〔F18p22〕

FLEXIBILITY OF CONTROL PROCESSES BY OOP METHODOLOGY

Hiroshi YOSHIKAWA, Hironao SAKAKI, Yuichi ITOH, Yasushi TERASHIMA and Hideaki YOKOMIZO

Accelerator Division, Japan Synchrotron Radiation Research Institute.
323-3 Mihara, Mikazukimachi, Sayogun, Hyogoken, JAPAN

Abstract

The injector linac of SPring-8 meets an opportunity to be expanded two beam transport lines for the medium energy storage ring (NewSUBARU) and for fundamental investigation aiming SASE in this summer(1998). The control system has been added three VMEs for the new beam transport lines, and the modification was put to software. On this work, the advantage of object oriented programming (OOP) is demonstrated sufficiently and able to execute with very few work quantity. The details of works on the software and the merit of the modeling of processes that enable to ease this modification are shown.

オブジェクト指向による制御プロセスの再利用性

はじめに

SPring-8 の入射系線型加速器は、S-band の 100MW 級ﾊﾟﾙｽｸﾗｲｽﾄﾛﾝ 14 本を擁する 1GeV 電子線型加速器である。最大 60pps で、1ns、10ns、40ns、1μs 等のﾊﾟﾙｽ幅ﾊﾞﾘｴｰｼｮﾝ、ﾋﾟｰｸ電流数ｱﾝﾍﾟｱのﾋﾞｰﾑを出力することができる。平成 8 年 8 月から 2 年間順調に稼動してきており、この夏に、兵庫県が建設する 1.2GeV の中型蓄積ﾘﾝｸﾞ（通称 NewSUBARU）に 1ns のﾋﾞｰﾑを入射するための輸送系と、線型加速器単独の開発研究である SASE を目的とした試験用ﾋﾞｰﾑ輸送系の 2 本が増設されることになった。これに伴い、線型加速器の制御系に新たな制御対象が追加されることになり、ﾊｰﾄﾞｳｪｱｿﾌﾄｳｪｱともに追加と改造が必要となった。

改造の規模

既存部分の線型加速器は全長 140m で、14 式のﾊﾟﾙｽｸﾗｲｽﾄﾛﾝとﾓｼﾞｭﾚｰﾀ、3m のﾃﾞｨｽｸﾛｰﾄﾞ型加速管 26 本、Q-Triplet20 台、ﾋﾞｰﾑ輸送系の四極電磁石が 9 台、ｲｵﾝﾎﾟﾝﾌﾟ 73 台、ｽｸﾘｰﾝﾓﾆﾀ 27 台、ｺｱﾓﾆﾀ 5 台等で構成されている。今回追加されるのは、最後尾のﾋﾞｰﾑﾀﾞﾝﾌﾟ手前にあるﾌﾞｰｽﾀｼﾝｸﾛﾄﾛﾝへの偏向電磁石の直後に設置する新しい偏向電磁石から下流部分で、偏向電磁石 4 台、四極電磁石 25 台、ｽｸﾘｰﾝﾓﾆﾀ 14 台、ｲｵﾝﾎﾟﾝﾌﾟ 11 台である。高周波加速系がないので、制御信号の点数は 6 分の 1 程度であるが機器の設置されている部分の規模は総延長で本体に匹敵する。

本体制御系は VME ﾊﾞｽ型ｺﾝﾋﾟｭｰﾀ(21 ｽﾛｯﾄ)のｹｰｼﾞ 21 台が使用されており、それに VME のﾌｧｲﾙｻｰﾊﾞを兼ねたﾏﾝﾏｼﾝ用ﾜｰｸｽﾃｰｼｮﾝ(HP-UX)、ﾃﾞｰﾀｱｸｲｼﾞｼｮﾝ用ﾜｰｸｽﾃｰｼｮﾝ(OpenVMS)を加えたものが 1 階層のｲｰｻﾈｯﾄ上に接続されている。今回増設される機器の IO をｶﾊﾞｰするために必要な VME ﾊﾞｽ型ｺﾝﾋﾟｭｰﾀは 3 台であった。

ﾈｯﾄﾜｰｸのﾄﾎﾟﾛｼﾞｰは変更せず、既存の階層にそのまま新たな VME ﾊﾞｽ型ｺﾝﾋﾟｭｰﾀが接続されるが、制御ﾛｼﾞｯｸとしてはﾋﾞｰﾑ輸送の許可と運転ﾓｰﾄﾞの管理を受けて動作する必要があるので、制御ｼｰｹﾝｽのﾛｼﾞｯｸ上は本体とは別の加速器が存在している形とした。機器の操作画面は既存部分のﾏﾝﾏｼﾝ用ﾜｰｸｽﾃｰｼｮﾝ上に追加する形で構築した。

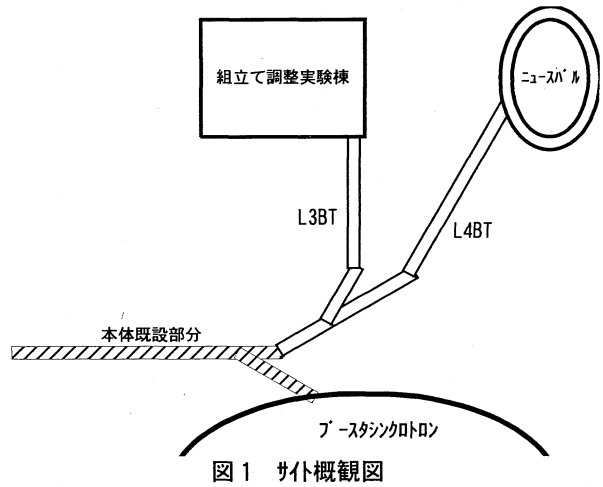

図 1　ｻｲﾄ概観図

円滑な改造のための戦略

制御システムは既存部分の資源を最大限活用して最小の作業量で速やかに対応することにした。夏の停止期間内に機器の設置から計算機制御系からの動作試験までを完了させないと、蓄積リングの営業運転に支障が出る。もっとも短時間にﾘｽｸの少ない確実な改造を行う方法を考えた。線型加速器本体の機器制御に使われている 21 台の VME ﾊﾞｽ型ｺﾝﾋﾟｭｰﾀは、起動時に必要なﾌｧｲﾙをﾜｰｸｽﾃｰｼｮﾝからﾀﾞｳﾝﾛｰﾄﾞしてﾃﾞｨｽｸﾚｽで動作しており、運転開始以来 2 年間 IO ﾎﾞｰﾄﾞの故障以外まったくﾄﾗﾌﾞﾙが無い。追加される機器の設計段階から、この信頼度の高い VME ﾚﾍﾞﾙのﾘｿｰｽを活用できるように、機器の動作仕様にｿﾌﾄｳｪｱの都合を盛り込んでｿﾌﾄｳｪｱ開発の期間短縮を図った。

全体の価格と予算の都合で、磁石電源の一部など同等の機能を果たす機器でありながら、本体既存部分と同じ機種や仕様にすることができない部分もあった。また増設部分の安全を確保するためのﾋﾞｰﾑｼｬｯﾀやﾗﾃﾞｨｴｰｼｮﾝｼｬｯﾀなど新規のﾀｲﾌﾟとなる機器もあった。これらに対応する VME ﾚﾍﾞﾙの機器制御ﾌﾟﾛｾｽは今回新たに製作することになったが、既存部分の制御ﾌﾟﾛｾｽが OOP によるｵﾌﾞｼﾞｪｸﾄとして製作されている上に、機器の種別によらない共通の状態遷移を定義して、それに沿った動作をするように

機器仕様を規定しているので、改造個所を明確に限定することができた。

線型加速器のﾏﾝﾏｼﾝﾌﾟﾛｾｽはﾜｰｸｽﾃｰｼｮﾝ上のものとして2種類ある。VME 上の制御ﾌﾟﾛｾｽと 1 対 1 に対応したﾌﾟﾘﾐﾃｨﾌﾞ画面と呼ばれるものと、運転する立場で最適化されたｵﾍﾟﾚｰｼｮﾝ画面と呼ばれるものである。ﾌﾟﾘﾐﾃｨﾌﾞ画面は、ﾎﾞﾀﾝの種類や配置、ﾌｨｰﾙﾄﾞの区分など詳細が規格化されており、異なる機種の操作でも戸惑うことが無いように配慮されている。従って新規に制作する場合も仕様を明確にできる。一方、ｵﾍﾟﾚｰｼｮﾝ画面は監視と操作が統合的に行えるように工夫されている。X11(Xlib)の機能をﾌﾙに活用しているので、ﾌﾟﾛｸﾞﾗﾐﾝｸﾞの初心者がすぐに理解できるような記述にはなっていないが、ﾌﾟﾘﾐﾃｨﾌﾞ画面のﾗﾝﾁﾝｸﾞ用ﾎﾞﾀﾝなどはﾘｺﾝﾊﾟｲﾙしなくても追加できるような工夫がなされており、今回の改造にはそれらの特徴を活用し、省力化を図った。

VME 制御ﾌﾟﾛｾｽ

線型加速器の VME 制御ﾌﾟﾛｾｽは機種によらず図 2 に示した状態遷移を持つ。

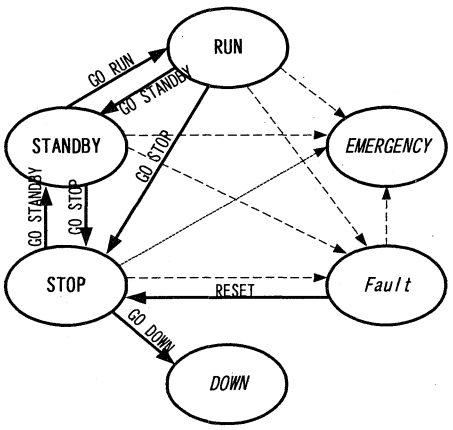

図 2　制御ﾌﾟﾛｾｽ状態遷移図

入出力の信号はｲﾝﾀﾛｯｸ(ILK)、ｽﾃｰﾀｽ(STA)、操作(OP)、ﾓﾆﾀ(MON)、ﾊﾟﾗﾒｰﾀ(PAR)に分類され、ｲﾝﾀﾛｯｸ信号とｽﾃｰﾀｽ信号によって状態と状態遷移が決まる。従って機器毎の違いはどのパスの信号がどの分類に属するか、ｽﾃｰﾀｽﾜｰﾄﾞと状態のがどう対応するか、GO ｺﾏﾝﾄﾞ受信や ILK 発生の外部事象をｲﾍﾞﾝﾄとして起動される状態遷移でﾊﾟﾗﾒｰﾀをどのように操作するかである。このうちﾊﾟｽと信号ﾜｰﾄﾞ、ﾓﾆﾀ値を物理値に変換するための校正係数等のすべての機器依存変数は、X11 のﾘｿｰｽ記述方法を模倣したわかりやすいﾌｫｰﾏｯﾄで記述した外部ﾌｧｲﾙになっており、ﾌﾟﾛｸﾞﾗﾑは一切の埋め込み変数を持たないので、それらの変更にはﾃｷｽﾄのｴﾃﾞｨｯﾄのみで対応できる。

```
/*  ｲｵﾝﾎﾟﾝﾌﾟ用ﾘｿｰｽﾌｧｲﾙ  IP1-L3.ENV  ------------*/
/*-----------------------------------------*/
/*ﾏｼﾝ・ﾀｲﾌﾟの宣言                              */
MACHINE:     IP          // ﾏｼﾝ・ﾀｲﾌﾟは IP
/*物理 I／O パスの宣言                        */
/* ｲﾝﾀﾛｯｸﾃﾞﾊﾞｲｽ */
DEV.1.TYPE:        DI              // ﾃﾞﾊﾞｲｽﾀｲﾌﾟ
DEV.1.PATH:        /di1/0.1         // ﾊﾟｽ
/* ｽﾃｰﾀｽﾃﾞﾊﾞｲｽ */
DEV.2.TYPE:        DI              // ﾃﾞﾊﾞｲｽﾀｲﾌﾟ
DEV.2.PATH:        /di1/1.1         // ﾊﾟｽ
/* ｵﾍﾟﾚｰｼｮﾝﾃﾞﾊﾞｲｽ */
DEV.3.TYPE:        DO              // ﾃﾞﾊﾞｲｽﾀｲﾌﾟ
DEV.3.PATH:        /do0/24.1        // ﾊﾟｽ
/* 真空度ﾓﾆﾀﾃﾞﾊﾞｲｽ */
DEV.4.TYPE:        AI              // ﾃﾞﾊﾞｲｽﾀｲﾌﾟ
DEV.4.PATH:        /ai0/8           // ﾊﾟｽ
/*  論理機能のﾏｯﾋﾟﾝｸﾞ                          */
/* ｲﾝﾀﾛｯｸ */
ILK.DEV:           1               // ﾃﾞﾊﾞｲｽ指定
/* ｽﾃｰﾀｽ */
STATUS.DEV:        2               // ﾃﾞﾊﾞｲｽ指定
/* ｵﾍﾟﾚｰｼｮﾝ */
OP.DEV:            3               // ﾃﾞﾊﾞｲｽ指定
/* 真空度ﾓﾆﾀ */
MON.1.DEV:         4               // ﾃﾞﾊﾞｲｽ指定
MON.1.OFS:         -6              // 校正係数(ｵﾌｾｯﾄ)
MON.1.GAIN:        0.5              // 校正係数(ｹﾞｲﾝ)
/* 真空度ﾛｸﾞ */
LOG.1.MODE:        0               // ﾛｸﾞﾓｰﾄﾞ
LOG.1.CYCLE:       10.0
LOG.1.SIZE:        128          // ﾛｸﾞ領域 MAX 件数
/* 真空度ｱﾗｰﾑ */
//ALARM.1.MODE:      0           // アラームモード
//ALARM.1.UPPER:     0.0            // 上限値
//ALARM.1.LOWER:     0.0            // 下限値
/*IP 定義終わり                               */
END:              IP         // IP の定義終わり
/*-----------------------------------------*/
```

例 1　IP 制御プロセス用定義ファイル

例えば、ある機器の機種変更でﾓﾆﾀ値がｱﾅﾛｸﾞからﾃﾞｼﾞﾀﾙに変更されても、ﾋﾞｯﾄ長が同じで状態遷移に変更が無ければﾌﾟﾛｸﾞﾗﾑのﾘｺﾝﾊﾟｲﾙは必要ない。状態遷移に伴う処理に変更がある場合にはﾌﾟﾛｸﾞﾗﾑの修正とﾘｺﾝﾊﾟｲﾙが必要となるが、その様な場合でさえ、記述した処理を呼び出す手段が厳格に規定され、扱う変数も分類等多重のﾁｪｯｸを受けるという OOP のﾒﾘｯﾄが発揮されて、予想外のﾊﾞｸﾞを生ずることは全く無かった。

線型加速器の VME ﾊﾞｽ型ｺﾝﾋﾟｭｰﾀは bootROM で起動され、必要なﾌｧｲﾙをﾀﾞｳﾝﾛｰﾄﾞしてくる。使用している OS9 の ISP の一部に手を加えて、IP に関する情報は SRAM から読むようにしてあるので、IP の変更や代替機を使う場合にも bootROM の作り直しが必要無く、すべての VME に同じ bootROM を使用できるようになっている。各 VME のｺﾝﾌｨｷﾞｭﾚｰｼｮﾝもすべてﾜｰｸｽﾃｰｼｮﾝで一括管理している。制御系ﾌﾟﾛｾｽに関するｺﾝﾌｨｷﾞｭﾚｰｼｮﾝはすべてﾜｰｸｽﾃｰｼｮﾝ上の config.linac というﾌｧｲﾙに記述され、専用ﾊﾟｰｻによる詳細なﾁｪｯｸを受けてからすべての VME のﾌﾟﾛｾｽ構成に展開される。例えば、VME を 1 台追加する場合は、次の記述を config.linac に追加するだけでよい。

```
def host {
        type=VME;          name=VME1-L3;
        ipaddr=172.24.1.32;
        bufsize=128;
};
```

同様に新しい制御ﾌﾟﾛｾｽを追加する場合には、

```
def machine {
        name=VBM1-L3;      addr=23502;
        host=VME1-L3;
        module   = control_BMBT;
        resource = vbm1_l3.env;
        bufsize  = 8;
};
```

の記述を追加するだけでよい。それぞれの制御ﾌﾟﾛｾｽはこ

のｼｽﾃﾑ内固有のﾏｼﾝ名(name)とﾏｼﾝｱﾄﾞﾚｽ(addr)を持つ。Host はこのﾌﾟﾛｾｽが動作する CPU を指す。module が機器種別毎のﾓｼﾞｭｰﾙを表し、これに機器毎のﾘｿｰｽ(resource)が与えられて実際の制御ﾌﾟﾛｾｽとして動作する。これらのｴﾃﾞｨｯﾄができたら、新規の VME をｲｰｻﾈｯﾄに接続し、専用ｺﾏﾝﾄﾞを用いて新規の IP を SRAM に書き込んでから再起動してやるだけでよい。

ﾏﾝﾏｼﾝﾌﾟﾛｾｽ

ﾏﾝﾏｼﾝｲﾝﾀｰﾌｪｰｽ(MMI)は主に HP-UX の X11 上に構築されている。MMI も制御ﾌﾟﾛｾｽと同様にﾏｼﾝ名とﾏｼﾝｱﾄﾞﾚｽを持ち、このｼｽﾃﾑ専用のﾌﾟﾛﾄｺﾙ(SCD)によるﾌﾟﾛｾｽ間通信ではまったく対等である。従って制御ﾌﾟﾛｾｽと同様にまずconfig.linac に次のｴﾝﾄﾘを追加する。

```
def mmi {
        name=BM1-L3;        addr=13502;
        host=haru755b;
        ctrl      = VBM1-L3;
};
```

ただし VME のようにｺﾝﾌｨｷﾞｭﾚｰｼｮﾝを規定する必要のない、ﾜｰｸｽﾃｰｼｮﾝ上で動作させる MMI のｴﾝﾄﾘは必須ではなく、起動するときにﾏｼﾝｱﾄﾞﾚｽをｱｰｷﾞｭﾒﾝﾄで与えさえすれば、起動時の検索時間が必要になるだけで問題なく起動できる。次に MMI のｸﾞﾗﾌｨｶﾙﾗﾝﾁｬ(gstart)の画面に新規のﾎﾞﾀﾝを追加するために、ﾍﾟｰｼﾞ毎のｳｨｼﾞｯﾄﾞﾘｿｰｽﾌｧｲﾙ(linac.all等)に以下のｴﾝﾄﾘを追加する。

```
Button   100, -10, 40, 20, name=BM1-L3, ¥
                    text=BM1-L3, pix=Bend.pm
```

【x=100,y=-10 を端点とする dx=40,dy=20 のﾎﾞﾀﾝ。文字表示として BM1-L3 が表示され、ｸﾘｯｸされると BM1-L3 が起動される】

このｴﾝﾄﾘを追加した後、ﾗﾝﾁｬを再起動すれば新しいﾎﾞﾀﾝが追加されている。次にその新しいﾎﾞﾀﾝがｸﾘｯｸされたときのｺｰﾙﾊﾞｯｸを規定する以下の記述を、ﾗﾝﾁｬ用ﾃﾞｰﾀﾍﾞｰｽﾌｧｲﾙ(gstart.addr)に追加する。

```
; gstart.addr ﾏｼﾝﾃﾞｰﾀﾍﾞｰｽ
    BM1-L3        bm  BM1-L3  13502   vbm1-L3
; 書式: name1 type(code) name2 addr vmename
;                              ﾃﾞﾘﾐﾀは TAB ｺｰﾄﾞ
; name1: 正式な機器記号(名称)
; code(type):    起動される MMI ﾌﾟﾛｸﾞﾗﾑ
;                機器ﾀｲﾌﾟと 1 対 1
; name2: MMI のﾏｼﾝﾈｰﾑ
; addr:          MMI のﾏｼﾝｱﾄﾞﾚｽ
; vmename:       対応する制御ﾌﾟﾛｾｽのﾏｼﾝﾈｰﾑ
```

これは bm という MMI 用ﾌﾟﾛｸﾞﾗﾑを、ﾏｼﾝ名 BM1-L3、ﾏｼﾝｱﾄﾞﾚｽ 13502 で、vbm1-L3 をﾀｰｹﾞｯﾄの制御ﾌﾟﾛｾｽとした MMI として起動するというｴﾝﾄﾘであり、ﾀｰﾐﾅﾙｳｨﾝﾄﾞｳから code 以降の記述を入力しても同様に起動できる。特別な操作ｼｰｹﾝｽを持つﾏｼﾝｱﾄﾞﾚｽ埋め込みのﾌﾟﾛｾｽを起動できるようにする場合は、ｳｨｼﾞｯﾄﾞﾘｿｰｽﾌｧｲﾙの name に run をﾍｯﾀﾞにして起動するﾌﾟﾛｸﾞﾗﾑ名を直接ﾀﾞﾌﾞﾙｸｫｰﾃｰｼｮﾝで囲んで与えればよい。

また、一時的な計測のためなどにﾃﾞｰﾀｱｸｨｼﾞｼｮﾝ用制御ﾌﾟﾛｾｽを増設した場合など、専用の MMI を作らずにすむように PC(Win95/NT)上の汎用 MMI もある。各信号ｶﾃｺﾞﾘ毎の定義を入力するだけで、同じ形の画面がいろいろな機器のﾏﾝﾏｼﾝとして動作するようになっている。

まとめ

大規模な増設に対応した制御ｼｽﾃﾑの改造を非常に少ない時間と手間で完成させた。ｿﾌﾄｳｪｱ作成の時間と労力と増大することは、計算機制御におけるｿﾌﾄｳｪｱの重要性の高まりから当然のことであるが、一方、急激に複雑化するｿﾌﾄｳｪｱを如何に効率よく管理し、柔軟性を損なわずに素早く構築するか、ということも重要な技術である。OOP はその点でいまや必須の技術であるが、構造化ﾌﾟﾛｸﾞﾗﾐﾝｸﾞの世代に蓄積されたﾘｿｰｽを引きずる歴史ある装置の制御系では、部分毎に OOP を取り入れてもﾒﾘｯﾄが現れにくい。その点、我々の線型加速器制御ｼｽﾃﾑは設計当初から OOP の考え方を取り入れたｼｽﾃﾑである。制御ﾌﾟﾛｾｽでは OOP 専用ｺﾝﾊﾟｲﾗこそ使用していないが、ﾌﾟﾛｾｽ設計とﾓﾃﾞﾘﾝｸﾞに OOP の概念を導入することで、そのﾒﾘｯﾄが十分に発揮されることが今回の改造で示された。

ﾜｰｸｽﾃｰｼｮﾝや PC 等の GUI 構築ﾂｰﾙは次々と新しい物がﾘﾘｰｽされ、使い物になるｸﾗｽﾗｲﾌﾞﾗﾘを備えたものも廉価で登場しつつあるので、今後もしばらくはそれらのﾊﾟｯｹｰｼﾞﾌﾟﾛｸﾞﾗﾑの評価を兼ねて MMI のﾘﾌﾟﾚｰｽが行われるであろうが、機器との入出力を扱う部分は少なくともﾊｰﾄﾞｳｪｱの寿命程度までは生き残る。ﾄﾞﾗｲﾊﾞなどﾊｰﾄﾞｳｪｱの詳細情報なしには製作できない部分をうまくｶﾌﾟｾﾙ化できれば、その寿命期間に発生する改造に最小限の投資で対応できることが示され、その方法として OOP は有効である。

このような中規模以上の制御ｼｽﾃﾑを構築する場合、ﾊｰﾄﾞｳｪｱﾒｰｶに機能仕様で要求されることの多いﾄﾞﾗｲﾊﾞなどのﾌｧｰﾑｳｪｱ的な部分の情報を確保し、ﾌﾞﾗｯｸﾎﾞｯｸｽにしないことが肝要である。機能仕様だけでなく、ｶﾌﾟｾﾙ化されているﾓｼﾞｭｰﾙに適切なﾒｿｯﾄﾞを追加できるための内部構造に関する情報を自分らで確保していれば、製品寿命の極端に短い計算機市場の潮に巻き込まれずに、適切な時期に最小限の投資で移植等を行うことも可能になる。上位ｱﾌﾟﾘｹｰｼｮﾝ作成時に見なくて済む部分があるということと、必要なときに手が出せない部分があるということは、ｿﾌﾄｳｪｱの寿命や生産性、再利用性に大きな差異となって現れる。そのための初期投資ができるか否かも重要であろう。

また、制御ﾌﾟﾛｾｽに OOP を適用する場合に、ｼｽﾃﾑのﾓﾃﾞﾘﾝｸﾞとそのﾓﾃﾞﾙの評価が機器の設計に先行することが必要である。この制御ｼｽﾃﾑではすべての制御ﾌﾟﾛｾｽのｽｰﾊﾟｰｸﾗｽとなる machine class が定義されたが、実際の機器の動作がそのﾓﾃﾞﾘﾝｸﾞと合致するための仕様を機器の設計に反映させることができたのでﾒﾘｯﾄを得ることができた。機器の設計者が想定するｼｰｹﾝｽがﾓﾃﾞﾘﾝｸﾞと合致しない場合も、本当に必要な機能仕様を抽出すると、ﾓﾃﾞﾘﾝｸﾞと整合する別のｼｰｹﾝｽで対応できることが多い。現場ではそれらの調整作業が一番大きな負荷であったかもしれない。