Ⓟ-23

Control of SPring-8 Injector Linac

H.Yoshikawa, Y.Itoh, H.Sakaki, T.Taniuchi, M.Kodera, A.Kuba,
T.Hori, A.Mizuno, S.Suzuki, K.Yanagida and H.Yokomizo
JASRI-JAERI-RIKEN SPring-8 Project Team
Kamigori-cho,Ako-gun,Hyogo-ken,678-12

Abstract

The injector linac of SPring-8 is under construction to be completed inr 1996. This linac has several plans of use not only for the injection but also for the independent use to have high efficiency of utilization. To have a reliability, an easeness and a flexibility of operation modes, it is very important to have a systematic integrated control system. We designed the control system to establish such an system, and promote an software project to realize the design concept. In the project, we made a set of prototype software which regurates not only the functionalities but the structure of each components of software modules. And it is enabled to minimize the amount of coding works. This report represents the status of the software projects.

ＳＰｒｉｎｇ－８入射系線型加速器の制御系

1　はじめに

SPring-8の線型加速器は１９９６の完成を目指し現在建設が進められている。この線型加速器は、単に入射系として利用されるだけでなく、入射時間以外の有効利用をはかるために様々な計画を持っている。いろいろな形態の運転を簡便に確実に行うためには、システムとして適切に統括された制御系を持つことが重要である。そのような制御系を持つために、我々は独自の制御系システム設計を行い、確実にその設計思想を実際の制御系に反映させるためのソフトウェアプロジェクトを進めた。機能仕様だけでなく、将来の改造や維持管理のし易さを実現するためにプログラムの構造自体も規定するためにプロトタイプを作成し、実際の膨大な制御プログラムを最低限の作業量で作成することを可能にした。

2　基本構想

制御系の設計は線型加速器ハードウェアの全体設計が決まった平成３年から始まった。制御系に求められる機能自体が明確ではなかったので、用意に改造が行えるための柔軟性はもっとも重要な点であった。陽電子発生部などの個別コンポーネントのＲ＆Ｄのなかで、計算機制御とソフトウェアの作成を経験したが、ある程度以上の規模になると、ソフトウェアの内部構造を把握して後の改造などが行えるようにすることは、従来の構造化プログラミングで制作したのでは不可能であることを実感した。また設計者と違う者がコーディングを行う場合には、設計思想を共通概念として持つことが不可欠で、思想のずれがプログラムの細部に現れるだけでも全体としての機能に重要な影響が出る場合が多いことがわかった。

計算機工学の分野では今や常識となっているオブジェクト指向プログラミング（ＯＯＰ）とは、上位のプログラムを書くときにいじることができる部分とできない部分を明確に分けるための思想であると言うこともできる。プログラムを書くときの主眼を手続きの流れからデータそのものに移し、それぞれのデータの特異性もひとまとめにして扱うことで、プログラムの抽象化を徹底して推し進めるという、プログラミングの新たなパラダイムであり、大規模プログラムにおける、複雑さの管理手法であるともいえる。

我々の制御系に対する要請はＯＯＰ抜きには実現不可能と判断した。ＯＯＰは大規模なプログラムのすばやい構築と、維持管理のし易さ、各モジュールの再利用性の高さなど優れた点が多いが、制作に取り掛かってから実際に稼働するプログラムが制作されるまでのリーディングタイムが長いという欠点がある。ＢＡＳＩＣなどで言語で容易にくめる小規模なプログラムをＯＯＰで書こうとすると無駄な時間を費やすことにな

る。つまり、ＯＯＰでは使いものになる部品が十分に揃わないとその利点を発揮することができない。ＯＯＰを適用すべき規模のシステムかどうか、あるいは利用可能なＯＯＰで作られた部品が既にあるかどうかが重要なポイントになる。

我々がこのソフトウェア構築プロジェクトを開始した頃は、制御用計算機の環境にインプリメントされたＯＯＰ言語がまだあまり出回っていなかったが、ＯＯＰはあくまでも考え方であり、言語仕様そのものではないという判断で、広い意味でのＯＯＰをＣ言語を使って実現していくことにした。そして、モジュレータや電磁石、モニタ機器などの個別機器についての機能を制御系の立場から見直して、それらに共通する部分を抽出作業から取り掛かった。そしてその抽出された抽象モデルは線型加速器全体にも適用できるものであるように配慮された。

まずこのモデリングに十分な時間をかけ適切なクラスを構築し、状態遷移に掛かる手続きも抽象化して組み込んだプロトタイプを制作する。そして次に各機器の機能を逆にこのモデルにあてはめることで実現できるように、具体的な機器の状態をソフトウェア的な状態に対応づけていく作業を行い、機器固有の手続きのみをプロトタイプに書き加えるだけで、必要なプログラムが生産される。というのがこのソフトウェアプロジェクトの基本構想である。

3　ハードウェア・プラットフォーム

ハードウェアは、特殊な専用のハードウェアは開発する必要がなく、すべて市販されているもので必要な機能は実現できるという結論に達した。

マンマシンはワークステーションのＸ１１環境を利用したグラフィカルな画面とマウスの操作で必要なすべての機能を果たせるようにする。各個別機器に対応したプロセス群は２１台のＶＭＥバス・コンピュータ上で動作させる。このＶＭＥコンピュータはモトローラ６８０３０のＣＰＵ、デジタル入力ボード、デジタル出力ボード、アナログ入力ボード（12bit レゾリューション）、アナログ出力ボード（同じく 12bit レゾリューション）、ＧＰ－ＩＢボード、ステップモータ用ボードが装填されている。ＧＰ－ＩＢボードは主に計測器のインターフェースとして利用されるもので、ボード上にＣＰＵを持つインテリジェントタイプを採用した。また、駆動部を持つ機器はトルクと精度の点から、ニューマチックでないものはすべてステップモータとした。正転逆転のパルス出力の他に上下限のリミット入力、原点入力、汎用入出力各１点を一組として１枚で２台のステップモータが制御できるタイプである。

ＶＭＥコンピュータとワークステーションは、１階層のイーサネット・ネットワークで接続されており、すべての情報がワークステーションの画面で得られる。

4　ソフトウェア・プラットフォーム

ＶＭＥコンピュータ上のプロセスはＯＳ９で作成される。特にＯＯＰによるモジュール化を適切なものにするために、デバイスドライバはすべて内作した。アナログとデジタルの入出力ボードは、アセンブラの他にＣ言語で記述されたドライバも作成された。柔軟なパスの切り方が可能で、高機能なリアルタイムＯＳが成熟してきたときの移植性も考慮されている。

ワークステーションは実際のオペレーションの操作性を反映させて、非常に多くのプロセスが長期にわたって増えていくことが予想される。そのときに基本的なプロセスの応答性を劣化させないようにするため、プロセスに優先度を設定できるＶＭＳを採用した。ただし、施設全体のルックアンドフィールの共通化と、完成度の高い開発環境を利用するために第一段階の制御系ではＵＮＩＸを用いることにした。

ＶＭＥ上のプロセスの開発もＵＮＩＸ上でのクロス開発環境を利用することでリビジョン管理を行うことにした。

5　マシンモデル

できあがった抽象化モデルをマシンモデルと呼んでいる。STANDBY, RUN, STOP, EMERGENCY, DOWN の状態とそれらの状態間の遷移を起こさせたりするための抽象化されたコマンド（ＳＣＣコマンド）が定義されている。各プロセスは、ＳＣＣコマンドの受信を機器故障などと同列のイベントとして受け取り、必要な処理を開始する。

このようなイベント駆動型のプロセス構造にする事を徹底することが重要である。あまり重要でない情報を「ときどき見に行く」というのは、あまり害にならないが、重要な情報ほど見に行かなくても割り込みで通知されるようにしておくべきである。ループで監視するというのは、ハングアップの最大要因であり危険である。したがって、ＯＯＰとそれに基づくイベント駆動型プロセスによる構成を適切に実現するためには、機器の入出力信号のあり方にもその思想を反映させな

ければならない。

機器の製作が先行し、プログラム開発が後手に回ると、この点で問題が生じることが多い。

6　通信プロセス

ネットワークを介したメッセージの授受を司るのが通信プロセスである。我々のシステムをＯＯＰの観点で見たときに問題となるのが、ＶＭＥ上のプロセスとワークステーション上のプロセスがネットワークで切り放されている点である。また、採用したイーサネット＋ＴＣＰ／ＩＰというネットワークは実時間性を持たないパケット型通信である。

我々は、プロセス間のメッセージをＳＣＣコマンドで抽象化しており、ほとんどの場合フラグメンテーションが生じない点に着目し、また、このネットワークは汎用の系とは切り放された線型加速器制御の専用セグメントになることから、再送により確度を高めたＵＤＰ／ＩＰ上の専用プロトコルを作成し実装した。クライアントとなるワークステーションと制御処理サーバとなるＶＭＥコンピュータ間の通信はプロセス設計からオペレーションプロセスの機能依存であることが明らかで、そのトラフィックも比較的容易に見積もることができる。また、専用プロトコルを用いることでこの系のセキュリティを確保することもできる。

特にワークステーション側の通信プロセスはＣ＋＋で記述され、仮想機器を含めたワークステション上のオペレーションプロセスにオブジェクトとして組み込むことができるようになっている。

7　プロトタイプ

仮想的な機器も含めて、すべての機器をプロトタイプのもつ共通の状態遷移にあてはめることで将来新たな機能を付加する場合も、純粋に新しい部分だけを追加することで必要な機能が発揮できる。

重要なのはこのプロトタイプに至るまでの、機能と構造の抽出作業であり、その過程で思想の共有化が図られる。それ以後は、プロトタイプのなかの機器固有部を記述するための箱となる local func の記述のみを許すことにすれば、おのずと型にはまらざるを得なくなる。

もともと加速器の構成要素である機器にはそれほど複雑なものはない。我々の場合、高電圧パルス放電機器であるモジュレータの状態をどのように定義するかが一番問題となった。放電素子のヒータの加熱時間という、状態遷移中のような時間が３０分から１時間と長い。また、加熱が十分であればすぐに稼働可能な状態にしたいという、手続き依存の状態定義が要求された。これはヒータ電源を含む低圧電源の入り切りを操作信号から初期条件に置き換え、余熱完了を状態判断の条件に組み込むことで対処した。

このプロトタイプは、実際の機器に対応するプロセスだけでなく、いくつかの機器を複合させた仮想的な機器に対応するプロセスにも適用される。仮想機器のプロセスが実際の機器に対応するプロセスのクラスを継承し内包しているのである。

8　オペレーションプロセス

ワークステーション上のプロセスがオペレータの操作を受けて、常駐している制御プロセスに処理要求を発行するクライアントになる。このプロセスはワークステーション上で動作するものであり、現在ＵＮＩＸ上のＣＡＳＥツール及びＧＵＩビルダを用いて制作を進めている段階である。

優れたツールのおかげで、予想以上に簡便に作成することができる。洗練された画面を構築するためにいくつかのウィジッドは作成しなければならないが、ＳＣＣコマンドと１対１に対応したボタン等のウィジッドが並ぶプリミティブ画面と、オペレータの操作や応答性を反映させてブラッシュアップしていくグラフィカル画面の作成との２段階に分けて制作を行い、当初必要となる機能を確実に確保するとともに、使いやすい環境をじっくりと練り上げる予定である。

9　まとめ

ＶＭＥ上の制御プロセス、ワークステーション上のオペレーションプロセス、ネットワークを透過にする通信プロセスの３つの構成で、ＯＯＰを実践してプログラムを構築している。このような規模のプログラムを作成するときには初期段階で少人数のチームを組み、概念を供用できるようにすることと、その概念に基づく適切なモデリングを行うことが重要である。従来の、機能仕様→コーディング→テスト→保守　という流れではなく、システム分析→(モデリング→設計)[繰り返し]→コーディング→テスト→改良　という流れに変わっており、特にオブジェクト指向によるシステム分析、設計が重要である。