1.はじめに

1.1 加速器制御の世界

加速器制御は国際会議の対象になるほど広範囲な分野を網羅する。Table 1 にあげるのは 2009 年 10 月に神戸で開催される ICALEPCS2009 のセッショントラックのリストである。

Status Reports	最新施設状況
Project Management & System Engineering	プロジェクト管理とシステムエンジニアリング
Control System Evolution	制御システムの進歩
Safety/High Reliability and Major Challenges	安全、高信頼性への挑戦
Protection Systems	保護システム
Hardware Technology	ハードウェア技術
Reconfigurable Hardware	再構成可能ハードウェア
Industrial Systems in Controls	工業システムの制御への応用
Software Technology Evolution	ソフトウェア技術の進化
Web Technology	Web 技術
Process Tuning and Feedback Systems	加速器調整とフィードバック
Operational Tools	運転ツール
Data and Information management	データーと情報管理
Fabric management	コンピューターとネットワーク管理

Table 1 ICALEPCS2009 のセッション

ICALEPCS とは、International Conference for Accelerator and Large Experimental Physics Control System; 加速器と大規模物理実験の制御システムのための国際会議の略で 2 年に 1 度開催される[1]。名前が示すとうり、大規模物理実験-高エネルギー実験検出器や、大型望遠鏡、核融合施設の制御も対象だが、加速器の制御が最大の対象である。

表からもわかるとうり、Reconfigurable hardware などハードウェア技術から、Web を使用するインターネット技術に関するセッションまで様々な技術が現代の加速器制御にに使用されている。

このテキストでは加速器制御という広範囲な分野から、一般的で重要な部分を抽出して伝えることを目的とする。ただし筆者は SPring-8 の制御システムの開発と維持管理に長年携わってきた者であり、どうしても SPring-8 の制御システムを仮定して論を進めることをご了承いただきたい。

また、現在の執筆時点で新規には採用されることのなさそうな技術、規格についても割愛した。加速器のライフサイクルと、制御で使う IT (Information Technology) のサイクルは全く異るので、古い加速器で古い技術がまだまだ現役であることも多いが、ここでは述べない。

このテキストの構成は

- •はじめに-加速器制御の考えかた
- •加速器制御で使用される標準技術
- •MADOCA の解説
- •安定運転のための技術

の順で述べる。

2.加速器制御の考え方

2.1.加速器制御の範囲

制御システムの範囲について、各加速器施設では異ることが多い。

例えば、タイミングシステムの基本的な構成は SPring-8 では、制御グループの範囲外である。また Bunch-by-bunch フィードバックも運転軌道解析グループが設計製作し、制御グループは使用されている機器の制御しかおこなっていない。

また安全のインターロックも加速器運転には重要な要素であるが、これも安全管理の部署が独立しておこなっている研究所もあれば、SPring-8 のようにその構成を制御グループが行なっている施設もある。

[1] 加速器制御の国際会議はその他にも偶数年に開催される PCaPAC (International workshop on Personal Computers and Particle Accelerator Controls, パーソナルコンピューターと加速器制御に関する国際ワークショップ)がある。

このテキストではタイミング制御や、安全インターロックなどは除外して、加速器制御の基本的な部分について述べる。

2.2.加速器制御の基本的役割り

加速器制御の基本的な仕事は主に 3 つある。

- •加速器モデルに従った値を機器に伝達し、実行させる。
- •加速器の機器が実際に意図した通り動作しているかをモニターする
- •モニターした値が意図していた値と異なれば補正させる(フィードバック)。

以上を実現し、加速器の性能を最高に引き出し、かつ安定に動作させるのが加速器制御の役割である。

加速器の寿命は長く、その間絶えざる改良が行なわれる。制御システムはその進歩に対応できるものでなければならない[2]。

かつ制御システムのコストにも敏感でなければならない。このコストには初期の導入費用だけではなく維持、管理、保守に必要なマンパワーのコストも算入しなければならない。どの施設でも加速器制御にかけられる資源は限られている。その制約の中で上記の目的に合致したシステムを製作、維持しなければならないのが加速器制御を行なう上の役割である。

3.制御システムの基本的な構成

加速器制御システムとは基本的には操作するオペレーターと操作される機器をつなぐものである。

3.1.加速器制御の標準モデル

現代の加速器制御システムでは、3 層からなる。すなわち人間が操作するマンマシンインターフェイス層と加速器各部分に分散配置されたローカルコントローラーから成るインターフェイス層、それらを結ぶネットワーク層である。ローカルコントローラーと I/O 層はローカルバスで通信している。この方式は広く採用され加速器制御の標準モデル (The Standard Model) と呼ばれることもある[1][3]。

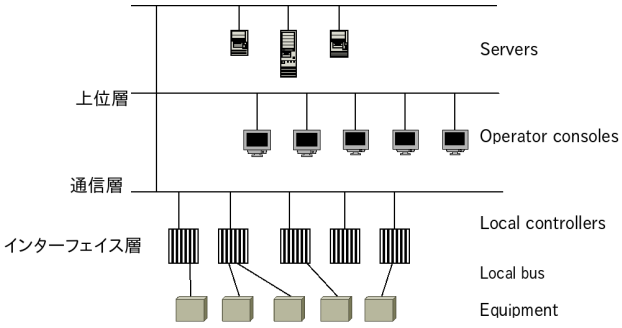

Fig. 1 加速器制御の標準モデル。

集中管理と比較してローカルコントローラーを分散配置する標準モデルの利点は

- •上位のオペレーターコンソールとはネットワークケーブルで配線するので、機器との配線が短かくてすむ。地理的に分散した大規模な加速器施設ではこれが重要である。
- •拡張性が得やすい。機器の追加にもコントローラーを追加することによって対応可能。
- •故障箇所の同定がしやすい。
- •故障の影響範囲を限定できる。
- •機器のテストもローカルで独立におこなえる。

などがあるが反面メンテナンスの際には現地に行かなければならない、などの不利な点もある。

3.2.ソフトウェア的な 3 層構造

この 3 層構造にはソフトウェア的な意味もある。オペレーターコンソールでオペレーターの対象は抽象的な加速器である。抽象的な加速器とは磁場は Tesla,加速電場は kV/m 真空度は Pa で表現されるいわば教科書の中、加速器モデルの中の加速器である。

ところが実際に加速器制御システムで制御する対象は電磁石ではなく、電磁石電源である。それも 1 つの電磁石にも複数の電源が接続されているのが普通である。高周波空洞ではなく、クライストロンやモジュレーターである。さらには機器に最終的に与えられ

[2] しかし加速器の設置形態により事情は異る。近年増加してきた、治療用加速器など、一旦完成後には加速器の専門家は不在の施設では制御システムに求められるのは安定性が第一で、拡張性はあまり考慮しなくともよい。

[3] 標準モデルの 3 層にはもう一説があり、それによるとネットワーク層のかわりに下位の機器制御層を置く。

るのは電流値(A)ですらなく DAC (Digital to Analog converter デジタル-アナログ変換器) に送られるバイナリーの数値である。 モニターすべきものは ADC (Analog to Digital Converter アナログ-デジタル変換器) のバイナリー値である。接点のオンオフにしても必ずしも 1 が on とは限らず機器によって異る。

この抽象的な加速器と実際の機器の集合である加速器とのギャップを各層のソフトウェア構造で埋める。まずオペレーターコンソールでは加速器のモデルの言葉である T や kV を各機器に命令するときに使う物理値である A や V などに変換する。その命令をローカルコントローラーに送る。

例えばある Q 電磁石の磁場を 0.7T にしたいのであれば、それを変換して、電磁石に接続されている主電源には 18.2A, 副電源には 2.02A を送る命令をローカルコントローラーに送る。このときの命令は物理値(この場合は A)によって表現される。

ローカルコントローラーは受け取った命令を解釈し、物理値をローカルに保持してある変換係数によって bit 値に変換し、各電源に渡す。またローカルコントローラーは各機器をモニターして得た値を物理値に変換してオペレーターコンソールに送る。

これが加速器制御における 3 層構造のソフトウェア的な意味である。

3.3.オペレーターコンソール

オペレーターが操作するコンピューターはオペレーターコンソールと呼ばれる。

一般的にはオペレーターが操作しやすいようにグラフィカルユーザーインターフェイス(GUI) を表面にまとったアプリケーションソフトウェアがその上で動作する。そのアプリケーションソフトウェアがネットワークを通じてローカルコントローラーに命令を発し、加速器を制御する。

SPring-8 のような大規模な加速器ではオペレーターコンソールは複数使用され、役割を分担している。表に SPring-8 におけるコンソールとその役割を示す。

役割りは便宜的に設定されたもので、どのアプリケーションどのコンソールでも動作する。 SPring-8 のような大規模の加速器施設にしてはコンソールの数は比較的少ないといえど 15 台(2009 年 7 月末現在) で構成されている。

3.4.ローカルコントローラー

機器に近接して設置され、機器に接続され制御をおこなうコンピューターは総称してローカルコントローラーとよばれる。ローカルコントローラーはフロントエンドコンピューター (front-end computer)や 組込みコンピューター (embbeded computer) と呼ばれることもある。 ローカルコントローラーはローカルバスによって機器と接続される。

SPring-8 では 2009 年 7 月末現在 301 個のローカルコントローラが加速器周辺に分散配置されている。

3.5.ネットワーク

オペレーターコンソールからの命令とローカルコントローラーからの返事を伝送するのはネットワークシステムである。

ここで以後の説明のためにネットワークの構造を表現するのによく用いられる OSI 参照モデルを説明しておく。これは ISO(国際標準化機構)によって策定されたネットワーク機能を階層構造で表現したものである。

各層とその実装を例示する。

- 7 層 アプリケーション層 加速器制御アプリケーション
- 6 層 プレゼンテーション層 MADOCA メッセージ, Channel access
- 5 層 セッション層 RPC, CORBA
- 4 層 トランスポート層 TCP,UDP
- 3 層 ネットワーク層 IP
- 2 層 データーリンク層 Ethernet
- 1 層 物理層 UTP,光ケーブル

があるが加速器制御でコンソールとローカルコントローラーとの接続に使用されるのは下位 4 層に限っては

- 物理層:光ケーブル,UTP
- データーリンク:イーサネット
- ネットワーク:IP

•トランスポート:TCP,UDP

以外使用されることはほとんどなくなったといえる。

上位 3 層については加速器制御フレームワークによって様々な実装がある。

3.6.サーバー計算機

オペレーターコンソールやローカルコントローラー以外にも重要なコンピューターが存在する。

それはファイルサーバーやデータベースサーバーさらには Web サーバーなどサーバー群で、これらも今日の加速器制御には不可欠である。

ファイルサーバーは NFS により各計算機からアクセス可能でアプリケーションプログラムや、ライブラリーの共有、データーファイルの共有も受け持つ。

データベースサーバーは現在では主にリレーショナルデータベースをパラメーターの保存、ログデーターの保存等に使用する。

Web サーバーはデーターの閲覧に使用する。このデーターとは加速器のパラメーター、ログデーターのみならず、シフト表、運転記録やマニュアルなども含まれる。Web の利点は加速器制御ネットワーク外からも、加速器のデーターが取得できることである。自室の研究室、自宅や出張先からものパソコン、携帯電話からの加速器の状態監視も Web によって加速器状況の把握ができる。

4.標準技術

加速器で使用されている制御機器は殆ど市販品で特別製作されたものでは少ない。特殊な用途、例えば高速 ADC や、リアルタイムデーター収集系など特別に製作することもある。

以下では最初にハードウェアの標準技術、ソフトウェアの標準技術をのべる。

4.1.標準規格の必要性

加速器制御では公的機関、団体で決定された規格の他にデファクトスタンダード (メーカーの自社規格が事実上の標準規格になること) を使用することが多い。これにはマルチベンダーによる入手の容易さ、また長期供給により機器の永続性にとっても利点がある。

加速器で使用される機器の使用サイクルは製品のサイクルと比較して非常に長く、製造から長くたった機器を修理、交換しようにも製造中止、さらには製造会社そのものも無くなっていたという例も珍しくない。

加速器制御機器を選定する際には製品のライフサイクルを十分に考慮する必要がある。その製品はいつまで使うのか、その時まで修理可能なのか、製品が製造中止になっても代替品はあるのか、別のメーカーでの代替品の可能性はあるのかといったことは部品を選定する際に重要になる要素である。

例えば、SPring-8 の運転開始時にはヒューレット・パッカード社製の HP-RT オペレーティングシステムが動作する同社製 VME 計算機を採用していた。ところが VME 計算機は製造中止、HP-RT がサポート中止となりそれ以後の採用と保守が困難になった。さすがに工業用だけあって、製造、サポート中止のアナウンスから実際の中止まではかなり時間があったとはいえ、全く異なったコンピュータとオペレーティングシステムを探す必要に迫られた。

最終的には Intel x86 アーキテクチャを使用してオープンソースの Solaris [2]を使用した VME コンピュータに転換した。ソフトウェアやデバイスドライバーの再開発には多大な手間がかかったが VME コンピューターの供給元は複数になり、選択の幅は増え、コスト的にも有利な選択ができるようになった。またオープンソースであることから、コミュニティからの貢献も期待できる。

ちなみに SPring-8 から HP-RT コンピューターが消えたのは、運転開始から 12 年後の 2009 年であった。

またサーバー計算機やネットワーク機器など日用品化した製品のコストダウンには目を見張るものがある。それらの規格品を適材適所で使用することが、コストダウン、信頼性、永続性を確保するこにとって重要である。

4.2.ローカルコントローラー

4.2.1.VME バス

ローカルコントローラーとして SPring-8 で最も使用されているのが VME バスを使用したローカルコントロ

ーラーである。外観を Fig. 2 に示す。機械的な仕様として Eurocard 規格のカードを採用している。19 インチラック規格で 3U の高さと 6U。9U の高さのカードがある。奥行はいずれも 160mm,1 スロット幅が 20mm である。SPring-8 では 6U のカード主に使用している。

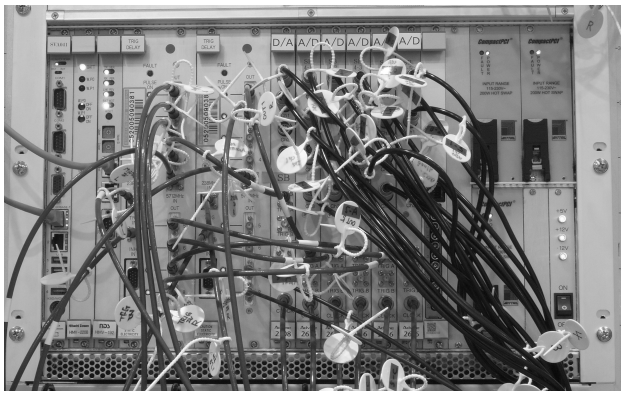

Fig. 2 VME 実装例。

バックプレーン上にコネクターが装着される(Fig. 3 エラー: 参照先が見つかりません。)。コネクターはピン型で、パソコンで使用されるカードエッジ型と比較して基盤の反りによる接触不良によるトラブルが少ないという利点がある。

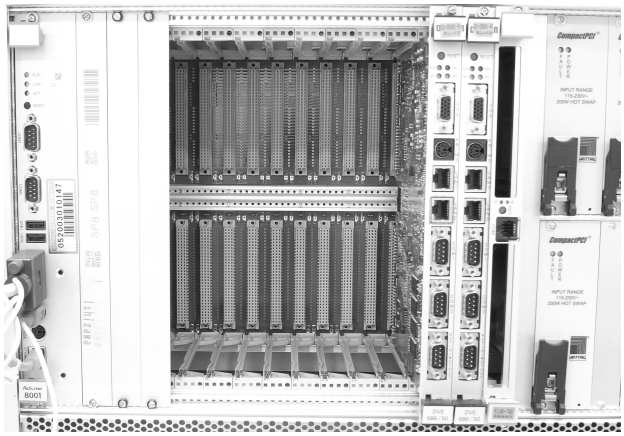

Fig. 3 VME バックプレーン。

筐体はクレートと呼ばれ縦型、横型ポータブルなものなど様々ある。電源はデジタル向 5V とアナログ用 +/-12V が標準である。

VME 規格は最初はモトローラの 68000CPU のためのバス規格として開発された VERSA 規格が元になっている。VME の名前は(VERSAmodulde Eurocard)から命名された。その後 ANSI/IEEE 1014-1987 として標準化された。使用できる CPU もインテル x86, PowerPC,SuperH などに広がっている。

当初の規格は 16 ビットバスだったが現在の規格は VME64 とよばれる 6U が 64 ビット 3U が 32 ビットバス規格である。標準的な転送速度は 40MB/s であり、画像や動画などの高速転送には不向きといえる。

派生した規格に VXI バス(VME eXtensions for Instrumentation)がある。160MB/s の転送速度があり、KEKB の BPM よみだしに使われているが、新規のシステムに使われることは少ない。

VME バスのアドレスバスとデーターバスは分離している。6U のカードは 2 つのコネクタ P1 と P2 をもつ。P1 は 32 本のピンがあり 24 ビットアドレスと 16 ビットのデーターバス、その他の制御用信号をもつ。P2 にはもう一列のピンがあり、残り 8 ビットのアドレスと 16 ビットのデーターを受け持っている。

VME はどのスロットもマスターにもスレーブにもなれる。バックプレーンは単純だがそのぶん各カードに複雑なコントローラーが必要になる。

VME はその信頼性や互換性が評価され、1981 年の登場以来、最初はワークステーションやサーバーコンピューターのバスとして使用された後、現在は加速器制御以外でも FA、軍事などで広く使用されている。

4.2.2.CompactPCI

VME と似たような規格に CompactPCI がある [3]。これは Intel の策定した PCI バスをもとに信頼性の高いコネクターや筐体を使用して組込み用コンピューターに使用できることを目指した規格である。

VME と同様に 3U または 6U の Eurocard 規格カードを使用するがピンのピッチが 2mm であることが 2.54mm ピッチの VME とは異る。標準ではバスの負荷の関係から 1 セグメントあたり 8 スロット接続できるがブリッジにより複数のバスを接続することが可能である。VME と比較して高速なバスで 132MB/s あり動画処理にも対応できる。

組み込みコンピューターベンダーも Compact PCI 新製品の投入に熱心で、事実上 VME バスの新製品投入をやめて Compact-PCI に主軸を移してしまった

有力ベンダーもある。VXI バスも事実上 CompactPCI に駆逐されてしまったといえる。

4.2.3.その他のローカルコントローラー

マイクロプロセッサーの高度化にともない、以前ならVME コンピューターにしかできなかったような高度な制御も安価なモジュールに任せられるようになりつつある。

一例を挙げるとPC/104 規格のボードがある。これは ISA バスと PCI バスをもった小さなサイズ(90.17mmx95.89mm) のコンピューターで、モジュールを組合せる際には VME のようなバックプレーンではなく、積み重ねて使用する。

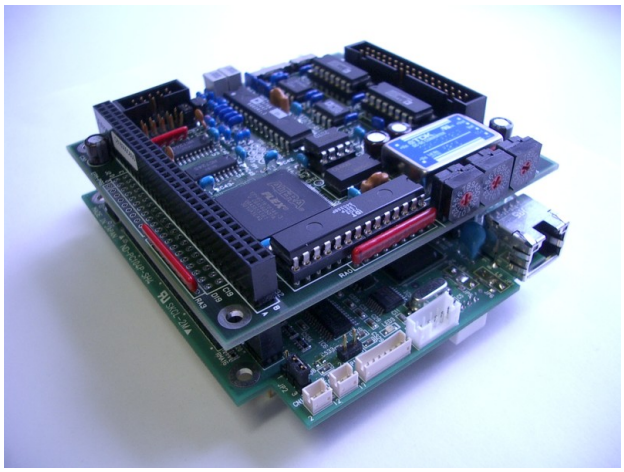

Fig. 4 PC-104Plus. スタックされた下のモジュールが CPU.

SPring-8 では PC/104 の可搬性をさらにたかめるために、PoE (Power over Ethernet) で動作する PC104plus の CPU モジュールを開発した[4]。

PoE は最大 15.4W の電力を Ethernet ケーブルから供給できる規格で、小電力のモジュールなら電源の配線が不要になるという利点がある。さらには cpu がハングした際の電源のオンオフもネットワーク経由でスイッチングハブに命令を送って実現できるなどメリットもある。

4.2.4.組み込み Windows を使ったローカルコントローラー

組み込み機器の中には、その中に Microsoft Windows を組み込んだものもある。ハイエンドのオシロスコープなどにその採用が広がっている。当然イーサネットポートを持っているので組み込み機器そのものをインテリジェントなローカルコントローラーとして使用することも可能である。

SPring-8 でもオシロスコープ組み込みの Windows に VME コンピューターと同等な機能を移植しバンチ毎の電流測定サイクルをオシロスコープ内で解析することで、測定サイクルを 30 秒から 7 秒に短縮させた。Windows OS に後述の SPring-8 での制御フレームワークである MADOCA の機能をもたせるためには cygwin を使用して Unix で開発されたソフトウェアに移植を容易にした。

4.3.ローカルバス

ローカルコントローラーと機器を接続するローカルバスには、標準品は少なくベンダーの独自規格を使用する例が多い。

標準品には PLC (Programmable Logic Controller)、GPIB (General Purpose Interface Bus) や近年採用する例が増加してきた Ethernet などがある。

4.3.1.PLC

SPring-8 で使用されているローカルバスで標準的なものは PLC と呼ばれている Programmable Logic Controller がある。シーケンサとも呼ばれることもあるがが、これは三菱電機の登録商標である。外観を図で示す(7)。

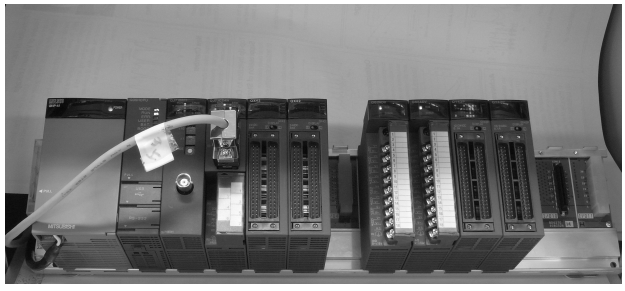

Fig. 5 PLC 外観。

PLC はリレー制御をマイクロプロセッサーにより置き換えることを目的としたもので、1968 年アメリカの自動車工業界からの要求によって設定された。

豊富な I/O モジュールを組合せることにより、自動制御を必要とする工場、工作機械、ビル制御(空調や

エレベーター、自動ドア)など広範囲に使用されている。。

PLC はその信頼性や機器の制御に特化したソフトウェアの扱いやすさから加速器制御にも広く使用されている。

4.3.1.1.PLC のソフトウェア

PLC にはマイクロプロセッサーが使用されるとはいえ、そのソフトウェアはワークステーションや VME など”普通”のコンピューターとはかなり異る。 PLC のソフトウェアはリレー回路を置き換えたラダー(ladder はしご)図と呼ばれる特殊なプログラムにより作成される。ラダー図はプログラムというよりリレーの配線図に似ている。Fig. 7 にラダー図の例を示す。ラダー図は各社が提供するパソコン上で動作するツールによって PLC 上で動作するプログラムに変換される。

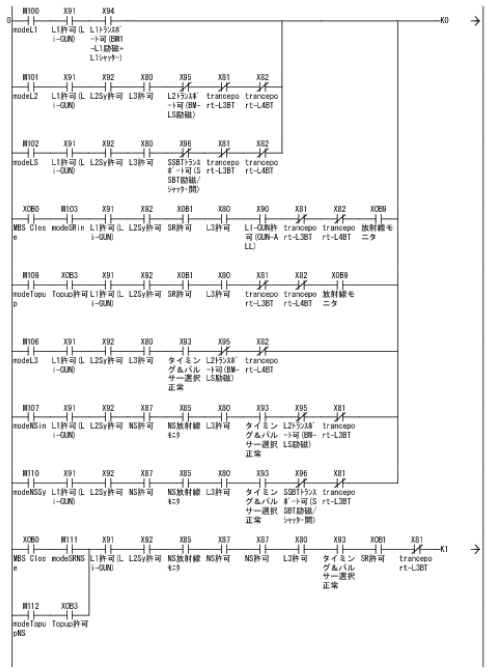

Fig. 6大規模ラダー図の例。多数のスイッチからインターロック許可信号を生成する。

簡単なラダーを説明すると左から右に スイッチ 1 がオンになりスイッチ 2 がオンになるとモーターが動く。

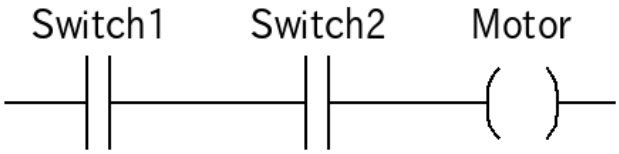

Fig. 7 簡単なラダー図。

そのラダー図もベンダーによって微妙に異ることが多く、PLC のベンダー変更を難しくさせている。

近年はラダーのみならず C でもプログラムできる PLC が商品化されている。大規模なラダー図は

4.3.1.2.PLC のハードウェア

PLC はプロセッサーモジュールと I/O モジュールからなり、用途により I/O モジュールを自由に組み合わせてシステムを構成する 大規模な場合は I/O モジュールをネットワークで接続することも可能である。その際にもデバイスドライバーは不要でブロックのようにモジュールを組み合わてシステムを構成することが可能である。

プロセッサーモジュールは信頼性を確保するためハードディスクは使用せず、近年はフラッシュメモリーにプログラムやデーターを保持する。

また表示やユーザーインターフェイスとして液晶タッチパネルを使用することも多い。

Fig. 8液晶タッチパネル。

4.3.1.3.PLCとの通信

PLC はローカルコントローラーとして上位のワークステーションとの通信、また PLC のプロセッサーモジュール間の通信は標準的には RS-232C というシリアル通信規格を用いてきたが、速度の点で問題があった。

SPring-8 では FL-net [5]という規格を使用して PLC-VME ホスト間、PLC-PLC 間の通信をおこなっている。 FL-net とは日本の自動車工業会の要求によって誕生した規格で、通信には Ethernet,UDP を使用する。前述の OSI 参照モデルでいえば物理層がイーサネット、ネットワーク層が IP,トランスポート層が UDP, セッション層が FL-net である。

日本電機工業会規格や JIS によって標準化されており、ベンダー間の相互運用も日本電機工業会の認証によって保証されている。

1 つの FL-net には最大 254 までのノードが接続でき、マスタースレーブの区別はない。すべてのノードが平等に順にトークンと呼ばれる送信権を回して、同時に全てのノードでトークンの喪失などの監視をおこなう通信方式をとっている。 そのためどのノードでも自由に電源のオンオフができ、またイーサネットでおきる通信のコリジョンも避けられるのでリアルタイム性が確保できる。

FI-net には 2 つの転送モードがある。1 つは各ノードが同一データーを常に保持できるサイクリックモード、1 つは必要な時にデーターを転送できるメッセージモードである。

FL-net をもとにした PLC ベースの加速器 制御システムも存在する[6]。

SPring-8 では VME に接続される SCSS のために理研播磨研究所が開発した FLnet-VME ボードを通じてオペレーターコンソールとの通信をおこなっている[7]。 性能は 50ms/32node(2kbit+2k word)の高速転送が得られる。

FL-net の他にも建物管理用に使用される BAC-net 等の規格がある。BAC-net が加速器制御で使用されることはないが、加速器を収納している建物の状態をモニターする機会があるかもしれない。

4.3.2.GPIB

GPIB(General Purpose Interface Bus)は 1960 年代後半にヒューレット・パッカード社により開発された HP-IB (Hewlett-Packard Instrument Bus) をもとにした 8 ビットパラレルの信号をデイジーチェーン方式で 15 台まで接続できる規格である。コネクターの形状を示す。

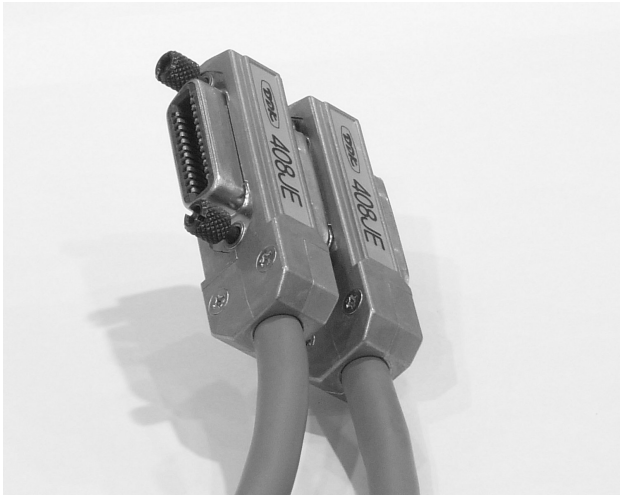

Fig. 9 GPIB コネクター: 2 個のコネクターが接続されている。

主に測定器の制御とデーター取得に使用され、速度は 1MB/sec から 8MB/s(IEE48801.-2003)に強化されたが、接続している中で最も遅い機器によって決定されてしまう。

IEEE 488.1 で設定された規格は機械的、電気的な仕様を決定しているだけで命令体系は供給者に任されていたため、各社バラバラな命令で運用しなければならない。 これは IEE488.2 で改善されたものの IEE488.1 にしか準拠していない製品が多く、開発、メンテナンスを難しくしている。

さらに SPring-8 では長期運用でトラブルが出ることも多く、現在でよほどの事情が無いかぎりは使用していない。

4.3.3.Ethernet

これも近年の普及からコストメリットもあり普及が顕著である。しかしローカルバスとしての規格はトランスポート層までで、IEEE488.1 と同様各社不統一の命令体系を受け入れざるをえない。

SPring-8 ではベンダー間の命令体系をデバイスドライバーレベルで吸収しデバイスファイルを読み書きするように、各ベンダーの機器を制御できるようにしている[8]。

4.4.ネットワーク

ネットワークでは Ethernet が標準である。 現在の標準は 100Mbps, 1Gbps(GbE)。10Gbps の Ethernet も登場しているが、コネクターの規格も GbE までのものとは互換性が無いなど、GbE までの規格とは別物といってよく、基幹ネットワーク以外には普及は進んでいない。

4.4.1.メタル(金属)ケーブル

GbE 以下のメタルケーブルによる接続には Unshielded Twisted Pair (UTP)ケーブルが使用される。 ツイステッドペアケーブルは上限周波数によりグレードがわけられ、分類名はカテゴリー category と呼ばれる。 GbE にはエンハンスドカテゴリー 5 またはカテゴリー 6 を使用する。コネクターの規格は RJ45 を使用する。

4.4.2.光ケーブル

光ケーブルはシングルモードとマルチモードに大別され、構内での比較的長距離の通信にはマルチモード光ケーブルを使用する。コア径 10μm、50μm、62.5μm のものがあり GbE でも 550m 程度の通信が可能である。

シングルモード光ケーブルは長距離の通信が必要なときに使用する。

4.5.オペレーターコンソール

オペレーターコンソールには特別な標準はない。ただ 24 時間の連続運転をするものなので、耐久性についての配慮は必要である。

SPring-8 では x86 アーキテクチャで Linux が動作するサーバー仕様のものを採用して安定運転を実現している。

4.6.サーバー

サーバーについても標準はない。Unix サーバーが web サーバー、データベースに使用されることが多い。

ファイルサーバーについては Unix サーバー+DAS (Diirect Attached Storage) という組み合わせが一頃の主流であったが、専用のアプライアンス型のものを使用する例が増えてきた。アプライアンスとは家電製品のことで、家電製品のようにセットアップの手間がほとんどかからない単機能のサーバーを総称する。アプライアンス型サーバーの導入コストは Unix サーバーと比較して高価であることが多いが、運用、拡張のメンテナンスにほとんど手間がかからないことを考えれば長期のコストでみると割安になることもある。 加速器制御にかけられる人員の少さと、年々増加するデーターの量から考慮するとアプライアンス型サーバーの導入は有効といえる。

4.7.オペレーティングシステム

加速器制御には Unix 系とよばれるオペレーティングシステムが使用される例が多い。近年では Windows を使用する施設も増加してきた。

Windows 系でのプログラム開発ツールに慣れた技術者の増加や Windows にしか対応していないデバイスを使用しなければならないといった事情もある。

SPring-8 ではオペレーターコンソールには Linux、ローカルコントローラーには Solaris といった Unix 系のオペレーティングシステムを採用している。

Windows 系ではなく Unix 系を使用しているのは歴史的経緯、プログラミング環境の充実、オープンな環境による永続性、安定性、セキュリティなど様々な理由がある。

4.7.1.リアルタイムオペレーティングシステム

加速器制御ではリアルタイムオペレーティングシステムが使用されることが多い。リアルタイムオペレーティングシステムとは時間性能に特化したオペレーティングシステムである以下の条件を満たす。

- 命令が実行完了するまでの時間が予測できること。

•高優先度と指定されたタスクが確実に実行できること

•割り込みから実行までの時間が予想できること

普通の (非リアルタイムの) Unix オペレーティングシステムではこれらを満すことはできず、同じ命令の実行時間がばらつくこと、他のタスクの影響で、実行時間の予測がつかないことを仮定しなければならない。

これに対してリアルタイムオペレーティングシステムはこれらの実行時間が予測可能になる。

注意しなければならないのは、リアルタイムオペレーティングシステムは必ずしも速いオペレーティングシステムということではない。実行は遅いかもしれないが、ある時間で確実に仕事を終えることを保証するのがリアルタイムオペレーティングシステムである[4]。

加速器制御で使用される Unix 系リアルタイムオペレーティングシステムは商用の VxWorks[9]が有名であるが他にも Linux をリアルタイムに改造したものがある。 他にも小規模組み込み系用のリアルタイムオペレーティングシステムが市販またはフリーで入手可能である[5]。

SPring8 では VME 上で主要なタスクを確実に実行させるため リアルタイムオペレーティングシステムの Solaris を使用している。

4.8.ソフトウェアフレームワーク

加速器制御に使用される標準的なソフトウェアフレームワークについて説明する。

殆どのフレームワークは加速器制御の標準モデルにしたがってオペレーターコンソールとローカルコントローラーからなるシステムを制御することが共通している。

各フレームワークはオペレーターコンソールとローカルコントローラーの通信 (セッション層、プレゼンテーション層、アプリケーション層)、ライブラリー群などである。

4.8.1.クライアントサーバーモデル

いずれのフレームワークもクライアントサーバーモデルを使用している。クライアントサーバーモデルとは、特定の役割をうけもつサーバーを複数(または単数)用意し、利用者はクライアントコンピューターからネットワークを介してそれらのサーバーに要求を出し、そのサーバーから再びネットワークを介して結果を得るという分散コンピューティングのためのモデルである。

4.8.2.その他の分散コンピューティングモデル

これ以外にも分散コンピューティングモデルとして P2P(peer to peer)モデル、つまり役割を固定せずネットワーク上のコンピューターがどちらの役にもなりうるモデルもある。

また publish-subscribe (出版購読) 型モデルもはデーターを発信するコンピュータが宛先を特定せずにデーターを出版 (publish) し、その内容に応じて予め購読予約 (subscribe)をしていたクライアントにデーターが配信されるというモデルである。

出版者と購読者との結合度が低いためスケーラビリティが良く、動的な構成にも対応しやすいという利点がある。 最近 EPICS をはじめ、加速器制御にもこのモデルを使用する動きがある。クライアントサーバーモデルはここ 20 年ほどの変らぬ主流であるが、新しい動向にも着目しておくのは重要である。

4.8.3.加速器制御でのクライアントサーバー

加速器制御でのクライアントはオペレーターコンソール、サーバーは要求をうけとり、結果を出すローカルコントローラーである。 他にファイルサーバー、データベースサーバー、Web サーバー等を使用する。

---

[4] はっきりした定義はないが、確実にタスクを実行できるものをハードリアルタイム OS、統計的に時間内のタスクを実行できるものをソフトリアルタイム OS と区別することもある。

[5] リアルタイムオペレーティングシステムは、今まで組み込み機器が主戦場であった。ところが近年、金融取引の分野でのリアルタイムオペレーティングシステムが注目されている。巨大銀行がコンピューターを使用して 10ms 単位の速度で取引することにより巨大な収入を得ていることが報じられている。この超高速取引はリアルタイムオペレーティングシステムによって可能になった。リアルタイムオペレーティングシステムの発展は今後このようなところでも期待される。

4.8.4.EPICS [10]

このフレームワークは現在の加速器制御界の主流であり、のみならず大型望遠鏡制御、検出器制御などでも採用が広がっている。

開発元は米国のアルゴンヌ研究所とロスアラモス研究所であるが、EPICS Open License のもと米国を中心に世界中で使用、開発が活発になされている。またその成果を共有することによる好循環が生じている。

使用している施設は米国以外でも KEK-B, J-PARC, RIBF (理化学研究所)、KSTAR (韓国), BSRF (Beijing Synchrotron Radiation Laboratory、中国)、SSRF (同)、 DESY(独) BESSYII,(同) Diamond(英)等世界中に広がっている。

ドキュメントライブラリーもしっかりした、加速器制御フレームワークの代表といえる。

4.8.4.1.EPICS record

EPICS ではローカルコントローラーは IOC (Input Output Controller) と呼ばれる。 IOC 上には record をあつめたデータベースがある。

Record とは機器の状態を表わす。record には多数の種類があり、ユーザーの独自定義もできる。
record の例として

- AO,AI アナログの I/O レコード
- BI,BO バイナリの I/O レコード機器の状態や命令を表わす。
- Stepping motor 文字通り stepping /motor の制御用レコード

などがある。
各レコードには scan time が定義されていれば、その時間間隔で処理される。定義されていなければ処理されない。イベント発生時にも処理することもできる。

4.8.4.2.Channel access

IOC のデータベースには channel という概念でアクセスする。channel はレコード名 | レコード名 '.' フィールド名で定義される。多くの場機器名:信号名の名前が使用される。例えば、

- S1:VAC:reading 3.2e-08 torr
- LINAC:BPM4:xPosition-0.323 mm
- BOOSTER:gateValvePosition 'OPEN'
- S3:DIPOLE:PS:setPoint 123.4 Amps
- APS:Mode 'Stored Beam'
- BL3:HISTOGRAM {3, 8, 1, 2, 56, 44, 32, 43, 3, 5, 1}

Torr, Amps などの単位は".EGU"フィールド、値の数字が ".VAL" フィールドにある。CA ではこれらをまとめて取得するための API が定義されている。

通信プロトコルは CA channel access と呼ばれるバイナリープロトコルで行なう。

4.8.4.3.オペレーターコンソール用ソフトウェア

その他オペレーターコンソールで実行される多様なソフトウェアとライブラリーが用意されている。 EDM (Extensible Display Manager) や MEDM (motif based editor/display manager) といった GUI 生成用ツールがあるのでこれらから GUI プログラムを作成できる。

4.8.4.4.EPICS の動作環境

IOC ではクリティカルなリアルタイム性が必要なときは RTEMS や VxWorks のような本格的リアルタイムオペレーティングシステムが使用され、リアルタイム性がそれほど要求されないときは Linux や Windows が使用されることも多い。

また豊富な言語バインディングが準備されている。MATLAB、LabVIEW、Perl、Python、Tcl、ActiveX などから CA にアクセスできるので、これらのスクリプトで機器を制御することも可能である。

4.8.5.TANGO

TANGO[11]は ESRF で開発された加速器制御用フレームワークで主にヨーロッパの放射光施設で使用され LGPL や GPL のオープンソースでライセンスされている。

TANGO は CORBA (Common Object Request Broker Architecture) [12]を通信プロトコルに使用している。 実装は omniORB[13]というオープンソースのライブラリーを使用している。

CORBA は別のコンピューターにある object instance を使用できるようにする標準の規約で、MADOCA で使用されている ONC-RPC が関数呼び

出しをモデルにしているのに対して、CORBA ではオブジェクト呼び出しが特徴である。

ローカルコントローラー内のプロセスは Device Sever と呼ばれハードウェアへのアクセスを device class として実装する。

実行時には device server はハードウェアを表現するインスタンスを作成し、クライアントはその離れた場所にあるインスタンスに対して要求を送るというメカニズムである。CORBA のおかげでインスタンスの場所については考慮する必要はない。

4.8.6.LabVIEW

小規模な加速器制御に使用される機会が増えてきた、グラフィカルなプログラミングができる National instruments 社製の LabVIEW[14]である。

上記 2 フレームワークが加速器制御の標準モデルのためのフレームワークであったのに対して、LabVIEW はコンソールに直接ローカルバス用のカードを取り付ける場合もサポートしている。

LabVIEW は最初は Macintosh で GPIB 機器を操作するためのソフトウェアとして開発された。現在は Windows の他に Linux 上でも動作する。また対象とする機器も飛躍的に増加し最近では EPICS の CA もアクセスできるようになった。

LabVIEW は開発環境の画面上でアイコン化された VI(Virtual Instruments、仮想機器)の間を線で繋ぐことにより機器間のデーターフローを表わしプログラミングをおこなう。For や if のような制御構文は画面上の長方形として表現する。

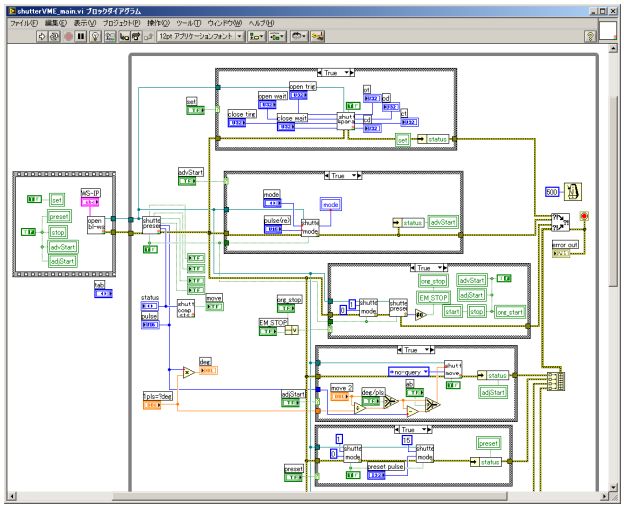

Fig. 10 LabVIEW での開発画面

LabVIEW をコンパイルしてスタンドアロンシステムとして実行することも可能である。容易なプログラミングでシステムを組むことができるので小規模な加速器制御システムを構築する際には選択肢となるであろう。

1 つのプログラムは単独で実行される以外に 1 つの VI として他のプログラムでも再利用できる。また互いにデーターのやりとりがないループは別スレッドで実行されるのでマルチコアの CPU の性能を発揮させやすいという利点もある。

[15]には LabVIEW と PLC を使用してごく少人数で制御システムを組み上げた例がある。

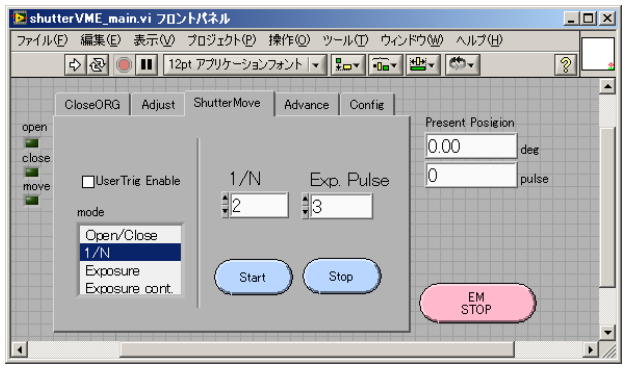

Fig. 11 LabVIEW で開発した画面

4.8.7.その他のフレームワーク

DESY の FEL 施設(Tesla,flash)で開発、使用されている doocs (The distributed object oriented control system)などがある[16]。

4.9.通信

加速器制御ではトランスポート層まではほぼ Ethernet+TCP(UDP)/IP 以外の規格は絶滅したといえる。

4.9.1.Ethernet

Ethernet の技術を詳述することはここではできないので加速器制御する際に留意しなければならない点を述べるに留める。

Ethernet はバス型の通信方式として誕生した。バス型とは同じセグメントのバスに接続された機器には

同じ信号が配信されるということである。このセグメントという用語も 1 つのバスに物理的に接続され、各ノードは電気的に等価な領域という意味でしたが、今は論理的な意味で用いられている。現在はスイッチングハブを中心としたスター型のトポロジーで接続されることがほとんどだが Ethernet のプロトコル自体は論理的にはバス型なので、加速器制御に使用するときには、そのことに留意する必要がある。

バス型の通信方式のイーサネットを特徴づけるのが CSMA/CD (Carrier Sense Multiple Access/Collision Detection キャリア検知多重アクセス/コリジョン検出方式) である。同じセグメントに接続されたノードは常にネットワークの信号を監視してる。信号が途絶えてからある時間の後、信号を送出することができる。これがキャリアセンスです。多数の端末で 1 つの伝送路を共有することを multiple access という。

この方式では複数のノードから同時に信号が流されることは避けられない。これが信号の衝突(コリジョン)ある。衝突を検出した際には特殊な信号(ジャム信号)が発生させる。ノードはこの信号を検知するか自身の信号の乱れを検知したとき、信号の送出をただちに中止し、信号は送信前の状態の戻される。
信号の送出を中止したノードは乱数によるランダムな時間待ち、伝送路が空いていれば送出をおこなう。再び衝突が起きれば、2^n個の待ち時間候補からランダムに選ばれる時間待つ。

以上の説明からわかるとうり、Ethernet には時間的な確実性、即ちリアルタイム性は無い。

前述の FL-net はこの制約の中でリアルタイム性を追求したといえるが、ネットワークでリアルタイムを求めるなら、Ethernet を使用しない方法などを考慮しなければならない。また Ethernet を使う場合でも、非リアルタイム性が目的に影響を及ぼさないことを十分なテストをおこなうことが求められる。

4.10.プログラミング言語

加速器制御の世界でよく使われるプログラミング言語と、制御に使用する際の注意をあげる。

4.10.1.C/C++

インタープリターでの実装も存在するがほとんどはコンパイルして機械語に変換して使用する。高速で動作し、機械語にも近いのでローカルコントローラーのプログラムによく使用されるがコンソール用の GUI プログラムにも使用される。

実装としては GNU による gcc[17]が標準として使用されることが多い。gcc もバージョンによって機能が異るが、永続性を考慮するならば、なるべく保守的な機能しか使用しないことが勧められる。

SPring-8 ではオペレーターコンソールを HP-UX の cc から gcc に、VME ソフトウェアも HP-RT からソラリス上の gcc に移植したがどちらも特定のバージョンやライブラリーに依存する特殊な機能を使うことが少かったため移植は比較的スムーズに実行できた。gcc は文法に厳格なため移植の過程でソースコードの信頼性も向上できた[18]。

C は柔軟な書式に対応して、かつプログラミングの自由度も高いので、見た目に汚く、デバッグもし難い複雑怪奇なプログラムを製作することも容易である。そのため多人数で C を使用して大規模なプログラムを作製する際には、最初にプログラム規定を作製してそれを厳格に守る必要がある。

4.10.2.Python

加速器制御で最もよく使用されるスクリプト言語が python[19]である。Python SoftwareFoundation License というフリーのライセンス形態のもとで配布されている。

加速器制御フレームワークでも使用できるようにライブラリーの整備もなされており EPICS, TANGO, MADOCA にもインターフェイスライブラリーが用意されている。

Rapid prototype language などと呼ばれているように C や JAVA と比較しても高速で開発ができるのがおおきなアドバンテージである。反面 C などコンパイル言語と比較してかなり遅いという欠点もあるが、近年のハードウェアの進歩がそれを補っている。またどうしても遅いときにはその部分のみを C で作製したライブラリーの置き換えるという方法で高速化を図る方

法もある。これについても swig など開発しやすい環境が用意されている。

KEK のオペレーターコンソールや SPring-8 の Web プログラミングなど幅広く使用されている。

4.10.3.その他のプログラミング言語

他にも JAVA もコンソール用のアプリケーションに使用される例が多い。

4.11.GUI

オペレーターコンソールで動作する制御用のアプリケーションのほとんどはグラフィカルで多くの情報が一望できるグラフィカルユーザーインターフェイスによって操作する。

見た目が美しく、かつ操作の容易な GUI は製作が難しいが開発する際グラフィカルな操作でソフトウェアが製作できるツールの採用でこの困難さが幾分軽減される。 SPring-8 では C のコンソール用アプリケーションのために X-mate[20]というビルドツール+ライブラリーの市販品を採用している。また python の GUI プログラムには wxPython[21], PyQT[22]などが使用されることが多い。

4.12.バージョン管理システム

ソフトウェア開発を進めると、ソースコードを編集していくとにき、実際に動作することが確認されたソースコードを残しつつ新バージョンのソースコードを編集していくことが多い。

こういった場合、たとえば source.c.090807 などと日付入りで古いソースを管理していくのも 1 つの方法ではあるが、効率も悪く、古いバージョンに戻るとき、どのソースコードを使用すべきなのか混乱することもある。

それを避けるため、大規模なグループでソースソースコードを開発するときのバージョン管理ソフトウェアがある。

バージョン管理ソフトウェアは基本にリポジトリという一種のデータベースが存在し、ソースコードを編集するときは、そのリポジトリからソースコードをローカル環境に引き出す。この操作のことをチェックアウトと呼ぶ。編集が終って終れば再び新バージョンとしてリポジトリにソースコードを預け入れる。この操作をチェックインと呼ぶ。複数の開発者が同じファイルをチェックインした場合にはエラーが出るので、その問題は開発者同士で解決しなければならない。

リポジトリには誰が何時どのような操作をおこなったかがソースコードとともに収納されているので、後から古いバージョンのソースをチェックアウトすることも可能である。

バージョン管理システムはリポジトリが 1 つであるか複数あるかで大別され前者は集中型後者は分散型と呼ばれる。集中型でよく使用されるのが CVS[23]や Subverson[24]であり、分散型では git[25]や Mercurial[26]に人気がある。

5.SPring-8 制御システム

5.1.MADOCA 総論

SPring8 のストレージリングを制御するために 1994 年から開発された制御用フレームワークである[27]。MMessage and Database Oriented Control Architecture の意味で、message と呼ばれるシンプルな命令体系とリレーショナルデータベースを用いた矛盾のない、拡張性にとんだ制御システムの実現を目指して設計された。

MADOCA もまた加速器制御のスタンダードモデルを基本にしている。オペレーターコンソールとローカルコントローラがメッセージをやりとりして制御を行う。

MADOCA の設計思想は SPring-8 での制御プログラムの製作体制に深く関係している。 SPring-8 では制御用アプリケーション(オペレーターコンソールとローカルコントローラー用)の製作は加速器や機器の専門家が行ない、制御を担当するグループはそれに必要なツールの提供をおこなうという体制であった。機器の専門家はプログラミングの専門家ではない。なるべく短い学習期間で多くの機器専門家がアプリケーションを作製できるようなフレームワークであることが要求された。

そこで MADOCA ではメッセージを送り、返事を返すコマンドのみを提供している。機器のエキスパートはアプリケーション中メッセージを作成し、それを MADOOCA ツールで送り、返事を受け取るだけで、

機器の制御、モニターが可能になる。メッセージも機器のエキスパートにわかりやすくシンプルな文法で作製できる。機器のエキスパートは他にもローカルコントローラー用の制御用プログラムを担当するが、これも上位とのインターフェイスをメッセージのみでおこなう。

5.2.メッセージ

MADOCA の基本になるメッセージは 255 文字までの ASCII テキストで構成されている。メッセージは OSI 参照モデルでいうプレゼンテーション層である。ASCII テキストにしたためバイナリーのメッセージと比較してデバッグ時の可読性に優れているが反面解釈に時間がかるという欠点もある。

文字列は英文法に習って 主語(S)/動詞(V)/目的(O)/補語(C)の 4 要素が/(slash)によって区切られている。この 4 要素のみを使用し、要素数が増減することは無い。

- S:プロセス名、アプリケーション名、プロセス id、ユーザー名をアンダースコア(_)で繋いだ文字列。
- V:get,set,put
- O:対象となる機器
- C 対象となる機器の信号

アプリケーション内部では S を指定する必要は無い。MS が自動的に付加する。
例えば
V/O/C の名称例

get/sr_vac_ivg_11_cr1pressure
get/sr_mag_ps_q_aux_1_1/current_adc
get/sr_mag_ps_q_aux_1_1/current_dac
get/sr_mag_ps_q_aux_1_1/status
set/sr_mag_ps_q_aux_1_1/1.234A
put/sr_mag_ps_q_aux_1_1/on

Table. 2. V/O/C の例

O の機器名 sr_mag_ps_q_aux_1_1 は

- sr:蓄積リングの
- mag:電磁石グループの
- ps:電源
- q: Q マグネット
- aux:補助電源 1 セル 1 番目

という意味である。
V は put get set の 3 つのみ。

- put:値を送って実行する
- set:は値を送って実行しない
- get は値を得る

このメッセージを処理する関数は

- ms_open
- ms_option
- ms_send
- ms_send_ed
- ms_recv
- ms_set_recv_timeout
- ms_close

のみ。頻繁に使用する関数は ms_send,ms_recv の 2 関数のみである。

使用できる値は単位付きのアナログ値(5.02A, 10e-8Pa 等)、状態ビット列、ファイル名である。大量のデーターのやりとりは NFS サーバー上のファイル名を指定することで行う。

状態ビット列は機器の様々な状態を 1 つの値あたり 30 ビットまでまとめて収納するものである。上位 2 ビットはエラー処理のための予約ビットで。各ビットの意味はデータベースによって管理する。以下にある信号(電磁石電源) の状態値のビット内容を例示する。

1. On:1, Off:0
2. Ready:1, Not Ready:0
3. Remote:0, Local:1
4. Fault:1
5. Over Current:1
6. Over Voltage:1 (cont. to run)
7. Over Temperature:1
8. Magnet Interlock:1
9. Water Off:1
10. Fuse Down:1
11. Oven Error:1 (cont. to run)
12. Polarity+:1
13. Smoke Detect:1

14. Ground Fault:1
15. Warning:1 (cont. to run)

以上の 15 ビットの情報もデータベースによって管理されている。

5.2.1.メッセージの処理

アプリケーション内では ms 関数がメッセージに S を付加して、MS に送る。Unix system V のシステム関数である message queue を使用する。MS(Message Sever)は同一のオペレーターコンソールで動作している。

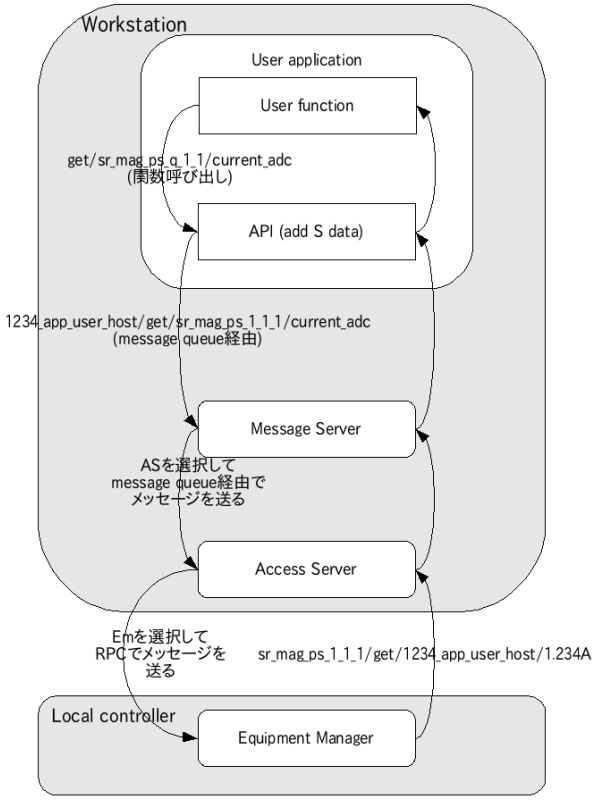

Fig. 12 アプリケーションと EM のメッセージの流れ。

MS はそれを同一のワークステーションで動作している AS(アクセスサーバー)に message queue 経由で転送する。AS は機器グループ毎に複数動作しているが、それは MS によってメッセージの内容から自動的に選択される

AS はメッセージの内容を判断して、それがどのローカルコントローラーあてのメッセージなのかを判断する。メッセージとローカルコントローラーの対応はデータベースに収納されている。AS は ONC-RPC を介してローカルコントローラーでサーバープロセスとして動作中の EM(Equipment manager)を呼出す。

ONC-RPC とはネットワークを介して、異るホストで動作するプロセスに関数を呼び出す機構である。OSI ではセッション層にあたる。プレゼンテーション層として XDR という文法で、関数の il/O をシリアライズしてネットワーク越しに関数(プロシージャ)を操作する。

EM はメッセージを解釈し、抽象化された値、(3.45mA とか 123.4V や ON, OFF)を設定ファイルに書かれている変換表をもとに、機器のための値に変換する。

また設定ファイルには機器のアドレスや対応する機器のデバイスファイルも書かれているので、それらを参照して、機器を制御する。

EM は再び RPC 経由で答えを返す。このサイクルはおよそ 5ms の速さでおこなわれる。

以上のように MADOCA の特徴は

- 開発者はわかりやすい機器の名前を、簡単な動詞で操作する。
- 補語として ON やオフ、単位付きの数値をやりとりするので、機器の具体的なアドレスや、接続されているホスト名などを指定する必要は無い。それらはデータベースに蓄積されている値を問合せることで得る。

MS と AS を分離することには以下のような理由がある。EM との通信は同期型である RPC を使用している。同期型通信は、問い掛けに対して、必ず返事を要求する。もし通信が途絶えた場合を想定すると、同期型の場合、呼び出し側のアプリケーションは返事を待って立ち往生してしまう。非同期型通信は呼び掛けに対する返事は必要としない。従って通信の途絶によってアプリケーションが停止してしまうことはない。message queue という非同期型通信を間に置くことで、同期型通信の弱点を補うことになる。

5.3.ローカルコントローラー内の高速制御

加速器制御の機器では速いフィードバックが必要となる場合がある。例えばクライストロンの真空度をモニターしつつ加速電圧を高めていくような場合、フィードバックは 1 サイクルを 10ms 程度でおこなう必要がある。ネットワーク経由でオペレーターコンソールの

指示を待つ方式では 100ms の時間が必要で、実用的ではない。

そこで 1 つのローカルコントローラー内でフィードバックなどの速い制御をおこなうために EMA (Equipment Manager Agent) [28]という機構を開発した。これは 1 つのフィードバックプロセスを 1 つの仮想的な機器とみなして制御する。

MADOCA では仮想的に存在するのはメッセージとそれに対応する機器というシンプルな構成をとっている。そこでプロセスも仮想的な機器として表現している。

5.4.データー収集系

MADOCA では MS-AS_EM でのメッセージ経由での制御の他に特徴的なのはローカルコントローラーからのデーターたとえば電流値、真空度、機器状態(on off error などのビット情報)CPU 負荷、を定期的(1 秒から 60 秒、現在の最大は 300 秒)に収集してそれをデータベースに収納することである。

データーを取得するためオペレーターコンソール上のアプリケーションはローカルコントローラーに直接メッセージを送信/受信することも可能であるが、数秒前のデーターでも良しとするならばデータベースにデーター取得の問合せをすることも可能である。機器から直接データーを取得しないことによって機器とのネットワークでのトラフィック量の安定化、またアプリケーションのテストが機器が無くとも行える利点もある。

収集され蓄積されたデーターはそのまま、また間引きされ永久保存される。

5.4.1.データー収集方法

データー収集はローカルコントローラー上でデーターを定期的に収集するプロセス(poller)とそのデーターをネットワーク経由で取得するプロセスが独立して動作することによっておこなう。データーの受け渡しは Unix の共有メモリーを使っておこなう。

Poller は EM 関数を使用してそれを共有メモリー上に書き込む動作だけを行う。1 つのローカルコントローラー上では周期に合せて複数のポーラーが動作すること可能でそれぞれに独立した共有メモリーが割り当てられている。共有メモリー内では、一定期間のデーターが保存される。

中央制御室用データー収集クライアント(コレクタークライアント)は定期的にローカルコントローラー上で動作しているコレクターサーバーに ONC-RPC によってポーラーの id を指定してデーターを要求する (1)。

コレクターサーバーは要求があったときに指定の共有メモリー上のデーターを取得し(2)、RPC 経由でコレクタークライアントにデーターをわたす(3)。データーはデータベース書きこまれる(4)。

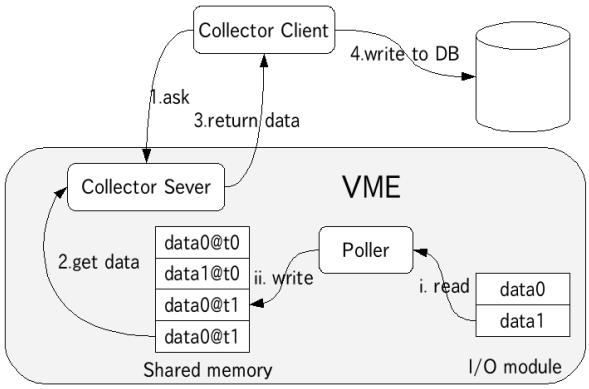

Fig. 13 MADOCA データー収集系

5.5.データベース

MADOCA を説明する際に今までデータベースについて言及してきたが。データベースシステムを広範囲に使用することは MADOCA の大きな特徴であり、MADOCA の名前の元にもなっている。

MADOCA で使用しているデータベースはリレーショナルデータベース管理システムであり、Sybase ASE (Adaptive Server Enterprise) [29]という商用データベースを採用している。これは MADOCA 製作当時最も性能が良かったのが採用の理由である。現在のシェア的には一般的とはいえないが、商用データベースでトップシェアの Oracle[30]と比較して管理の容易さ、サポートの良さで満足できる製品である。ただし現時点で採用するならばオープンソースの MySQL[31]や PostgreSQL[32]がその視野に入るであろう。

5.5.1.リレーショナルデータベース

リレーショナルデータベースが提唱されたのは1970年であるが[33]その普及は80年代半ば以降である。以後オブジェクト指向データベースなどが取って代わるとも言われたが、依然としてデータベースの主流である。

リレーショナルデータベースが広く、長期間にわたって普及している原因として

- •2次元のテーブルを基本とするシンプルな構成
- • 使用するプログラムの構成とは完全に別の実世界のモデルに基づいた構成でデータベースが設計されるのでプログラムの変更にデータベースが影響を受けない。
- •関係代数というしっかりとした数学モデルに基いている。
- •正規化という方法でデーターを1箇所に置く方法を確立した。データーの置き場所が複数あることはデーターの矛盾の元である。

などが挙げられる。

また初期には、概念はともかく、実装上は遅くて使いものにならないといわれたリレーショナルデータベースであるがハードウェア、ソフトウェアの両面の進歩により、大規模高速のデータベースが実現できるようになったことも普及に欠かせない要素である。

5.5.2.MADOCA におけるデータベース[34]

MADOCA においてデータベースは以下の分野で使用される。

1. 電磁石の位置、BPMの位置、較正係数などの静的なパラメーター
2. 機器と信号の関係、機器とローカルコントローラーの関係など関係を示すパラメーター
3. 現在の運転パラメーター、過去の運転パラメーターを記録したデーター
4. コレクタークライアントによって記録される全ての機器の状態を記録したログデータベース
5. コレクタークライアント以外のアプリケーションによって収集されたデーター COD linac BPM, bunch 電流

前者の3つはかなり目的も構成も異ったデータベースではあるが全てパラメーターデータベースとよばれるデータベースに収納されている。実は関係を表現した2以外ではデータベースサーバーは単なる独立した2次元の表としてしか使っていない。これらの目的にデータベースを使うのは2次元の表としての定義が明確で、検索、挿入、消去、アップデートが統一的にかつ容易にできることが理由である。

5.5.2.1.関係を表現するデータベース

対して2では対象、加速器、機器グループ、サブグループ、機器、ホスト、信号、ポーラーコレクターサーバー、コレクタークライアント、ログテーブルなどの関係を記述してる。数万になる信号を矛盾無く収納でき、破綻しないのはリレーショナルデータベースの得意とする分野である。

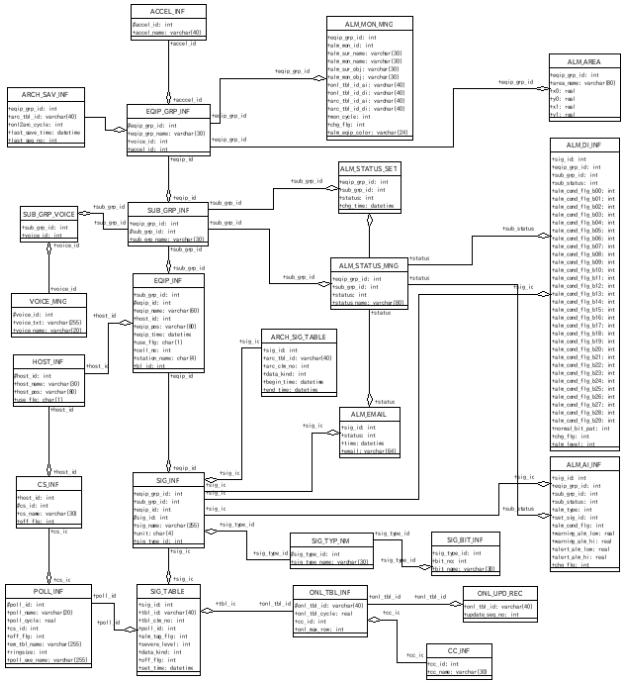

Fig. 14 パラメーターデータベースの一部のテーブルとそれらの関係。一部正規化されていないところもある。

5.5.2.2.ログデータベース

SPring-8では全ての信号のログデーターを永久保存する。そのため3層構造で確実にかつ大量のデーター保存をおこなっている。

ONLINE_DB ではでは取り零しのないように小さなテーブルで 30 分ぶんのデーターを蓄積する。テーブルが大きくなるとどうしてもデーターの挿入の時間が安定しないことがあり、小さいデータベースを使用せざるを得なかった。

ARCHIVE_DB では ONLINE_DB のデーターをコピーして 15 日間蓄積する。

さらに ARCHIVE_DB を間引きして永久保存するデータベースがあ。

どのテーブルにどの期間のどの信号が保存されているのかもパラメーターデータベースを参照して得られる。

5.5.2.3.運転パラメーターデータベース

運転に使用するパラメーターを収納するのために開発された運転パラメーターデータベースがある。これは 2 つの機能がある。

- •運転パラメーターを蓄積し、後日呼び出すことにより運転状態を再現する。
- •ネットワークに接続された異るホスト上のプロセス間での現在の運転パラメーターの共有。

テーブルは 4 種類ある。

- •RUN_MODE
- •RUN_SET
- •RUN_CURR_SET
- •RUN_MODE_HISTORY

RUN_MODE にはその run mode の名前、id,生成日時、最後に使用された日時が記録される。

RUN_SET には機器毎のパラメーターが run mode id とともに記録される。 つまり run_mode_id を指定すれば機器毎の設定値が引き出せる(1)。

異るプロセスで設定値を共有するためには、1 つの run_mode_id で指定された機器ごとの設定値を RUN_CURR_SET にコピーする(2)。

このとき RUN_MODE_HISTORY に自動的に引き出したことが記録され、どの run_mode_id がいつ使用されたか記録に残る。

アプリケーションプログラムがこの RUN_CURR_SET を読んで機器に設定値を与える。 機器の設定値を RUN_CURR_SET の値とは異った値を設定するときには該当の RUN_CURR_SET の値を変更する(3)。 従って別のプロセスは RUN_CURR_SET の値を読むことによりその時の設定値をリアルタイムに知ることができる。

設定された RUN_CURR_SET を後の運転に使用したいときは新たな run_mode_id を生成し、RUN_MODE,RUN_SET に記録できる (4)。

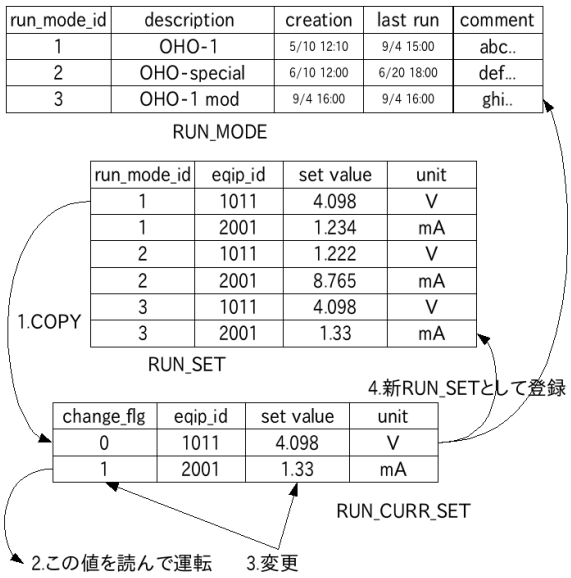

Fig. 15 運転時の RUN_MODE, RUN_SET, RUN_CURR_SET の操作とデーターの流れ、RUN_MODE_HISTORY は省略した。

RUN_SET は階層化され、上位の RUN_SET には下位の run_mode_id を記録してあるので、上位の run_mode_id を指定すれば下位の RUN_CURR_SET に自動的に設定値が収納されるといったことも可能である。

5.6.アラーム

SPring-8 のアラームシステム[35]の機能は機器の不調をオペレーターに知らせるということが全てである。

機器の不調の情報により機器を自動的に停止させる機能、例えば、電磁石の冷却水の水流が停止した場合にその電源を停止させるといったことは機器保護インターロックシステムに任せてあり、アラームシステムは機器対機器の操作はしない。あくまで人間が

関与しなければならない機器の不調があったときに人間に知らせるためののアラームシステムである。しあがってアラームシステムの動作速度も人間に合せて１秒-数秒のレベルで設計されている。

SPring-8 アラームシステムの特徴はそれが全て、データベースのクライアントであるということである。他のフレームワークではアラームの設定値がローカルコントローラー内部に設定されていて、異変があった時にはアラーム信号を出す場合が多いが、SPring-8 のアラームではアラーム監視プロセスはポーラーコレクター系で読まれデータベースに書き込まれた値のみを対象としている。

このことはネットワークトラフィックの安定化に寄与する。加速器運転においてアラームは単独で発生する以外に集中して多量に発生する場合も多い。ローカルコントローラーで発報されたアラームが集中した場合、ネットワークの混雑により、適切な制御ができない、必要な情報が取得できないという事態が予想される。

アラームが大量に発生した時に制御が自由にできないということは致命的事態と言える。これに対して、データベースから値を取得してアラーム値のみをデータベースに書込む方式では、ローカルコントローラーとオペレーターコンソールとのトラフィックは、アラームが多量に発生した場合も変らず、制御を続行できる。

またローカルコントローラーにはアラームのプロセスが不要になり、設定値のロードが不要になる。さらにアラーム設定値を柔軟に変更することが容易になるなどのメリットがある。

運転の状況に応じてアラームの設定値を変更したいときがある。SPring-8 では運転パラメーターデータベースと同じようにこれを１つのモードとして扱う。ただし運転モードとは別でこれをサブグループ状態と呼んでいる。

サブグループとは、同じアラーム状態にできる機器の集合である。例えば、SPring-8 では真空機器をセルと呼ばれる単位で管理している。メンテナンスの際にはセル毎で行うことが基本である。メンテナンスの際には、あるセルのみがメンテナンス状態アラームがかけられ通常状態とは異る閾値が使用される。他のセルは通常アラームが使われる。この場合はセル毎にサブグループとして機器をまとめる。

サブグループ状態は各サブグループにつき１つ以上設定でき、サブグループ状態が変更されればそのサブグループ状態に対応した信号毎の設定値がデータベースからロードされ、監視状態になる。設定値はアナログの他、ビット状態にも定義され、そのサブグループ状態での正常ビットパターン、監視すべきビット、アラームレベルがデータベースから指定される。

アラームが発生するのは、アナログ信号の場合は、上限値を越えた、下限値を下回った、上下の指定範囲を外れた。状態ではビットが指定のものと相違したがある。またデーター収集系が正常に動作しているかのチェック用に、poller がデーター取得時刻をデーターとして収集しているので、それにもアラームをかける。時刻がずれてアラームが発生すればデーター収集系に異常が発生したということである。

アラームが発生した際にはアラーム監視プロセスはそのことをデータベースに記録する。 ONLINE_DB には現在のアラームだけ、ARCHIVE_DB には過去のアラーム履歴が記録される。 アラームが収まったときには ONLINE_DB のアラームは消去され ARCHIVE_DB には、アラームが収まった時刻が記録される。

5.6.1.アラーム表示

以上のことからわかるようにアラームを表示するために必要なのは ONLINE_DB のアラームテーブルを定期的に読み込み、アラームの有無を知るだけでよいことがわかる。この自由度のため多彩なアラーム表示が製作されてきた。

- •グラフィカルな表示
- •音声
- •Web
- •メール、データベース上でアラーム通知が欲しい信号を登録しておけばアラーム発生時にメールが送信される。
- •構内 PHS への音声通知

これらすべてがデータベースのクライアントプロセスとして製作された。

5.7.WEB 表示

機器のログデーターのために Web による表示は大変有用である。次のような利点がある。

- ブラウザーがインストールされていればクライアントのオペレーティングシステム、機種に依存しない。
- ファイアウォールが http プロトコルを通せば、制御ネットワークの外からでも加速器の状態が見える。

である。SPring-8 では S/V/O/C による直感的に理解しやすい信号名によって、信号をサブグループ毎に列挙して、信号名を選択させ、グラフを表示している。

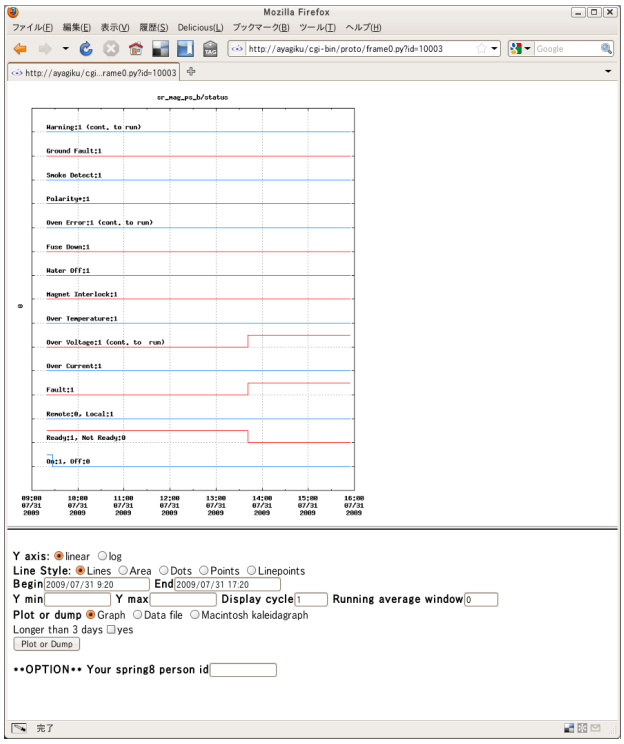

Fig. 16 Web 表示の例。デジタル状態信号。

グラフは

- アナログシングルトラック
- アナログマルチトラック
- アナログ移動平均
- アナログ相関プロット
- デジタル状態信号

などがあり、任意の日時を指定することができる。Web サーバー内で gnuplot[36]により、グラフの画像を生成させ転送する。データーをブラウザーに送り、ブラウザー側の flash や Javascript などで書いたプログラムにグラフを表示させる方法はインタラクティブ性の点で優れるが、データー量が多量の場合には、転送時間が安定せず、採用を見送っている。

Gnuplot は時間軸の表現に優れることが採用の理由である。

5.8.高速同期データー収集 [37]

加速器制御の中で特に多数の BPM (Beam Position Monitor, ビーム位置検出器) のデーターを同時に取得する需要がある。SPring-8 でも入射用線形加速器の 47BPM でのビーム位置を 60Hz でショット毎に取得するためのシステム、また蓄積リングに配置された BPM を 1Hz で取得しフィードバックするためのシステムは、Ethernet を使用した標準モデルでは時間的に折り合いがつかない。複数の VME にまたがった高速のデーター収集が必要である。

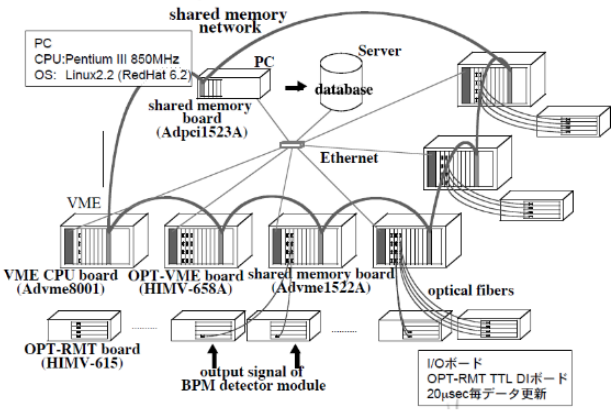

Fig. 17 SPring-8 入射用線形加速器の BPM 読み出し系、下から BPM の信号が光ケーブルで VME に集約されさらに光共有メモリーでデーター収集用 WS にデーターが集められる。データーはデータベースに書かれる。

そのため新規開発したシステムは、BPM 処理回路からは光リンクリモート I/O で VME にいったん信号を集め、さらに VME とデータベース書き込み用ワークステーションとは共有メモリーネットワークで接続した。

共有メモリーボードはメモリーサイズ 4MB、公称の通信速度は 250Mbps、実効 4MBps と GbE と比較し

ても高速とはいえないが、通信制御をハードウェアでおこない、各ノードの遅延時間が 1.5μs とリアルタイム性に優れている。

また EMA を使用し、共有メモリーネットワークの割り込みに対応している。

6.安定運転のために

最初でも書いたとうり、SPring-8 は 24 時間、365 日間の安定運転が要求される加速器である。最後に安定運転のための様々な経験の一端を述べる。

6.1.電源

SPring-8 では中央制御室は大容量の CVCF を使用し、停電、瞬低に備えているがローカルコントローラーは停電、瞬低の対策はローカルに持たなくてはならない。

一般に最も良く使用されるのが UPS(無停電電源 Uninterruptible Power Supply)であるが、蓄電池式の UPS は問題が多い。蓄電池には寿命(3-10 年)があり、交換作業が必要になる。広い敷地内でラック内の UPS の重い蓄電池の交換作業にはコストがかかる。

SPring-8 では UPS を見直すこととした[38]。日本はおそらく世界一電力事情が良い国で、1 分以上の長期の停電は稀で、日本の加速器施設内での停電対策とは実は瞬低 (瞬時電圧低下) 対策である。

瞬低によりローカルコントローラーがどう停止してしまうかは、その種類によって大幅に違う。また、ベンダーによってもポリシーが異る。ある PLC ベンダーは瞬低が起きたときには、PLC を停止させる方針を取っている。瞬低を見過すと、制御対象の他の機器に思わぬ悪影響があるかも知れないので、一旦停止させ、人間がチェックすべきだという考えに基づいている。

SPring-8 では瞬低シミュレータを使用し、各ローカルコントローラーの瞬低に対する耐性を確認し、かつ瞬低の統計的な記録を参考に 5 年に一度の瞬低による停止を許容量とした。

その結果、蓄電池式の UPS ではなく大容量コンデンサ式の瞬時電圧低下保護装置を採用することになった。この方式は約 1 秒の瞬低に対して機器を保護するもので、寿命も長く、メンテナンスコストが UPS と比較しても格段に良い。この瞬時電圧低下保護装置を採用することでトラブルも UPS に比べて大幅に減らすことの成功した。

6.2.ハードウェアの信頼性向上

機器の故障に備えるために最も有効な方法は機器を二重化、三重化することである。一方が故障してももう一方で運転を継続し、その間に故障した機器の修理をしてしまうものである。

機械が停止してしまうのはソフトウェア的な故障とハードウェア的な故障があるが二重化はハードウェア故障を主に想定している。

SPring-8 では二重化は

- サーバー計算機
- 基幹ネットワーク

でおこなっている。ローカルコントローラーでは VME のファンや電源の二重化をおこなっている。

二重化で重要なのは single point of failure を作らないことである。例えば二重化していたハードウェアが同じコンセントから電源をとっていれば、それで二重化は意味をなさなくなる。

サーバー二重化はディスク共有型高可用性クラスターでおこなわれることが多い[6] 複数の cpu(ノード)がディスク(storage area network)を共有し、指定されたサービスを実行する。相互のノードはハートビートと呼ばれる信号をネットワーク経由でやりとりして相互の生死を常に確認する。ハートビートが途絶える理由は 2 つあり

1. 1 ノードが故障してサービスが維持できなくなった。
2. ネットワーク機器に異常があった。

ハートビートが途絶えた際には、それがネットワーク異常ではないことを確認するため、ノードは共有ディスクのある領域を確保しようとする。その領域を確保できたノードが、サービスを代行するノードとして、クラスター用のネットワークアドレスを取得し、かつサ

[6] これは密結合型クラスターと呼ばれる。その他にシェアードナッシングとよばれる、共有するものが無い疎結合型クラスターも存在する。

ービスを動作させる。この動作を fail over と呼ぶ。またノードを修理した後、サービスを元の戻す動作を fail back と呼ぶ。

SPring-8 ではこれをファイルサーバーやデータベースサーバーに使用し信頼性をあげてきた。

ただ高可用性クラスターをデータベースサーバーに使用した際の問題も明らかになった。それは fail overl, fail back の際にデータベース管理システムの行うデータベースのチェックに約 20 分の時間がかかり、その間データベースを使用できないことが明かになった。またクラスター管理ソフトウェアも複雑でわかりにくかった。さらにデータベース管理システムのライセンスもスタンバイ用のものを購入、保守する必要がある。

そこでデータベースサーバーについては fault torelant 型サーバーを導入しつつある。これは高可用性クラスターをハードウェアレベルで実現したような製品で、ソフトウェア的には 1 個のノードに見えるが中には全く同一のノードが 2 つある。このためデータベースライセンスは 1 つですみ、またデータベースサーバーのチェックも必要でないため、フェイルオーバーフェイルバック作業にも時間がかからないといった利点がある。

以前はフォルトトレラントのコンピューターは非常に高価なものであったが、近年では比較的安価な製品も登場してきたもで SPring-8 ではこれを導入している。

6.3.ソフトウェアの信頼性向上

24 時間 365 日間の安定運転を目指す際にはソフトウェアの信頼性向上も必須である。制御用ソフトウェアでは起動当時は正常動作していたものの、長期間の運転でトラブルを起す例がある。デバッグする際にも何日も待たねばならない場合もあり、厄介である。

このようなトラブルは経験上メモリーリークによるものが多い。すなわち、プログラム内でメモリーを占有し、解放を怠ったため、メモリー使用量が増加し、最後にはシステムのメモリーを食い潰すという現象である。このバグは目視でソースから発見するのが難しい。Purify[39]や valgrind[40]などのツールは実行時のメモリーリークを検出するツールである。Purify はこの他にアレイのリミット越えも検出できる。これらを使用すれば、飛躍的に問題箇所の発見が楽になる。C で長時間動作させるプログラムの開発には必須といえる。

6.4.セキュリティ

加速器制御ではインターネットと同一の Ethernet TCP(UDP)/IP を使用し、使用するコンピューターも Unix マシンや Windows マシンである。それらを加速器制御ネットワークも外部ネットワークと接続することによる利点もあるが同時にセキュリティ上の問題も発生する。

研究所内外からのネットワーク攻撃にさらされることを防ぐため外部とのネットワークの間にはファイアウォールを設置し、不要なアクセスを遮断することは言うまでもない。一方では完全に物理的に遮断することもセキュリティ上好ましくない。たとえネットワークを物理的に完全独立させたとしても、ウイルスに汚染されたパソコンが誤って接続されると、遮断されたネットワーク全体に汚染が広がる。

外部接続をして、オペレーティングシステムにセキュリティパッチを当て、パソコンのアンチウイルスソフトウェアを更新することは例えファイアウォールの中にあるコンピューターでも重要なことである。

またエキスパートがファイアウォール外からコンピューターを操作することも時として必要になる。このために VPN(virtual private network) 等の仕組みで、外部からの接続に対応することが必要である。

ローカルバスとして使用している Ethernet 機器のセキュリティも見落しがちな点である。これらの機器はネットワーク回りに割ける資源も少なく、過剰なブロードキャストトラフィックに耐えられない例が見受けられる [41]。

もしウイルスに汚染されたパソコンがネットワークに接続され、過剰なブロードキャストトラフィックを発生させた場合、これらの機器が停止する事態も発生する。このような機器を新規導入する際にテストすることも必要である。

さらに組み込み Windows にセキュリティパッチを当てるか否かは難しい問題である。セキュリティパッチが、本体のアプリケーションに悪影響を与える可能性

は無いとは言えない。これは個々のベンダーの指示に従うしかないが、その指示が間に合わないケースも考えうる。 制御用ネットワークに新たにパソコンを接続する際はウイルスチェックをしっかり行うといった対策の他に、SPring-8 ではウイルスに感染して、不要なブロードキャストを発生させるマシンをネットワークから隔離させるアプライアンスを採用している[42]。

7.最後に

SPring-8 のサイト内では X-FEL (X 線自由電子レーザー) の建設が進行中である。またここ数年の間でも世界でも新しい加速器が新たに稼動を開始、また建設中である。

これらの加速器は方式が異るとはいえ、いずれも以前の加速器と比較してはるかに高性能、高安定である。これらの加速器の性能向上には加速器制御技術の進歩とそれを支える IT の進歩が大いに寄与していることは間違いない。

加速器制御の標準モデルが提唱され 20 年近くたった。その間には革命的な変化は無かったが、加速器制御は着実に進化をとげた。

加速器の寿命は長く、その間の性能向上の要求は絶えることはない。加速器制御はそれらの要求に柔軟に対処し、かつ新しい技術を取り入れることも可能なシステムでなければならない。

8.謝辞

OHO09 で発表の機会を与えていただいた OHO09 校長の古屋貴章様、KEK 山本昇様に深く感謝いたします。

このテキストを書くにあたり SPring-8 の田中良太郎、増田剛正、石井美保、大端通、佐治超爾、広野等子、KEK の山本昇の皆様にはコメントと図版の提供をいただきました。記して感謝いたします。

9.参考文献

[1]Kuiper, B,.″Issues In Accelerator Controls″, **Proc. ICALEPCS 91**, Tsukuba, lJapan, 1991
[2]http://sun.com/solaris/.
[3]http://www.picmg.org/v2internal/compactpci.htm.
[4]Ishii,M. et.al., ″DEVELOPMENT OF A PC/104-PLUS BASED CPU MODULE WITH POWER OVER ETHERNET CAPABILITY″,**Proceedings of the 5th Annual Meeting of Particle Accelerator Society of Japan**, Higashihiroshima, Japan,2008.
[5]JIS B 3511, FA コントロールネットワーク標準 ―プロトコル仕様, Feb. 2004.
[6]加藤 他, ″FL-net 上に構築された PLC ベースの加速器制御システム″,第 28 回リニアック技術研究会, 東海,日本, 2003
[7]Hasegawa,T., et.al., ″CONTROL SYSTEM FOR ELECTRON GUN MODULATOR WITH FL-NET″, **Proceedings of the 2nd Annual Meeting of Particle Accelerator Society of Japan**,Tosu, Japan,2005.
[8]Ishii,M. et.al., ″A Software Framework to Control a Network-connected Equipment as a Pseudo Device ″, **ICALEPCS 2003**, Gyonju, Korea, 2003.
[9]http://www.windriver.com/.
[10]http://www.fujidatasystem.com/X-Mate/X-Mate.html.
[11]http://www.aps.anl.gov/epics/.
[12]http://www.tango-controls.org/.
[13]http://www.omg.org/.
[14]http://omniorb.sourceforge.net/.
[15]http://www.ni.com/labview/.
[16]Tanigaki,M., et.al, ″DEVELOPMENT AND CURRENT STATUS OF THE CONTROL SYSTEM FOR 150 MeV FFAG ACCELERATOR COMPLEX″, **Proceedings of PCaPAC 08**, Ljubljana, Slovenia,2008.
[17]http://tesla.desy.de/doocs/doocs.html.
[18]http://gcc.gnu.org/.
[19]Fujihara, R., et. al., ″Construction of Linux Computer System for Acceleration

Equipment″,**Proceedings of the 5th Annual Meeting of Particle Accelerator Society of Japan**, Higashihiroshima, Japan, 2008
[20]http://www.python.org/.
[21]http://www.wxpython.org.
[22]http://www.riverbankcomputing.co.uk/software/pyqt/intro.
[23]http://www.nongnu.org/cvs/.
[24]http://subversion.tigris.org/.
[25]http://git-scm.com/.
[26]http://mercurial.selenic.com/wiki/.
[27]Tanaka, R.,et.al,″Control System of the SPring-8 Storage Ring.″ **Proc of ICALEPCS'95,1995**, Chicago, USA, 1995.
[28]Taketani, A., et.al,″Medium-Speed Feedback Software Based on the Existing Control System″, **Proceedings on ICALEPCS 97**, Beijing, China, 1997.
[29]http://www.sybase.com/.
[30]http://www.oracle.com/.
[31]http://www.mysql.com/.
[32]http://www.postgresql.org/.
[33]Codd, E.F. . ″A Relational Model of Data for Large Shared Data Banks″.: **Communications of the ACM 13 (6 377–387)**. 1970.
[34]Yamashita, A., et.al,″The database system for the SPring-8 storage ring control.″**Proc of ICALEPCS'97**, Beijing, China, p.427, 1997.
[35]Yamashita, A., et.al.,″The Alarm System for the SPring-8 Storage Ring.″ **Proc of ICALEPCS'97**, Beijing, China, p.585, 1997.
[36]http://www.gnuplot.info/.
[37]Masuda, T., et.al,″Event-synchronized data acquisition system for beam position monitors of SPring-8 linac.″.**Nucl. Instrum. Methods A 543** p.415,2005.
[38] Ishizawa, Y., et.al., ″Installation of Instantaneous Voltage-Drop Protector without UPS″,Proceedings of the 2nd Annual Meeting of Particle Accelerator Society of Japan, Tsukuba, Japan, 2005.
[39]http://www-01.ibm.com/software/awdtools/purifyplus/.
[40]http://valgrind.org/.
[31]Sugimoto, T., et.al.,″TCP/IP VULNERABILITIES OF EMBEDDED-SYSTEM IMPLEMENTATIONS″, **Proceedings of PCaPAC08**, Ljubljana, Slovenia, 2008.
[42]Ishii, M., et.al.,″CONSTRUCTION AND MANAGEMENT OF A SECURE NETWORK IN SPRING-8″, **Proceedings of ICALEPCS05**, Geneva, Switzerland, 2005.