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, 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 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\(^3\). 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
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 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. 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. 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.

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, --)

1. Machine output (state ID, machine, output)
   - "transporter velocity" = "V"
   - "heater power" = "P"
   - "intake valve" = "open"
   - "exhaust valve" = "open"

2. Measurement data list
   - "transporter position initial"
   - "intake valve status"
   - "transporter motor speed"

3. Equipment list
   - "transporter"
   - "heater"
   - "intake valve"
   - "exhaust valve"

4. State variable list
   - "transporter position"
   - "intake valve status"
   - "transporter motor speed"

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.
(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 resulting output using this tool.

Fig. 4 shows an example of these functions described in natural language (feedback).

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

- 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.
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. 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 and Specification Matching.

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.
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 component 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 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 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 SCFT, FCDT, and SSCT, to actual PCS developments. 

We have applied the IPDE to five plant developments (Plant 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\(^{(12), (16)}\).

\[
RR_{CM} = 100 \times \frac{Nrcm}{Nrcm + Nnncm} \\
RR_{CD} = 100 \times \frac{Nrcm + Nnncm}{Nrloc + Nnloc} \\
RR_{DT} = 100 \times \frac{1 - DTppm}{DTesm}
\]

Table 3: Number of existing and newly developed software components

<table>
<thead>
<tr>
<th>Plant Name</th>
<th>Components of sequence control type sub-plants [pieces]</th>
<th>Components of Feedback control type sub-plants [pieces]</th>
<th>Components of other parts [pieces]</th>
<th>RR_CM [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>34 31</td>
<td>11 11</td>
<td>8 5</td>
<td>66</td>
</tr>
<tr>
<td>B</td>
<td>24 22</td>
<td>8 8</td>
<td>8 3</td>
<td>64</td>
</tr>
<tr>
<td>C</td>
<td>17 16</td>
<td>29 29</td>
<td>8 3</td>
<td>76</td>
</tr>
<tr>
<td>D</td>
<td>17 15</td>
<td>13 13</td>
<td>8 5</td>
<td>69</td>
</tr>
<tr>
<td>E</td>
<td>44 40</td>
<td>15 15</td>
<td>8 7</td>
<td>58</td>
</tr>
</tbody>
</table>

Table 4: Reused source code [LOC] of each development

<table>
<thead>
<tr>
<th>Plant Name</th>
<th>LOC of sequence control type sub-plants [LOC]</th>
<th>LOC of Feedback control type sub-plants [LOC]</th>
<th>LOC of other parts [LOC]</th>
<th>RR_CD [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6907 3917</td>
<td>2418 897</td>
<td>449 105</td>
<td>67</td>
</tr>
<tr>
<td>B</td>
<td>7108 4554</td>
<td>1574 731</td>
<td>449 81</td>
<td>63</td>
</tr>
<tr>
<td>C</td>
<td>4159 2740</td>
<td>6106 3101</td>
<td>449 465</td>
<td>63</td>
</tr>
<tr>
<td>D</td>
<td>5016 2501</td>
<td>2346 1297</td>
<td>449 282</td>
<td>65</td>
</tr>
<tr>
<td>E</td>
<td>8688 5061</td>
<td>1528 449</td>
<td>449 303</td>
<td>65</td>
</tr>
</tbody>
</table>

Table 5: Development time [hours] of each development

<table>
<thead>
<tr>
<th>Plant name</th>
<th>Development time in established method</th>
<th>Development time in proposed method</th>
<th>RR_TM [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>548</td>
<td>222</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>323</td>
<td>55</td>
</tr>
<tr>
<td>C</td>
<td>396</td>
<td>162</td>
<td>59</td>
</tr>
<tr>
<td>D</td>
<td>387</td>
<td>163</td>
<td>58</td>
</tr>
<tr>
<td>E</td>
<td>751</td>
<td>315</td>
<td>58</td>
</tr>
</tbody>
</table>
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. \( \sum \) represents a sum total of all sub-plants.

\[
T_{Nscc} = \sum (N_{sc} + N_{sco} + (N_{scm} + N_{sci}) \times N_{smp})
\]
\[
= \sum (3 + 2 \times N_{smp})
\] (4)

\( T_{Nscc} \): Total number of S-type software components
\( N_{sc} \): Number of S-type software components
\( N_{sco} \): Number of S-type output driver components
\( N_{scm} \): Number of S-type software components for measurement
\( N_{sci} \): Number of S-type input driver components for measurement
\( N_{smp} \): 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.

\[
T_{Ls} = \sum ((L_{slc} + L_{slo}) + (L_{slm} + L_{sli}) \times N_{smp} + (L_{sli} + L_{sid}) \times N_{smp} + 2 \times N_{sls} + 51)
\]
\[
= \sum (699 + 174 \times N_{smp} + 2 \times N_{sls})
\] (5)

\( T_{Ls} \): Total LOC for S-type software components
\( L_{slc} \): LOC for S-type software components
\( L_{slo} \): LOC for S-type output drivers
\( L_{slm} \): LOC for S-type output software components for measurement
\( L_{sli} \): LOC for S-type input drivers
\( N_{smp} \): Number of S-type measurement points
\( L_{slp} \): LOC for sequence management data per one state
\( L_{sid} \): LOC for output measurement data per one state
\( N_{smp} \): Number of state transitions
\( N_{sls} \): Number of state transitions

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. \( \sum \) represents a sum total of all sub-plants.

\[
T_{Nfc} = \sum (N_{fc} + N_{fco} + (N_{fcm} + N_{fci}) \times N_{fmp})
\]
\[
= \sum (21 + 2 \times N_{fmp})
\] (6)

\( T_{Nfc} \): Total number of F-type software components
\( N_{fc} \): Number of F-type software components
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 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.

- 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% [1]. 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 the reduced rate of development time was 58% [1]. 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


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.