「加速器」Vol. 2, No. 2, 2005（162–169）

解　説

SPring-8 独自開発の制御システム—MADOCA

田中　良太郎*

MADOCA—Accelerator and Beamline Control Framework Developed by SPring-8

Ryotaro TANAKA*

1. はじめに

1997年3月，兵庫県は播磨の地で，大型放射光施設SPring-8の8 GeV蓄積リングはコミッショニングを迎えていた．蓄積リング制御システムの骨格はそれから遡ること3年，1994年に産声を上げようとしていた．

加速器の制御システムはいかにあるべきか，その思想と実装をゼロから検討するために，加速器文化，高エネルギー文化，企業文化を交えて，大いに議論を戦わせた．その結果，SPring-8の独自開発として，加速器とビームラインの制御フレームワーク「Message And Database Oriented Control Architecture（MADOCA）＝メッセージとデータベース指向制御様式」は生まれた．そして10年たった今，MADOCAはSPring-8の線型加速器，ブースターシンクロトロン，蓄積リングとビームラインの制御を担い，兵庫県立大学の放射光加速器NewSUBARU，広島大学の放射光加速器HiSORにも移植され，順調に加速器とビームラインの運転を支えている．また，現在SPring-8に建設中の，理化学研究所のSCSS試験加速器（Cバンド XFEL試験加速器）にも導入作業を行っている．

この解説では，日の丸印の制御フレームワーク（＝制御の枠組みと実装のための構成要素群）MADOCAが，どのような思想の元に設計され，どのような形で構成されているかを解説してみたい．加速器の制御システムがどのようなフレームワークでできているかに触れる機会はあまり無い．また，制御の話は分かりにくいとよく言われるので，専門用語は極力説明し，3文字略語の羅列とならない読み物としたので，この機会に読んでいただければと思う．

解説の始めの方では，設計思想，盛り込まれた機能などを説明し，次にそれらをどのような技術でどう実装したかを述べる．また，MADOCAは「メッセージとデータベース指向」の制御機構なので，メッセージ通信，データベース機能，そしてアーキテクチャに特色があり，これらを初めに説明している．

MADOCAはフレームワークなので，もはや「MADOCA＝SPring-8制御システム」ではない．少し掴みにくくなるかも知れないが，ここではMADOCAとSPring-8制御系を分けた書き方にしている．今となっては，「SPring-8制御系はMADOCAを使って実装した1つの構築例」と言うことができる．とはいえ，実際にどのようなハードウェアを用いて，システムを構築するかに言及しながら説明すると分かりやすいので，応用例として，SPring-8制御システムを適宜引用しながら解説してみたい．

1.1 誕生前夜

94年当時，加速器の制御システムには様々な設計があり，実際，世界の加速器研究所には制御システムは大小取り混ぜていろんな実装があった．その中にあって，これだけの加速器技術と文化，そして実績を持つ日本から，独自の設計で加速器制御システムを誕生させたいという思いがあった．また，幸運なことにそれが可能となるような役者達が，当時，蓄積リングを担当した理化学研究所に揃っていた．20世紀最後を締めくくる大型放射光施設の建設気運に乗って，新たな制御フレームワークを作る気概に満ちていた．最初の一歩は，既存の加速器制御システムを参考にはしても，最適なものを目指して，そもそも「加速器制御に求められるものは一体何か」を議論するところから出発した．

マクロな加速器も，実は多数の機器から構成されて

* ㈶高輝度光科学研究センター　SPring-8
（E-mail: tanakar@spring8.or.jp）

いる．加速器制御はこれらの機器を制御して，ビーム制御することがミッションと言える．制御すべき機器は，電磁石電源，パルスモーター，バルブ，ゲージなど多岐にわたる．制御系は，これらの機器に設定値を送る（指令の送信＝put），機器からステイタスを読み出す（データ取得＝get）などの，データ通信を介して機器制御を行うことで加速器を制御している．SPring-8 では制御点数は全部で約 6 万点に上り，これらに対して頻繁に get/put を行っている．これは人間業ではできないので，計算機などの電子情報技術が要素技術として必要になる．

さて，制御システムに求められる要件定義のために，各機器の担当者にどのような運転をするのか，聞き取りを行うことから始めた．機器の単体テスト，組み合わせテスト，ローカル制御，リモート制御，通常運転とメンテナンス時に何を望むか等も聞き取りを行った．思考実験で機器操作のイメージを作り上げていった．このような思索から，制御系が具えておくべき応答性，安定性，柔軟性，拡張性，独立性，冗長性などの制御要件が見えてくる．そして，制御系はこれらを具備して，誰もが容易に構築でき，使い易いものでなければならない．特に，SPring-8 加速器全体として見たときには，62 本のビームラインで放射光利用実験を行うユーザーを思い浮かべながら，耐障害性（実際には高可用性）を有したアーキテクチャの実現に思いをはせながら設計した．

そういう意味では，優れた安定性，実績，各種のアプリケーションとツールの豊富さから，UNIX オペレーティングシステムを使って実装することを前提に設計を考えていた．

2. アーキテクチャ

一般に，加速器施設は装置の複合体であり，その施設は大きく，広い．必然的に加速器を構成する機器は施設内に広がって設置することになる．加速器全体を中央制御室一カ所から制御するためには，どうしても分散制御系の形態が必要になった．これは，制御のレイヤー構造で言えば，被制御機器とインターフェースする「機器制御層」，制御指令やデータを伝送する「通信層」，そして制御室に詰めるオペレータが操作に使う「上位層」の 3 階層構造になる．これをソフトウェア的に表現すると，「デバイスインターフェイス」「ミドルウェア」「マン・マシン・インターフェイス」の 3 階層になる．この構成は今日，制御の標準モデルと呼ばれており，これをベースにして前述の制御要件を設計の中に織り込んでいくことになる．

色々と検討した結果，「クライアント・サーバー型分散制御系」にたどり着いた．これには当時，Device server model を採用していた，フランスのグルノーブルにある放射光施設 ESRF の制御系が参考になった[1]．また，全体設計をリファインするには，分散オペレーティングシステムに関する A. S. タネンバウムの著書[2]，クライアント・サーバーモデルに関するダン・ハーキーらの著書[3]も参考になった．クライアント・サーバー型分散制御系は，6 つの制御要件の中で柔軟性，拡張性，冗長性を実装しやすく，ビーム性能向上，モニター機器の追加など，日々成長していく加速器の制御アーキテクチャとしてはぴったり合っていた．

制御系の話，中でもアーキテクチャはどうも分かりにくいという方には，私は日本伝統の漁法である「鵜飼い」を例に説明している（日刊工業新聞に連載の解説記事を参照のこと[4]）．「SPring-8 の制御系は鵜飼い型ですよ」と説明している．外国人には通じないが．中央制御室のオペレータは「鵜匠」で，通信レイヤーは「繋がれたヒモ」，そして機器制御層は「鵜」で，狙う魚は「信号」に例えている．鵜匠がクライアントであり，鵜がサーバーになる．大型加速器では，クライアントが数人でサーバーが数百羽になりうる．実際の構成例では，数台の上位系計算機上でクライアント・ソフトウェアが走り，数百台の機器制御層計算機上にあるサーバー・ソフトウェアと通信する構成となる．MADOCA では，このクライアントとサーバーの組み合わせで任意に冗長構成をとることができる．

冗長構成は制御系には必要な要素で，局所的障害が全系に及ばない，また 1 系統が不具合でも別系統で制御性を維持できるというのは安定性に寄与する．後に分かるように，MADOCA は設計の段階でこれらのことを考慮されている．**図 1** に概念的な例を示す．図

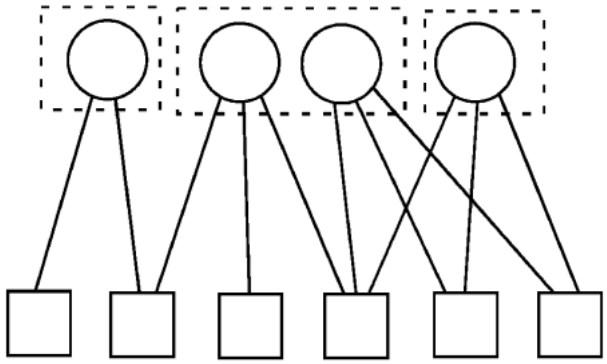

図 1　制御プロセスの構成例．○は上位系プロセス．□は機器制御系プロセス．右の○ 2 つは冗長構成．

の○と□はある制御機能を示し，○印は上位層で，□は機器制御層になる．ラインは通信層を示す．実装時のハードウェアの境界として，複数の○を1台の上位層計算機に載せても良いし，1つの○のみ載せても良い．同様に複数の□を1台の機器制御層計算機に載せても良いし，1つの□を載せても良い．SPring-8は点線で囲まれた範囲が1台の計算機に対応するような構成を取っている（**図1**の例では3台になる）．図中，複数のラインが1つの□に繋がっていること，右側二つの○が同一計算機上にないことはポイント．拡張性が必要になった場合は，この○と□の組み合わせを適宜，並列的に追加するだけでよい．

2.1　加速器を操る人々

制御担当者からすれば，ビームオプティクスを担当する人，そして機器を担当する人たちは，制御系を利用するユーザーに見える．制御系はこれらの人たちにとって使い易いものであり，かつ，彼らの要求を実現できなければならない．

聞き取り調査によると，そのために満たすべき条件として，1)信号ケーブルがどのように接続されているかを必ずしも意識させないこと（必要時には見えること），2)ビームの状態，機器の状態を記録でき，いつでも見られ，加速器の運転状態を再現できること（蓄積リングは特に）3)機器のデータ収集が簡単にできること，4)制御するためのプログラムが簡単に書けることが浮かび上がった．実際のところ，MADOCAではこれらが実現できており，1)は「機器の抽象化（device abstraction）」，2)はデータベースの導入とデータアクセスの標準化，標準的な表示系の提供，3)は機器データ収集系のテンプレート化，4)はソフトウェア部品のコンポーネント化とビルダーによるプログラム作成という形で取り入れられている[5]．以下に，これらを少し詳しく説明したい．

2.2　機器の抽象化

ビーム制御プログラムを作る場合，信号ケーブルがどのシャーシの，どのモジュールの，どのチャンネルにつながっているか，物理的構成を明示的に知らなくてもコードが書けるように「カプセル化」の考え方を導入した．もしも，機器の構成をそのまま制御プログラムに書き込んでいたら，モジュールの構成を変えるたびに，関連する全てのコードを書き直さないといけない．また，スロットやチャンネルを意識した操作では，それらを見ただけでは，どの機器の何という信号かイメージが連想し辛い．

そこで，機器の表記は人に理解し易くなることを目指して，「構文的（syntactic）」ではなく，「意味論的（semantic）」になるように，統一した思想で表すことにした．例えば，人を呼ぶのに「3番の人」などよりは「田中さん」の方が，イメージがはっきりする．「第5部署の3番の人」よりは「SPring-8の田中さん」の方がさらにイメージがはっきりする．機器名もそうである．

機器の表記は，オブジェクト指向の考え方を取り入れて階層化して，「sr_mag_ps_st_h_1_2」というように表すことにした．これはstorage ring, magnet, power supply, steering, horizontal, cell-1, number-1の機器という意味で，機器オブジェクトと呼び，この例は電磁石電源を示す．オブジェクトは「アンダースコア“_”」で区切られており，例えば，任意の機器階層（sr_mag_ps）でまとめて一気に操作することもできる．機器オブジェクトは，単なる共通名称ではなく，制御プログラム，データベース，Webでの表記など制御系のあらゆる局面で出てくる重要なものである．全系でユニークに命名される．マン・マシン・インターフェイスのプログラムは，この機器オブジェクトを使って作成され，スロット番号，チャンネル番号などは一切出てこない．これらはデータベースで一元管理されている．

制御指令も，どんな指令を送ったのか記録がとれ，かつ，命令が一見して人に分かるように，意味論的な表記方法を考えた．その結果,「SVOC方式」と呼ぶ，機器のオブジェクト化とマッチした制御指令表記に辿り着いた．これは，日本の英語教育の初期に習うS(subject), V (verb), O (object), C (complement) になぞらえている．「S」は計算機上の制御プログラムに自動的に付けられる識別子で，プロセス番号，プログラム名，アカウント名，計算機名を“_”で結んだ形になっており，計算機上に複数のプロセスがあってもユニークになるように付けられている．例えば，「123_magcontrol_operator_console4」.「V」は動作を表し，get, put, setなどがある．「O」は前述の機器オブジェクト名,「C」は機器に対する制御属性を表し，status, current_adc, voltageなどを表す．指令の完全形は“/”で区切られた「S/V/O/C」の形式（最大256文字）になっている．

2.3　メッセージ通信

S/V/O/Cが出てきたところで，制御指令をどのように送っているのか述べる．オペレータは通常目の前にある制御用端末（コンソール計算機）のマウスあるいはキーボードから設定値などを指定して，遠隔の機器に指令を送る．通信レイヤーでいうと「ミドルウェア」がこれを担う．MADOCAのミドルウェアはちょ

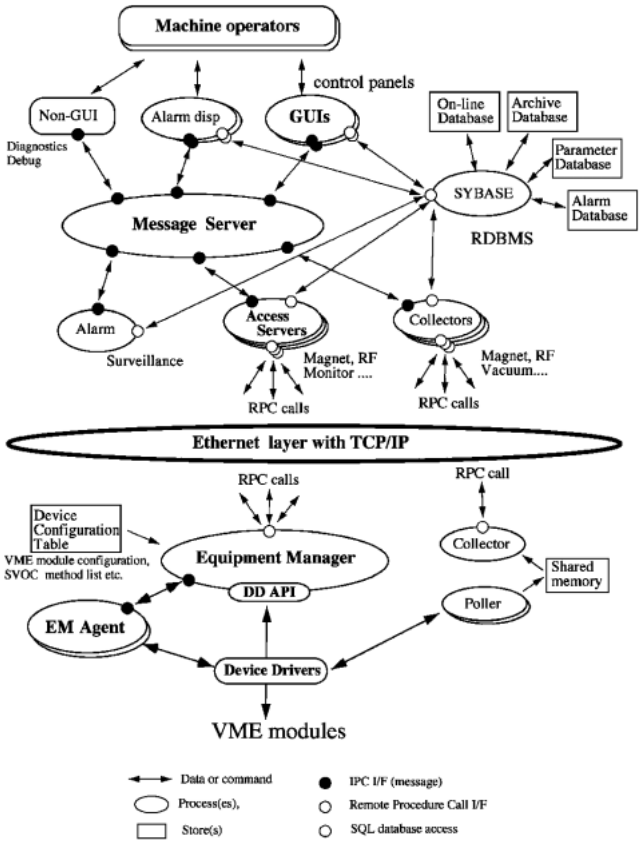

図 2　MADOCA ソフトウェア構成図

うど，郵便システムと電話通信システムの機構に似ている．

図 2 に MADOCA のソフトウェア構成を示す．上半分はオペレータコンソールとサーバーなどの上位系，下半分は機器制御系のソフトウェア構造を示している．まず，上半分(上位層)のみ見ていただきたい．これらの○はコンソール計算機上で動いている MADOCA のプロセス（プログラム）を示す．ここには計算機の境界を書いていないが，全てのプロセスを 1 台の計算機で動かすこともできるし，予算に合わせて数台の計算機に分散しても良い(普通は数台使う)．GUI（Graphical User Interface）は，オペレータの操作するソフト的なパネルを示す．操作用ボタンや設定値入力領域などがある．プロセスの役割を例えで説明すると，図中，Message server（MS）は郵便局で，Access server（AS）は電話局と見なすと分かりやすい．SVOC による制御指令＝メッセージとして，メッセージを手紙と考えてみる．S＝差出人，O＝宛先，VC＝通信文と考えると理解しやすいだろう．

GUI の中では，SVOC 指令を作り，指令を送り出すときは C 言語で書かれた数個の通信ライブラリー関数のうち 1 個を呼び出す．ちょうど手紙をポストに投函するようなもの．MS は送られてきた SVOC の O を見て，メッセージの届け先を判別し，機器制御指令の場合は該当する AS に送る．どの手紙を何処に届けるか（住所録）は Access Control List（ACL）と呼ばれ，ファイル上で指定される．AS は SVOC を受け取ると，データベース上の機器信号情報から，O に指定された機器がどの機器制御装置（VME など）にあるかを判別し，要求されているアクション V に従って 1：1 通信あるいは，一斉同時通信などの遠隔通信で指令を送る．

メカニズムの話をすると，MS は，「キュー」を用いたメッセージ機能で実装されている．キューは First-In-First-Out（FIFO）で入力順に処理される．よって，GUI から送った指令の順番に，結果が帰ってくる．GUI から見れば処理は非同期型になる．(手紙を送って，その返事をもらうために，いつ来るかと自宅のポストを見るようなもの)．AS はリモートプロシージャコール（ONC/RPC）で遠隔通信する．これは遠隔にいる VME などの機器制御装置で動いている Equipment Manager（EM）プロセスと，コネクションを張ったまま通信するので，電話で対話するイメージに似ている．これは同期型通信になっている．

GUI の作りは，指令を送って別の処理を行い後で機器からの返事をもらう，あるいは，指令を送って返事が来るまで待つ，などプログラムは自由に組める．キューの深さはメッセージ 300 個弱で，1 つ送っては 1 つ返事をもらう，あるいは，どっと送って一気に返事をもらうなど，使い方は色々できる．

さて，メッセージの応答性だが，設計当初には，GUI から機器に指令を送ったときの制御スループットは，往復で数十 Hz は超えたいと思っていたが，今実測すると 400 Hz は出ている（上位計算機から VME のアナログデータを読む）．数点の信号を使ったフィードバックなら上位計算機からでも数十 Hz はいけそうである．CPU クロックの高速化が効いている．

MADOCA の利用者（ユーザー）は，自身が使用する加速器運転パネルなどの GUI を作成する必要はあるが，MS，AS はフレームワークで提供されるので作成しなくて良い．

2.4　データベースの導入

MADOCA の特徴に一つに，リレーショナルデータベースを使った強力なデータベース機能がある[6]．データベース機能は設計の当初から必要性が認識されていたが，実際に作って，使ってみて，本当に強力なものだと実感させられる．実際に保存されているデー

タとしては，運転パラメータ，機器のデータ（設定値，ステイタスなど），ビーム位置モニター（BPM）のデータ，機器の較正データ，機器アラームの記録など各種様々ある．どのようなデータを載せるかはシステム設計者が自由に決定できる．

加速器制御では，機器データの保存が大変有用な事が分かった．例えば，電磁石電源では，設定値（DAC），読み取り値（ADC），アラーム（電流異常，電圧異常，温度異常など）を，真空系では真空ゲージの値，バルブの開閉状態などが記録できる．記録時刻は Network Time Protocol（NTP）で全系が合っているので，機器がおかしくなったときに，いつからどうなったかが一目で分かる．また，色んな信号との相関を見ることで，悪い影響を及ぼすエラーソースも容易に発見できる（ある操作をすると何かがおかしくなる，など）．

SPring-8 の場合，早く変動するものは 1 秒おきに，遅いものものは 30～60 秒おきで，定周期で全データを絶えず収集し，収集時間と共にデータベースに蓄えている（方式は後述）．約 6 万点の制御点数についてデータを収集し，記録しており，1997 年の蓄積リングのコミッショニング以来，保存し続けている．

放射光利用運転時は，約 300 台の BPM のデータからは，30 秒おきにビーム位置を読み出し，データベースに記録している．これらのデータを解析して，ビーム変動から，地球潮汐の影響，地震波の検出ができている．ビーム制御，軌道安定化のためには，BPM と電磁石，RF 系のデータが整然と蓄積されており，いつ何時のデータでも簡単に呼び出せるのは大変便利だと言われる．

データベースには 4 つの区分があり，「パラメータデータベース」「オンラインデータベース」「アーカイブデータベース」「アラームデータベース」がある．SPring-8 ではデータベーステーブル数が全部で 800 を超える大規模な物となっている．先ほど述べた機器データはまず，オンラインデータベースに保存され，一定時間経過すると，物によっては間引かれて，アーカイブデータベースに移動し，恒久保存される．ビーム電流値は 1 秒間隔のデータを間引かずにアーカイブしている．保存時間，間引き間隔などは自由に設定できるようになっている．加速器運転時はアラーム監視プログラムが，絶えずオンラインデータベース上の機器データをモニターしており，機器に異常が発生した場合はその記録をアラームデータベースに恒久的に残すようになっている[7]．運転パラメータは RUN-SET と呼ぶパラメータデータベースに，機器の設定値と共に記録しているので，97 年のコミッショニング以来の，いつ何時の運転状態でも再現することができる．

保存されたデータは解析して初めて御利益がある．そのために，データの構造など一切知らなくても望みのデータが読み出せるように，C 言語で書かれた多数の関数群が用意されている．アクセスするときは前述の機器オブジェクト名を指定するので，直接的なイメージで機器のデータが入手できる．もちろん，データ検索が有効にできるように，各データには時間などの検索用インデックスが貼り付けられている．

SPring-8 では毎年数十ギガバイトのデータが蓄積されているが，これをもし，一昔前のようにアスキーファイルにしてディスクに書いていたら，果たして検索などどうなるだろうか？ちなみに，97 年からのアーカイブデータベースの蓄積量は 490 GB になっている．（注：これには機器データの他に，時間，インデックスなどのデータ検索用の付加情報も含められている）．

図 3 にあるように，MADOCA には Web ブラウザーによるデータ閲覧も用意されている[8]．これは良くできた物で，研究者は居室の端末を操作して，Explorer, Safari などのブラウザーであらゆるデータを見ることができる．ブラウザー表示にはグラフ化の機能があり，データの異常変動などがあれば一目瞭然で発見できる．もちろん生データをファイルに保存する

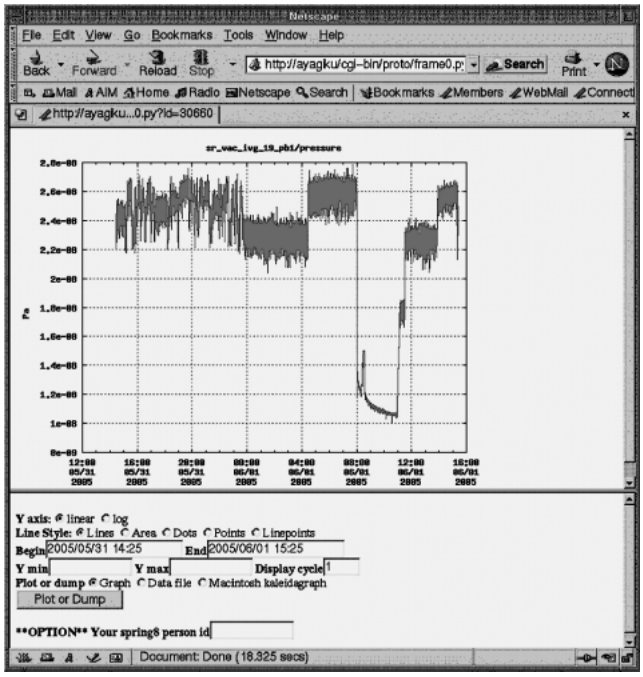

図 3　Web ブラウザーによる機器データの表示

こともできる．Web ブラウザーでは，機器データの他にも，パラメータデータ，アラームデータなども見ることができる．

これまで述べてきたデータベース機能は，全て MADOCA に含まれる物なので，MADOCA を導入したユーザーはこれらを利用することができる．

MADOCA は現在，データベース管理システム（Relational Database Management System, RDBMS）として，Sybase を用いている．Sybase には HP-UX 版，Linux 版などがある．予算規模に合わせて選ぶことができるし，クラスター計算機構成が構築できる場合は，高可用性データベースシステムを組むこともできる．これは 2 台の計算機を，相互監視しながら連携運転させ，主サーバーに故障ある時は待機サーバーに自動的に処理を引き継ぎ，ダウンタイムを無くすことを目的としている．勿論，単体システムとして運用することもできる．

Sybase は MADOCA の設計当初に選ばれた．これは当時あった Oracle などの RDBMS の処理性能を実測・比較して Sybase が優れていたために選んだ．MADOCA のデータベースとしては Sybase 以外の実装は現在ない．現在市場にあるオープン系データベースに移植するにはライブラリーの大幅な書き換えが必要になるためである．

3. 機器制御層

機器制御系で MADOCA の特徴的な構造として，制御用のプロセスが 1 台のコントローラーの上で複数起動するマルチタスク型になっている．よって，機器制御系コントローラ（VME）は，アーキテクチャと実装を含めて，優先度制御，排他制御などが確実に行えるシステムでなければならない．SPring-8 ではコントローラーの OS に Solaris を使っている．

3.1 Equipment Manager

さて，機器制御の中心に位置するのは MADOCA では Equipment Manager（EM）プロセスになる[9]．EM は C 言語で書かれたプログラムで，上位系から送られてきた抽象化された命令（SVOC）を受け取り，これを具体的なデバイスと結びつける．このために EM はデバイス構成表（Device Configuration Table）を持っている．この構成表には put/get 命令が来たときに，どんな処理関数を呼び出すかの対応関係が，オブジェクト毎に規定されている．構成表はアスキー形式のファイルであり，データベースに登録されている機器データ，信号情報をもとに生成される．実際にアナログ入力ボードなどにアクセスする場合は，デバイスファイル，チャンネル番号などのパラメータを指定して，デバイスドライバーを呼び出す．ドライバーは C 言語で書かれている．

EM の本体の中で，デバイスドライバーを呼び出す部分をうまく「コンポーネント化」しておけば，制御点数が増えたときなどは，デバイス構成表のパラメータを少し変更するのみで，信号を追加することができる．さらには，コンポーネントそのものを「ライブラリー化」しておけば，再利用可能で大変便利な物になる．MADOCA では実際にそのように作られている．便利な関数群は既にあり，EM を作成するユーザーに提供される．

3.2 機器データ収集系

機器データ収集系も標準的に提供されることは既に述べた．加速器を運転していれば，現在機器がどのような状態（ステイタス）であるかモニターする必要がある．勿論，EM に聞きに行けばよいのだが，同じデータをあちこちのプログラムから何度も聞きに行くのは無駄が多い．EM が優先度の高い制御をしているときに割り込んでステイタスを聞くのも避けたい．しかしながら，そのために全く新たにデータ取得のためのプログラムを書きたくない．簡単にできないものか？　その要求を満たすために，Poller/Collector システムを作成した[10]．

Poller はデータを取るためのポーリングするプロセスの意味で，EM と同じコントローラ上で動作する．Poller の本体は EM を構成する関数群でできており，そこに制御のためのフレームを被せた感じになっている．ポーリングする周期は Poller 毎に 1 つしかないので，違った周期でデータを取るには複数の Poller を走らせる．さて，Poller にどのデータを取らせるかは，データベース上の管理情報からファイルを生成する．これを POLCOL ファイルと呼んでいるが，Poller は起動すると，POLCOL ファイルを読み込み，そこにかかれた機器のデータを取得する（get する）．put 機能はない．

Poller は取得したデータを，UNIX の共有メモリ機能を使って，リングバッファ状のメモリエリアに周期毎にデータセットを書き込んでいく．「やめよ」という指令を受けるまで，とり続ける．Poller に動作指令を送り，取得したデータを上位系に送るために，Collector Server（CS）プロセスを用意している．CS は複数の Poller を管理しており，管理下の Poller が取得したデータを，上位系で動作している Collector Client（CC）の求めにより送る．CS–CC はクライアント・サーバー型で，通信には ONC/RPC を使って

いる．CCはあらかじめ指定された収集周期に従って，データ送信要求を管理下の複数のCSに送り，上がってきたデータをまとめて，データベースにSQL（Standard Query Language）を発行して記録している．

MADOCAのユーザーはデータ収集の枠組みを利用するだけでよい．EMを作成すれば，Pollerは自動的に作成できる．CC，CSは提供される．あとは，どの機器のデータを，どんな周期でとるかをデータベース上に指定すれば，必要な管理ファイル，制御用ファイルは自動生成されるので何もしなくて良い．ここにフレームワークを使う利点がある．

3.3　高度な機器制御

少し複雑になるが，フィードバックなどの制御を上位系からネットワーク経由ではなく，下位の機器制御側で高速に行いたい，あるいは，トリガーなどのイベントに同期して制御を行いたい，という要求にも応える仕組みが用意されている．

前者の例では，Equipment Manager Agent（EMA）という仕組みがある．EMは通常の制御で必要だが，それと並行して，例えば，真空度を見ながらRFのパワーを上げていく処理も行いたいという場合がある．この場合は，EMの処理を代行してもらうために，EMAプロセスをEMから起動し，フィードバック制御を任せることができる．EMは本来，機器を制御するものなので，EMから制御されるEMAの事を「Pseudo device（仮想機器）」と呼んでいる．

後者の例では，トリガー・イベントを取得するEVGEN（EVent GENerator）と，イベントで駆動されるEMA-EV（EVent-driver型EMA）を用意し，EM, EVGEN, EMA-EVがプロセス通信できる仕組みを実装した[11]．**図4**にソフトウェア構成図を示す．この例はVMEを用いてデータ収集している．EVGENはVMEボードから来るトリガー信号（イベント）を監視し，イベントが発生すると，EMA-EVにデータ収集開始の指令を送る．EMA-EVはデータを取得すると，共有メモリーボードに書き込み，データはネットワーク経由で自動的に上位系に送信される仕組みになっている．これを同一共有メモリーネットワークで結ばれた，複数のVMEで並行して行うと，データの同期収集ができる．

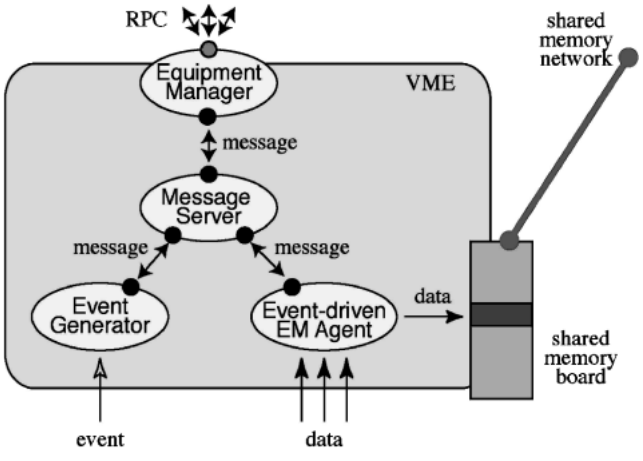

図4　イベント同期型のソフトウェア構成図．

4.　MAODOCAを使う

この解説記事をご覧になり，興味を持たれた読者のために，MADOCAはどのようなところに導入して使うことができるのか，少し触れておきたい．基本的には，特定のメーカー物はなるべく使わないように，オープン系のシステムで組めるように配慮している．

4.1　上位系動作環境

既に述べたように，フレームワークはUNIXを基本にしているので，上位系計算機にはUNIX系を使うことになる．動作可能なOSは，HP-UX, Linux, Solaris，そして最近Mac OS（Tiger）でも動作可能となった．制御の指令はC言語で書かれた数個の関数（API）を呼ぶことで送信できるので，GUIは使用者の都合に合わせて選択すればよい．SPring-8で使用しているX11ベースのX-Mateでも良いし，C言語と整合性のあるものであれば何でも良い．Linux搭載の手のひらサイズのPDAに移植して，機器メンテナンスに使うこともできている[12]．

ネットワーク系は，Ethernetベースで TCP/IPプロトコルであれば問題ない．帯域は100 Mbpsでも1 Gbpsでも用途に合わせて選択する．複数のファイバーをまとめて（トランク）広帯域化する，あるいは冗長化するなど，任意のアーキテクチャの上で利用できる．

データベースシステムはMADOCAの大きな機能の一つなので，是非とも用意してほしい．Sybaseは指定RDBMSなので，これが動作する計算機システムが必要になる．実績があるのは，HP-UXとRed Hat Linuxで，これらのOSに対応したライブラリー群が提供される．データ検索など高速に処理する必要があれば，複数のCPUを有するエンタープライズサーバー計算機を用いるほうがよい．最近では大容量メモリを搭載した高クロックの計算機も安価に入手できるようになった．もし，高い信頼性を求める場合は，SPring-8のように2台の計算機でクラスター構成とするのがよい．これに冗長化ソフトウェアを載せて，

高可用性システムを構築する．

4.2 機器制御系動作環境

機器制御系は幾つか実装方法があり得る．SPring-8 は VMEbus をメインに使っている．VME のシステムはシャーシに制御用の入出力ボード（モジュール）と CPU ボードを挿入して使用する．VME の CPU ボードの中には Pentium などのインテル系 CPU を搭載した機種があり，入出力ボードを制御するために OS を載せている．OS にはリアルタイム系の OS が選択されるが，SPring-8 では Solaris をメインに使っている．Solaris もオープン系となり，プロセス優先度制御など多彩な機能が使える．Solaris 用であれば各種ボードに対応するデバイスドライバーがある．CPU ボードでは Linux も動作するので，デバイスドライバーさえあれば動作可能となる．

VME 以外では，I/O ボード用のスロットを持った FA コンピュータで制御することも可能である．Linux を OS にして運用している実績がある．勿論，Linux が動作する一般的な PC に PCI カードを差して，わずかな点数の制御を行うことも可能である．

フィールドコントローラと呼ばれる，周辺機器制御でよく用いる PLC（Programmable Logic Controller）との取り合いが必要なときがある．今まではハードワイヤーでの取り合いに加えて RS232C 通信を行っていたが，最近，日本電機工業会規格の FL-net での通信がサポートできるようになった．VME ボードと PCI ボードと両方が可能となっている．

詳しくは触れないが，前述の共有メモリーネットワークを用いた実時間データ収集系も，フレームワークの一つとしてサポートされている．これは複数の VME あるいは PC を共有メモリーネットワークで接続し，データ取得指令（イベント）の発生をトリガーに，各 VME（あるいは PC）が同期して一斉に，データを取得する構成になっている．実際の応用例として，SPring-8 の線型加速器の BPM データをビームショットに同期して取得できている[13]．

4.3 提供される物

MADOCA のインストレーション・パッケージはまだできていないが，現在準備中．この中には，GUI 用の X-Mate 版各種グラフ作成ライブラリー，MS，AS，データベースライブラリー，EM（＋EMA），Poller, Collector Client/Server，各種デバイスドライバー，アラーム監視ライブラリー，Web 表示，開発用ツールなどが含まれている．これにリリースノートと，マニュアルなどのドキュメント一式がある．制御プログラム作成で有用なガイドラインを示した規定集もある．できる限りではあるが，何らかのサポート（質問回答，激励など）も提供できるようにしたいと思っている．

古くなった加速器制御システムを，MADOCA を使ってリプレースしたい場合，変更作業に着手してから新システムが動くまで，小型加速器であれば 1 年未満でできる．この機会に考えてみるのも良いかもしれない．

5. おわりに

MADOCA は SPring-8 で生まれたことを冒頭で述べた．MADOCA は SPring-8 建設の熱意に燃えた人々の勢いで生まれてきたものかも知れない．創造のメンバーは，今も，SPring-8 制御グループで MADOCA を磨いている．彼らのこれまでの努力と頑張りに感謝したいと思う．

最後に，この原稿を読んで有益なコメントを寄せていただいた，制御グループの増田剛正氏に感謝します．

参考文献

1） A. Gotz et al., Proc. of ICALEPCS ’93, Berlin, Germany, p. 22（1993）.
2） 「分散オペレーティングシステム」，A. S. タネンバウム著（水野忠則他訳），1996 年 8 月，トッパン，ISBN4–931356–21–4.
3） 「クライアント/サーバ紀行」，ダン・ハーキー他著（富士ソフト訳），1995 年 10 月，富士ソフト，ISBN4–89433–012–1.
4） 「SPring-8 の挑戦Ⅱ」，日刊工業新聞 2005 年 1 月 7 日.
5） R. Tanaka et al., Proc of ICALEPCS ’95, Chicago, USA, 1995 p. 201．田中良太郎，日本放射光学会「放射光」9 巻 3 号 p. 1（1996）. R. Tanaka et al., Proc. of ICALEPCS ’97, Beijing, China, 1997 p. 1.
6） A. Yamashita et al., Proc of ICALEPCS ’97, Beijing, China, 1997 p. 427.
7） A. Yamashita et al., Proc of ICALEPCS ’97, Beijing, China, 1997 p. 585.
8） A. Yamashita et al., Proc. of ICALEPCS ’99, Trieste, Italy, 1999, p. 434.
9） A. Taketani et al., Proc of ICALEPCS ’95, Chicago, USA, 1995 p. 625.
10） A. Taketani et al., Proc of ICALEPCS ’97, Beijing, China, 1997 p. 437.
11） T. Masuda et al., Proc. of ICALEPCS ’01, San Jose, USA, 2001.
12） Y. Ishizawa et al., Proc of PCaPAC ’05, Hayama, Japan, 2005.
13） T. Masuda et al., Nucl. Instrum. Methods A 543（2005）p. 415.