
	Sub-Plants	Measurements		Outputs		PGM Size	Sub-plants	Measurements		Outputs		PGM Size				
		analog	digital	analog	digital	[byte]		analog	digital	analog	digital	[byte]				
A	1	8	8	4	6	8632	1	8	8	6	4	11867	2567	23066	21156	109
B	2	20	12	8	8	16876	1	8	4	8	6	11651	2567	31094	28506	109
C	1	12	8	8	4	9206	4	20	24	16	18	44149	2567	55922	52512	106
D	1	8	8	6	6	8822	1	12	12	8	6	12835	2567	24224	22860	105
E	3	24	20	28	16	26834	2	16	16	8	10	23269	2567	52670	47808	110

Table 6. Execution time - instruction type -.

	Operation Gr.	Movement Gr.	Comparison Gr.	Divergence Gr.	Stack Gr.	IO Gr.	Interruption Gr.
Needed Clocks	5	5	6	22	20	20	80
Execution Time [micro-sec]	1.25	1.25	1.5	5.5	5.0	5.0	20.0

Table 7. Execution time - instruction type -.

Plant Name	Operation Gr.	Movement Gr.	Comparison Gr.	Divergence Gr.	Stack Gr.	IO Gr.	Interruption Gr.	executed instructions per control cycle	Estimated execution time[mili-sec]	Actual execution time[mili-sec]	Control Cycle [mili-sec]
A	7535	8733	4198	6444	6554	1453	651	35648	0.12	0.13	0.25
B	5931	6822	4699	3332	3688	983	258	25714	0.07	0.08	0.125
C	8725	8481	8518	8311	5481	1008	594	41118	0.12	0.11	0.5
D	7537	7783	2678	2678	6584	2019	872	31833	0.10	0.10	0.5
E	8092	7165	8766	8766	8361	2366	866	42133	0.15	0.16	1.0

Table 8. Result of SCI reuse rate.

Plant Name	Sequence Control Gr.		Feedback Control Gr.		Other Part		SCI Total [LOC]	Total LOC [LOC]	CRR [%]
	SCI	Coding	SCI	Coding	SCI	Coding			
A	4598	2611	2388	897	449	105	7435	11048	66.5
B	3204	2027	1454	731	449	89	4927	7774	63.5
C	2122	1398	6126	3201	449	465	8697	13761	67.1
D	2476	1225	2596	1437	449	282	5521	8465	65.2
E	5992	3491	3634	1558	449	303	10075	15427	65.3

Table 9. Result of Development Time.

Plant Name		Requirement Specification Creating time [Hours]	Basic and Detaied Design Time [Hours]	Programming Time [Hours]	Verification Time [Hours]	Total Time [Hours]	TRR [%]
A	IDPE	28	62	67	65	222	40.0
	newly development	75	136	176	161	548	
B	IDPE	22	40	58	43	163	40.5
	newly development	61	92	141	102	396	
C	IDPE	35	81	71	80	323	44.9
	newly development	118	172	252	178	720	
D	IDPE	19	41	51	52	163	42.1
	newly development	55	80	135	117	387	
E	IDPE	42	92	88	93	315	41.9
	newly development	134	180	233	204	751	

pendently. Therefore, the program size of the PCS is sum total of the ones of the all groups. Table5 shows the result of program size prediction for the plant A-E.

S-Gr consists of five SCI groups: sequence control, execution control, input/output, exterior interface and state indication. We newly develop analog and digital input/output drivers in the input/output SCI group because they depend on the hardware interface of the control device. These drivers should be developed for each measurement since the input/output ports and the interface operation timing defined in the drivers depend on the hardware. S-Gr includes the customizing parameters such as the state management data of plant motion and the analog/digital output value, however we ignore them because their sizes are rather small in comparison with the ones of SCIs. Consequently, the types and the number of SCIs used in S-Gr are: one each for execution control, exterior interface and state indication; the number of sub-plants of S-Gr for sequence control group; the number of output points for analog/digital output managements, outputs and output drivers in input/output group; and the number of measurements for analog/digital inputs and input drivers in input/output group. At the implementation of PCS, we need to add and modify the codes for the functions that are not covered with SCI. However, we do not consider the change

in program size through the addition and the modification because we have found from the development experiences that the size of added and modified codes is nearly equal to the size of deleted codes. Table5 shows the predicted program sizes of S-Gr.

F-Gr consists of five SCI groups: feedback control, execution control, input/output, exterior interface, and state indication. We deal with the analog and digital input/output drivers in the input/output SCI group in the same manner as S-Gr. F-Gr includes the parameters to customize feedback control, however we ignore them because their sizes are rather small in comparison with the ones of SCIs. Consequently, the types and the number of SCIs used in F-Gr are: one each for execution control, exterior interface and state indication; the number of sub-plants of F-Gr for feedback control group; the number of output points for analog/digital output managements, outputs and output drivers in input/output group; and the number of measurements for analog/digital inputs and input drivers in input/output group. We do not consider the change in program size through the addition and the modification for the same reason as S-Gr. Table5 shows the predicted program sizes of F-Gr.

The other part consists of the SCI groups for fault detection and recovery. When a fault arises, the entire plant must be secured. Therefore, the fault detection and recovery processes must be considered for the entire plant. We newly develop the device check output drivers and the emergency stop output drivers since their functions depend on the hardware interface of the control device. And the individual check functions should be developed for each plant. The other functions are implemented with SCI, and therefore they are included in the fault detection and recovery group used in the other parts. As the program sizes of the device check output drivers, the emergency stop output drivers and the individual checks are unknown at the creation of requirement specifications, we again use the average program sizes calculated from existing plants. This is because the detailed prediction is not required at this stage. We use the program sizes of SCIs for the other parts. Consequently, the program size of the fault detection and recovery is the steady value of 2567[byte].

As shown in Table5, the predicted value for the program size of PCS using IPDE is approximately 10[%] larger that the actual size. This is because some SCIs can be commonly used in multiple sub-plants and the actual number of SCIs is smaller than the prediction. This small error should be accepted as for the prediction at the initial stage of development. As a result of applying IPDE, we are now able to predict the program size of PCS at the initial stage of development. This should reduce the cases that PCS cannot be installed in the memory of control device and the modification is required in large scale after the completion of development, which severely affect the cost and the period of PCS developments.

6.2.3 (a-3) Evaluation of PCS Processing Time As described in (a-2), IPDE allows us to specify the SCIs to be used. As shown in 19, expanding the repeated processes in the SCIs enables us to grasp the entire codes executed within the control cycle of PCS. These codes can be classified into instruction types. Then the approximate processing time can be predicted by counting the number of each instruction type and multiplying it with its processing time. In the case of high-level languages, the processing time of each instruction type highly depend on the performance of compliers. We therefore compile the SCIs into assembler and classify the codes into instruction types such as Operation (SUB, ADD etc.), Movement (MOV etc.), Comparison (CMP etc.), Divergence (JNP etc.), Stack (PUSH, POP etc.), IO (IN, OUT etc.), and Interruption (INT, IRET etc.). We calculate the processing time for each instruction type of assembler level based on the operating clock of the control device and the number of clocks required for each instruction type. We assume the operating clock of the CPU as 4[MHz]. Table6 shows the processing time of each instruction type. Table7 shows the predicted and actual processing time of the PCSs. As shown in Table6, the error between the predicted and actual processing time is approximately 10[%]. We consider that this error mainly caused by the grouping of instructions. However, this small error should be accepted as for the prediction at the creation stage of requirement specifications. As a result of applying IPDE, we are now able to predict the processing time of PCS at the initial stage of development. This should reduce the cases that the processes do not completed within the control cycle ,and the modification is required in large scale after the completion of development, which severely affect the cost and the period of PCS developments.

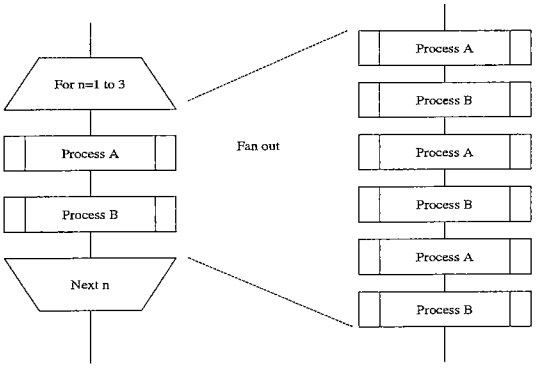


Fig. 19. Actual executed instructions.

6.2.4 (b-1) Evaluation of SCI Reuse Rate In the conventional methods to newly develop PCSs, the source codes are reused mainly in calculating modules, and their reuse rates are only around 10[%]. Table8 shows the CRRs in the developments of the plants A-E using IPDE. The average CRR goes up to 65.5[%], which is a significant improvement from the conventional methods. Applying IPDE enables us to implement most of the parts of PCSs using SCIs (the exceptions are input/output drivers, and fault detection and recovery functions that are inherent in each plant). As a result of applying IPDE, we have succeeded to reduce newly

written codes on PCS developments, and consequently we have reduced the development costs and shortened the development periods. If we develop SCIs for frequently used input/output drivers such as RS422, MIL-STD1553B and ARINC429, we will be able to improve the CRR and it should results in smaller development costs and shorter development periods.

6.2.5 (b-2) Evaluation of Development Time Reduce Rate (TRR) Table9 shows the comparison of development time between proposed method using IPDE and the conventional methods. As described earlier, we define the development time as the time spent between the creation of requirement specification and the completion of the unit test for PCS. The experiment subjects are the novice developers with two to three year experiences. As shown in Table9, the average TRR is 42.1[%]. We analyze this figure resulted from the facts that: (1)requirement specifications can be created efficiently by showing the operation of the prototype to the users, (2)standardization of design methods reduces trial and error at the basic and detailed design phases, and (3)using SCIs reduces the newly developed codes at the programming phase. As the results of above-mentioned evaluations, we confirm that applying IPDE to PCS developments reduces costs and time for developing PCSs.

7. Conclusion and Future Work

This paper proposed IPDE. As the results of applying IPDE to actual PCS developments, it become possible to: (1)develop PCSs that fully satisfy the requirements, (2)predict the program size and processing time within 10[%] error, (3)improve the software reuse rate of predefined SCI up to 66.5[%] and (4)cut down the development time to 42.1[%] in comparison with conventional methods. We expect that the efficiency of PCS developments can be further improved by reinforcing SCI libraries.

Acknowledgement

A part of this research was supported by New Energy and Industrial Technology Development Organization (NEDO) under contract "Basic Technology for Next Generation Transportation System Design". The author wishes to acknowledge all those involed.

(Manuscript received Nov. 29, 2001,
revised April 1, 2003)

References

(1) D'Souza D. F. and Willis A. C.: "Objects, Components and Framework with UML: The CATALYSIS Approach", Addison Wesley (1999)

(2) J. Karl-Heinz and M. Tiegelkamp: "IEC61131-3 Programming Industrial Automation Systems: Cncepys and Programming Languages, Requirement for Programming Systems, AIDS to Decision-Mark", Springer Verlag (2001)

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

(4) National Space Development Agency of Japan: "NASDA Parts Application Standard", NASDA-HDBK-4 (NASDA-CR-78602) (1992) (In Japanese)

(5) National Space Development Agency of Japan: "NASDA Design Standard Software Design Standard", NDC-1-9-1 (1996) (In Japanese)

(6) M. Natori, A. Kagaya and S. Honiden: "Reuse of Requirements Specification Based on Domain Analysis", *Inforamation Processing Society of Japan*, Vol.37(3) (1996) (In Japanese)

(7) M. Takahashi and K. Tsuda: "The Efficient Method of Plant Software Requirements Definiton", *Inforamation Processing Society of Japan*, Vol.42(2) (2001) (In Japanese)

(8) A. Norcliffe and G. Slater: "Mathematics of Software Construction", Ellis Horwood (1991)

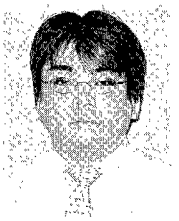
(9) M. Kawakita, M. Sakai, S. Yamamoto and K. Agusa: "A Model for Reuse Based on Formal Specifications", *Inforamation Processing Society of Japan*, Vol.36(5) (1995) (In Japanese)

(10) H. Kobayasi, Y. Kawata, N. Maekawa, A. Kawasaki, A. Yabu and K. Onogawa: "Modeling External Objects of Process Control System in Executable Specifications", *Inforamation Processing Society of Japan*, Vol.35(7) (1994) (In Japanese)

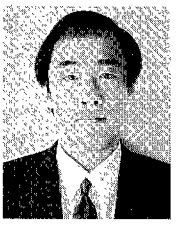
(11) S. Teshima, Y. Inamori and K. Agusa: "EFS Program Model for Embedded Real-Time System and EFS-Based CAE Tool Schetch", *Electric Inforamation Comunication Society of Japan*, Vol.37(8) (1997) (In Japanese)

(12) R. Prito-Diatz: "Implementing Faceted Classification for Software Reuse", *Communications of ACM*, Vol.34(5) (1991)

(13) J. Jeng and B. Cheng: "Specification Matching for Sotware Reuse: A Foundation", *In Proc. Software Engineering and Knowledge Engineering*, Vol.2(4), pp.523–546 (1992)



Masakazu Takahashi (Member) 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. Since 1988, he has been with Ishikawajima-Harima Heavy Industries Co., Ltd. Currently assigned to a subsidiary of IHI, Galaxy Express Corporation. He is a member of The Information Processing Society of Japan, The Society of Instrument and Control Engineers.



Kazuhiko Tsuda (Member) He 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 assistant professor in Graduate School of Systems Management, University of Tsukuba, Tokyo, Japan since 1998. 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 Electronics, Information and Communication Engineers.