SANYO DENKI TECHNICAL REPORT No.7 May-1999
特集

AML言語の開発とその特長

吉川　紘
Hiroshi Yoshikawa

1. まえがき

FAオープン化をより一層推進するために、パソコンNC用の制御言語AML(Advanced Motion Language)を開発したので、その特長と開発のコンセプトについて概説する。

一般的にいって、「動作」を記述する言語は、「位置」に着目してプログラムを組むか、「時間」に着目してプログラムを組むかのどちらかである。前者の代表例としては、3軸直交座標系(Cartesian Coordinate)の動作記述言語Gコード(IEC RS274D)がある。後者の例としては、プログラマブルシーケンスコントローラ(PLC)の言語であるラダー言語LD(Ladder Diagram)などがある。これら2つの言語は30年にわたり別々の発展をしてきたが、最近になってソフトウェアPLCの規格(IEC 1131-3)の中で、Gコードとラダー言語を統合化しようという動きが出てきた。シーケンスプログラムの中でGコードプログラムを呼び出す方法で、座標(位置)を記述するプログラムとタイミングを記述するプログラムとを別々に組むという意味では、従来の方法と何ら変わりがなく本質的な解決とはなっていない。

AMLはイベント処理方式の採用によりこの統合化の問題を解決した。また従来の言語ではサポートできなかった同期制御、多軸制御のプログラムも簡単に組めるように、ドライブトレインという補間(Interpolate)方式を開発した。

2. AML言語の開発コンセプト

AML言語は、山洋電気が提唱しているオープンアーキテクチャーコントローラ「S-MAC」システムのなかの重要なコンポーネントのひとつである。

「S-MAC」システムは大別してType A、Type B、Type Cがあり、動作記述言語からみればType Aはロボット言語までをサポートしており、Type BはGコードを、Type CはAML言語をサポートしている。この章では、「S-MAC」システムType CとAML言語の関係、AML言語を開発した背景など中心にその特長を説明する。

また、他の制御言語との比較も試みる。

2.1フルソフトウェアコントローラとAML言語

図1は「S-MAC」システムType A、B、Cの応用面から見た位置付けである。AML言語は「S-MAC」 Type Cの動作記述言語であるだけでなく、SERCOS IDNを実行するランタイムエンジンでもある。この機能により、Type Cからモーションカードをはずすことが可能になった。

2.1.1「S-MAC」 Type C開発コンセプト

「S-MAC」 Type Cの開発時点での基本コンセプトは以下の7項目であった。

1. システムの開発期間を短縮したい。それには簡単な制御言語が必要である。
2. モーションバスは、国際規格のSERCOSを使いたい。
3. オブジェクト指向の命令体系にしたい。

4. 同期運転を速度制御でなく位置制御でやりたい。それには回転座標系を定義できる言語がほしい。
5. ハードウェアの補間(モーションカード)は使いたくない。すべてソフトウェアでやりたい。
6. 外部にシーケンサを付けたくない。PLCをソフト化したい。
7. 補正に関して、切削加工用のコントローラで使われているオーバライド方式（ハード補正）ではなく、塑性加工用のコントローラで使われているレシピ方式(ソフト補正)にしたい。

7項目のうち、ひとつでもかけたらフルソフトウェアコントローラとしての特長が生かせない。といって、ゼロから言語の開発をしていたのでは商品化が遅れる。そこで、世界中の既存の言語を調査した。そして見つけたのが米国Automation Intelligence社が包装機械システム用として開発していたAML言語である。「S-MAC」 Type Cの制御言語として不足する部分は共同開発により追加修正され、現在のAMLがある。開発の歴史を図2に示す。

（注)C言語で組んでいた時から14年、Windows環境にのせてからでも4年経過している。

2.1.2 AML言語と「S-MAC」システムの開発コンセプトの一致点

AML言語にしても、「S-MAC」 Type Cにしても、システム設計を進めていく上で重要な選択をしている。

その選択肢が一致したからこそ、AML言語は「S-MAC」 Type Cに最適な言語と位置付けられ、種々の応用システムへの組み込みが可能になった。一致した重要な選択肢とは以下の3項である。

1. PC based Controller
   ハードウェアプラットフォームとしてPC(IBM compatible)を採用したこと。
2. SERCOSモーションバス
   サーボドライバとの通信プロトコルとしてオープン規格のSERCOSを採用したこと。
3. リアルタイムOS
   SERCOS実行環境はリアルタイムOSに依存しないマルチプラットフォーム化したこと。

2.2 AML言語を使用するメリット

オープンアーキテクチャのコントローラを採用するメリットは、一にも二にも開発リードタイムの短縮にほかならない。AML言語を採用することにより、開発サイクルタイムがどのように短縮されるか見てみる。

一般に、サーボモータを用いたコントロールシステムを新規に開発する場合の開発サイクルは次の手順になる。

1. システム設計(ハードとソフトの分担範囲を決める)
2. ハード設計、ソフト設計
3. ハードの調整(マニュアル運転)
4. サーボ系調整(単軸プログラム運転)
5. 動作検証(加工プログラム、シーケンスプログラムの結合テスト)
6. 総合調整(HMI(Human Machine Interface)との結合、精度、加工タクトタイムなど検証)

新商品の開発サイクルとしては、単純なシステムで6ヶ月、複雑なシステムでも1年というのが一般的な目標値である。時間軸そのものは短縮できないが、AML言語を採用することにより関連技術者を拘束する時間が減るため(デバッグの効率が上がるため)、確実に従来の1/2～1/3程度にまで工数削減が見込める。

たとえば、動作プログラムとシーケンス制御プログラムを同一人が組めるので、シーケンス制御プログラムを組む技術者が必要なくなるだけでなくインタフェース仕様書も必要最小限のもので済み、これだけでも設計工数の大幅削減ができる。また、AML開発環境そのものがプログラミングツールという機能よりオンラインデバッグツールという機能を重視して作られており、システムの動作検証作業の期間が大幅に短縮できる。

2.3 他の言語との比較

表1　従来の言語との比較表

	◎＞○＞□＞△	AML	汎用言語 C, VB, ALGol	G_Code	PLC	Robot Language	Local Command
動作	PTP動作	◎	○	◎	○	○	◎
	多軸補間	△	○	○	△	◎	○
	同期	◎	□	□	□	○	□
	超多軸(16軸以上)	◎	□	△	△	○	□
	ロボット（多関節）	△	○	△	△	◎	□
	制御応答	□	○	□	△	○	◎
プログラミングのレベル	プログラム開発時間	◎	□	○	□	○	□
	ネットワークドライブ	◎	○	△	△	○	□
	チューニング、モニタ	◎	□	○	△	◎	○
	上位通信 (E-net、 インターネット)	◎	◎	△	△	◎	○
機構（オブジェクト）	電子カム、ギア	◎	○	○	□	○	○
	電子シャフト （回転周期）	◎	○	□	□	○	□
	マスタ、 スレーブ バーチャルマスタ	◎	○	○	□	○	○

一般的に制御言語そのものは、開発段階でかなり明確に対象とする機械（システム）を絞り込んでおり、各々独自の目的があり厳密な意味での比較は難しい。5軸のシステムのコントローラを想定して、制御プログラムを汎用言語Cで組んだ場合を基準に、あえてランク付けをしてみたのが表1である。電子カム、電子ギヤの機能については、他の言語がサポートしていないので、AMLの得点が当然高くなる。

3. AML言語ソフトウェアの構成

3.1 開発環境と実行環境

AML言語を用いた制御システムは、開発環境PCと実行環境PCで構成される。システムの基本構成を図3に示す。

開発環境は主にAMLアプリケーション（AML言語で記述した複数のプログラムのこと）の作成や、開発環境および実行環境の設定に使用される。

実行環境はAMLアプリケーションを実行し、SERCOSリングに接続されている機器を制

御するために使用される。

開発環境と実行環境はRS232-Cケーブルまたは Ethernetケーブル(10BASE-T)を用いて接続され、AMLアプリケーションをダウンロード・アップロードしたり、実行状況の監視に必要なデータを転送できるようになっている。

一般的な制御システムを構築するにはHMI(Human Machine Interface)や上位システムによる管理が必要となる。このような場合でも、RS232-CケーブルやEthernetケーブルを用いてコンピュータ間を接続できる。

ここで、AML制御システムで構築可能な開発環境、実行環境の構成例を紹介する。

図4aは一台の開発環境と複数台の実行環境で構成した開発システムの例である。AMLは開発環境一台で各実行環境のAMLアプリケーションを管理できるため、Ethernetケーブルを介して一対多の制御システムを構築することができる。この場合、一台の開発環境で、ある実行環境の実行状況を監視しながら別の実行環境のAMLアプリケーション開発を行う、といったことが可能である。ただし、同時に同じ作業、たとえば複数の実行環境の実行状況監視を行うことはできない。

図4bは実行環境一台の開発システム(スタンドアロン)である。通常は実行状況の監視の必要がなくなる時点で、すなわちAMLアプリケーションの開発が終了した段階で開発環境を切り離し、実行環境単体の制御システムを構築することができる。AMLの実行環境には電源投入後にシステムを自動起動できる機能が組み込まれているため、このようなシステムを構築できる。

このように、AMLシステムは開発環境と実行環境の1対1の構成を基本として、システムを容易に拡張できる点に特長がある。またこれは開発環境側のAMLプログラムと実行側のプログラムの一元管理を可能にした。

3.2 開発環境

本節では、開発環境について詳述する。

3.2.1 用途

開発環境は、主に次の用途に使用する。

1. 開発環境および実行環境の設定
2. AMLアプリケーションの開発(プログラミング)保存
3. AMLアプリケーションの起動と停止
4. AMLアプリケーションのデバッグ
5. 実行状況の監視

3.2.2 ハードウェア

開発環境用に使用できるハードウェアは、当社製「SMS-10」またはPC/AT互換の市販パーソナルコンピュータである。前節で述べたように、実行環境と通信を行う必要があるため、少なくともRS-232Cシリアルポートまたは Ethernetポート(10Base-T, RJ-45ジャック)を装備していなければならない。

3.2.3 オペレーティングシステム

開発環境で使用するオペレーティングシステムはWindows3.1、95またはNTである。

3.2.4 ソフトウェア

開発環境で使用するAMLソフトウェア群について説明する(図5参照)。

ユーザが直接操作するソフトウェアには以下のものがある。

1. Configuration Tool
   開発環境、実行環境およびSERCOS機器の設定に使用する。
2. Maintenance Tool
   SERCOS機器の動作確認に使用する。
3. AML Development Environment
   AMLアプリケーションの開発(プログラミングとデバッグ)、起動および停止に使用する。
4. LogViewer
   実行環境の実行状況をエラー、ワーニングあるいはユーザメッセージとして表示する。
5. SERCOScope
   SERCOS通信データ(例えばモータ指令値やフィードバック値など)をリアルタイムでグラフ表示する。

バックグラウンドで実行されているソフトウェアには以下のものがある。

1. MPP(Multi Purpose Protocol)
   実行環境との通信処理を行う。上記AMLソフトウェアが送受信するデータはすべてMPP経由となっている。
2. Pipeline
   制御システムによって実行環境とデータを直接やりとりする必要がある。たとえば、VisualBasicで作成したHMIで制御システムを操作したり、モータの現在位置をExcelで取り込み何らかの処理を行う場合が考えられる。PipelineはWindowsアプリケーションと実行環境でデータの授受を行えるようにする。このとき使用されるプロトコルはDDE(Dynamic Data Exchange)である。

3.3 実行環境

本節では、実行環境について詳述する。

3.3.1 用途

実行環境は、主に次の用途に使用する。

1. AMLアプリケーションの実行
2. 接続機器の制御

3.3.2 ハードウェアプラットフォーム

実行環境に使用するハードウェアは、「S-MAC」コンポーネンツのひとつである「S-MAC PC」「SMS-10」である。これはPC/AT互換のパーソナルコンピュータであるが耐環境性にすぐれており、SERCOSインタフェースを標準で装備している。

3.3.3 オペレーティングシステム

実行環境で使用するオペレーティングシステム（OS）は、Radisys社製リアルタイムOSのiRMXである。順次VxWorks版、Windows NT(RTX)版も、リリースしていく予定である。

3.3.4 SRX

SRX(SERCOS Runtime eXecutive)はリアルタイムOS上で動作する実行環境側のAMLソフトウェアの総称で、AMLアプリケーションを実行し接続機器を制御するランタイムエンジンである。図6に実行環境ソフトウェア構成を示す。

図6では外部デバイスと通信するタスクを主に記述してあるが、その他にモータの制御に関係するタスク群やインタプリタなどがある。これらのタスクはメールボックスやセマフォなどを使用して互いに連携し、協調しながらシステムの処理を行っている。

ここではモータの制御に関係するタスク群であるMCU(Motion Control Unit)について記述する。

MCUでもっとも重要なタスクに、SERCOSインタフェースカードからの定期的な割り込み（1msec単位で1～20msecの範囲で設定可能)によって処理を行う割り込みタスクがある。この主な処理はSERCOSデバイスと通信し、定期的にデータの送受信を行うことである。また、指定した移動速度や加減速度から生成した加減速プロファイルを使用して、割り込み周期でモータ指令値の計算と送信を行ったりフィードバック値の受信を行う。さらに指定速度に達したり動作が終了したときにイベントを発生し、他のタスクに通知する処理も行う。AMLで特長のある機能の一つにドライブトレインがある。これはマスタ軸とスレーブ軸を同期させることにより電子的にギヤやカム動作を実現するものであるが、この同期処理もこのタスクで行っている。

MCUにはその他にホーミング処理を行ったり、ドライブトレインのための前処理を行うタスクなどがある。

3.3.5 マルチタスク

前項では、実行環境側のPCにおいては、リアルタイム・マルチタスクシステム上で複数のシステムタスクが連携して処理を行っていることを説明した。

ユーザが扱うAMLアプリケーションでも同様にマルチタスクシステムを採用している。AMLアプリケーションは複数のモジュールで構成されている（図7参照）。

ここで「AMLアプリケーション」とは、一つの制御システムに一つ用意されるユーザが構築する特有のソフトウェアシステムを意味し、複数のモジュールで構成される。

また「モジュール」とは、AML言語で記述されたプログラムを意味し、SRX上では独立したタスクとして処理されている。つまり、AMLアプリケーションに複数のモジュールが存在すると、モジュールはそれぞれが別々のタスクとして処理されている。

しかし、モジュールのタスクプライオリティ（優先順位）は固定で、モジュールごとに異なるプライオリティを設定することはできないという制限がある。そのため複数のモジュールを同時に実行する場合、タスクスイッチングを使用している。これは最大占有時間を越えるとモジュールは実行を一時中断し、次のモジュールに実行を移す仕組み（ラウンドロビン方式）のことである。このおかげで、あるモジュールが実行を占有し他のモジュールが実行できない、という状況を回避できる。

一般に、タスクごとに異なるプライオリティを設定し、システム全体が秩序よく実行できるようにする作業（タスクスケジューリングと言う）は専門家でも難しい問題である。AML

アプリケーションの実行にラウンドロビン方式を採用したのは、マルチタスクシステムに精通していないユーザでもシステムの構築を可能にするためである。

モジュールは独立したタスクであるがそれぞれが協調して処理できるメカニズムも用意されている。

その一つとしてあるモジュールから別のモジュールを制御する機能がある。つまり、実行環境へのモジュールのロード、起動、停止、アンロードである。この機能を利用することにより、多数のモジュールが存在するシステムを、限られたCPU資源しかもたない実行環境でも高いパフォーマンスで処理することができる。

その他の機能として次項で説明するイベント処理機能がある。

以上、これらの機能を有効に活用することによりモジュールごとに機能を割り当てたプログラムを作成でき、システム全体として見通しのよいAMLアプリケーションを構築することができる。

3.3.6 イベント処理機能

イベント処理機能はAMLアプリケーションの特長の一つである。これは、PLC制御システムで一般的なスキャニング処理（またはポーリング処理）と比較し次の利点がある。

1. CPU資源を有効に利用できる。
2. イベント発生後の応答時間が一定である。

ここで、イベント処理とスキャニング処理の違いを説明する（図8参照）。

(1)スキャニング処理（図8a）

1. 待機モジュールはシステム状態が変化するまで一定周期でシステムを監視する。この間、タスクとしてアクティブになっておりCPU資源を消費している。
2. イベントソースが状態を変更すると待機モジュール自身が検出しスキャニング処理を終了する。
3. 待機モジュールは処理を再開する。

(2)イベント処理（図8b）

イベントはシステム（OS）が管理・監視する。

待機モジュールはシステムからの起動を待つ。待機中のタスクはスリープ状態なのでCPU資源を消費しない。

1. イベントソースがイベントを発生すると、システムが待機モジュールのタスク状態をスリープからアクティブに変更する。
2. 待機モジュールは処理を再開する。

3.4 AMLインタプリタ

AML言語はインタプリタ方式を採用している。ここで、インタプリタ方式とは、ソースコード（作成したプログラム）を実行時に一行づつ処理していく（逐次処理）実行方式を意味する。一般にインタプリタ方式は、コンパイルして実行する処理方式に比較して処理速度は遅い、という欠点がある。反面、コンパイルする必要がないためにソースコードの扱いが楽である、という利点がある。AML言語では利点を維持しつつ欠点を補うために、実際にはソースコードではなく中間コードを使用している。

中間コードは編集時に自動生成されるバイナリファイルである（拡張子：AML）が、(a)編

集画面に表示されるAMLプログラムと(b)実行環境で処理される実行ファイルの双方に使用している。

中間コードの扱いを明確にするために、図9にAMLアプリケーションの開発から実行までの手順を示す。

(1)編集
開発環境のAMLソフトウェアを使用して、AMLアプリケーションを構成するモジュールをAML言語で作成・編集する。このとき中間コード(拡張子:AML)が自動生成され、ハードディスクにセーブされる。

(2)ダウンロード
開発環境のAMLソフトウェアを使用して、中間コードを開発環境から実行環境にダウンロードする。この結果、中間コードは実行環境のコンパクトフラッシュメモリにコピーされる。

(3)ロード
開発環境のソフトウェアまたは実行環境のオートロード機能を使用して、中間コードを実行環境のRAMにロードする。このとき、中間コードに含まれるコメントなど実行に不必要な部分は取り除かれる。

(4)実行
開発環境のソフトウェアまたは実行環境のオートロード機能を使用して、中間コードを実行する。先に説明したように、中間コードは編集時点でインタプリタ(逐次処理)で実行できる形式になっている。

PLCなどのコントローラでは、(a)開発環境でソースコードの編集、(b)コンパイルにより実行コードの生成、(c)実行環境へのダウンロード、(d)実行環境で実行と編集、という手順が一般的である。そのため、デバッグ作業を繰り返すうちにソースコードと実行コードが一致しなくなる、という問題が発生する。現場で実行コードを修正できるという利点がある反面、メインテナンスが難しくなる。

一方AMLではいままでの説明からわかるように、編集と実行の双方で同じ中間コードを使用する方式であるため、ソースコードと実行コードの不一致は決して発生しない。

4. AML言語の特長

前章ではAML言語のソフトウェアシステムの点から見た特長を紹介した。本章ではAML言語の動作記述プログラミングシステムとしての特長を紹介する。

4.1 プログラム

AML言語はPASCALに似た高級プログラミング言語を採用している。そのためラダー言語やGコードなどで記述されたプログラムに比較して可読性が良く、プログラミングが簡単である。また、デバッグなどのメンテナンスも容易となる。

基本文法は一般的な高級プログラミング言語と同じである。プログラムは ” DECLARE ” で始まる宣言文と、 ” BEGIN ” と ” END ” ではさまれた実行文で構成される。

関数呼出は一般に使用される呼出のほかに、AML言語特有の呼出が追加されている。

さらに、制御に必要な機能を集約したオブジェクトが用意されている(次節参照)。

4.2 オブジェクト

AML言語はオブジェクトベースの高級言語である。つまり、オブジェクトの概念を採用し、制御によく使用する機能ごとにオブジェクトととして定義してある。ここで、「オブジェクト」とは、データメンバとメソッドで構成されたソフトウェアブロックのことで、「データメンバ」はメソッドの実行に必要となる設定データを表し、「メソッド」はオブジェクトの処理を行うルーチンである。

AML言語はユーザが新たなオブジェクト（あるいはクラスとも言う）を定義することを、原則として許していない。このことは、高級言語、特に構造化プログラミングの経験者にとってC++やJAVAといったオブジェクト指向言語と比較して制約と感じるかもしれない。しかし、これはあくまで構造化プログラミングにあまり経験のないユーザでも、容易で安全に制御プログラムを記述できるように配慮したためである。新しいオブジェクトは必要に応じて追加してゆく。

ここで、AML言語ですでに用意されているオブジェクトをいくつか紹介する。

4.2.1 MOTORオブジェクト

用途：

システム設定で定義したモータを、AMLで使用できるように割り付け、モータの制御を行う。

データメンバ（一部）：

DefaultAccel：加速度のデフォルト値

DefaultSpeed：速度のデフォルト値

メソッド（一部）：

On()：モータを励磁する

Off()：モータの励磁を切る

Start()：モータを指定した移動方法で動かす

4.2.2 ABS_MOVEオブジェクト

用途：

絶対位置動作指令を設定する。

データメンバ（一部）：

Endpoint：終点

Mode：移動方向と方法

Speed：移動速度

メソッド（一部）：

NotifyCompleteAs()：動作終了時、イベントを発生する

NotifySpeedAs()：速度到達時、イベントを発生する

4.2.3 その他の主要オブジェクト

DRIVE_TRAIN

ドライブトレインの割付と制御を行う

EVENT

イベントハンドリングに使用する

IO_BOOLEAN

入出力の割付と制御を行う

PLS

プログラマブルリミットスイッチの設定と実行を行う

4.3 オブジェクトを使用したAMLプログラムの例

図10にAMLプログラムの例を示す。

```
// 宣言部
DECLARE MoveComplete EVENTID
DECLARE MyEvent EVENT

DECLARE count WORD
DECLARE MainAxis("Axis_1") MOTOR
DECLARE MyMove() ABS_MOVE

// 実行部
BEGIN

// モジュールで指定イベントを受信可能にする
LOOKFOR(MyEvent)

// MOTORオブジェクトのデータメンバ設定
MainAxis.DefaultAccel = 500
MainAxis DefaultDecel = 500

// ABS_MOVEオブジェクトのデータメンバ設定
MyMove.Mode = MOVE_FORWARD
MyMove.Speed = 100
MyMove.Endpoint =0
MyMove.NotifyCompleteAs(MoveComplete)

// モータの励磁
MainAxis.On()

// モータを絶対位置動作で初期位置へ移動
MainAxis.Start(MyMove)
// 動作終了の待機
MyEvent.Wait()

FOR count FROM 1 to 10 DO
IF (count%2 == 1) THEN
// 順方向移動量設定
MyMove.Endpoit = 500
ELSE
// 逆方向移動量設定
MyMove.Endpoint = 0
ENDIF

// 絶対位置動作で実行
MainAxis.Start(MyMove)
// 動作終了の待機
MyEvent.Wait()
ENDFOR

// 動作停止
MainAxis.Stop()
// モータ励磁断
MainAxis.Off()

// メッセージの表示
LogMsg("Abs Motion Complete")

END
```

※実際のAMLでは日本語を使用することはまだできない。

図10　AMLプログラムの例

5. 実際の応用例

5.1包装機への応用

ピロー型の包装機械への応用例を示す。

AMLプログラムサイズは3軸システムで約3千行(コメント含む)である。

5.1.1軸構成

図11にシステムの軸構成を示す。各軸の機能は以下のとおりである。

Lug chain軸：品物の同期送り制御

Fin seal軸：縦方向のシール制御

Crimp軸：横方向のシール制御

5.1.2概観

図12にシステムの概観を示す。

5.2 巻線機への応用

モータ分割コア用の巻線機で、コア側を回転させて、巻き付けて行く方式のシステムを示す。

AMLプログラムサイズは5軸で約2千行(コメント含む)である。

5.2.1軸構成

図13にシステムの軸構成を示す。各軸の機能は以下のとおりである。

θ軸：コアの回転同期制御

X軸：X方向のオフセット（移動）制御

Y軸：Y方向のオフセット（移動）制御

Z軸：Z方向のオフセット（移動）制御

S軸：ワイヤのしごき制御

図13 巻線機軸構成

5.2.2概観と加工例

図14にシステムの概観と加工サンプルを示す。

5.3転造盤への応用

ネジ転造盤への応用例で、左右のダイスの同期をとりながら寄せて加工する方式を示す。

AMLプログラムサイズは5軸で約1万2千行(コメント含む)である。

5.3.1軸構成

図15にシステムの軸構成を示す。各軸の機能は以下のとおりである。

SR軸:主軸(右)の同期制御

SL軸:主軸(左)の同期制御

TR軸:傾斜軸(右)の同期制御

TL軸:傾斜軸(左)の同期制御

M軸:寄せの同期制御

5.3.2概観と加工サンプル

図16にシステムの概観と加工サンプルを示す。

6. むすび

オープン化、ソフト化というFA業界の動向に対応して、AML言語というフルソフトウェアコントローラ用の次世代言語を開発した。

レーザ加工機などの制御で必要とされる高速高精度の円弧補間、多関節ロボットのコントローラで必要とされる高速座標変換機能など、これから追加していかなければならない機能はあるが、従来の言語(Gコード、ラダー言語など)でサポートできなかった分野に対しては、基本機能の範囲だけでも十分な解決策を用意できる。今後もお客様とともにAML言語のオープン化を推進するとともに、実行環境のマルチプラットフォーム化を順次行っていき、この分野(包装機械、転造盤、巻線機など)でのデファクトスタンダード言語を目指したい。

本稿をまとめるにあたって、とくに第3章と第4章については小林英一氏に協力していただいた。文末ながら感謝する。

＊本文中の会社名と商品名は、それぞれ各社の登録商標または商標です。

吉川　紘
1996年入社
サーボシステム事業部　コントロールシステム推進部
コントローラの開発に従事。

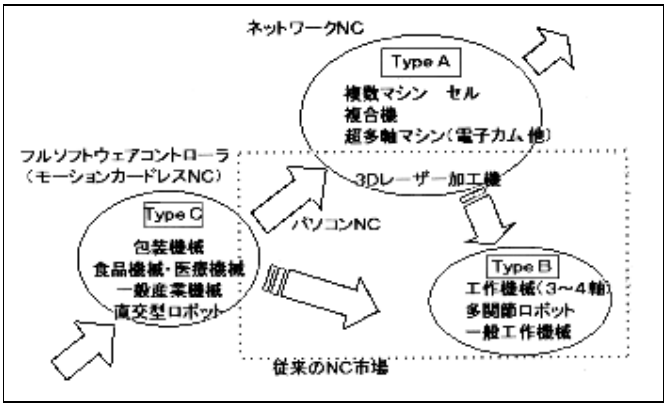
ネットワークNC
Type A
複数マシン　セル
複合機
超多軸マシン（電子カム 他）
フルソフトウェアコントローラ
（モーションカードレスNC）
3Dレーザー加工機
Type C
パソコンNC
包装機械
食品機械・医療機械
一般産業機械
高交型ロボット
Type B
工作機械（3～4軸）
多関節ロボット
一般工作機械
従来のNC市場

図1 「S-MAC」システム

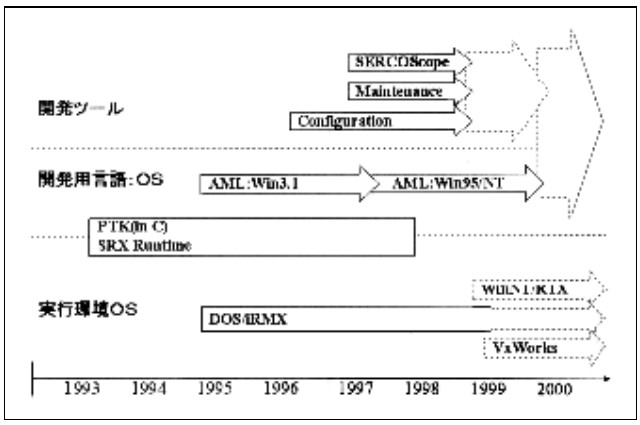

図2 AML言語開発の歴史

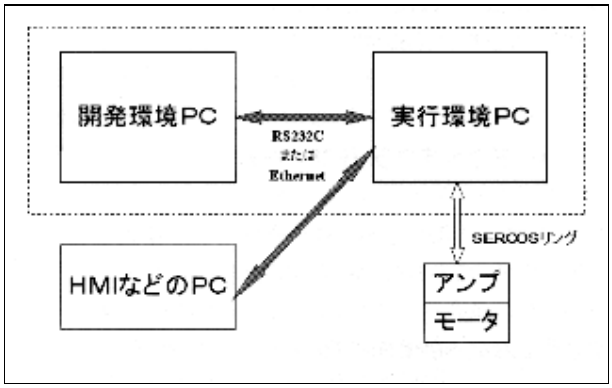

図3 AML制御システム基本構成

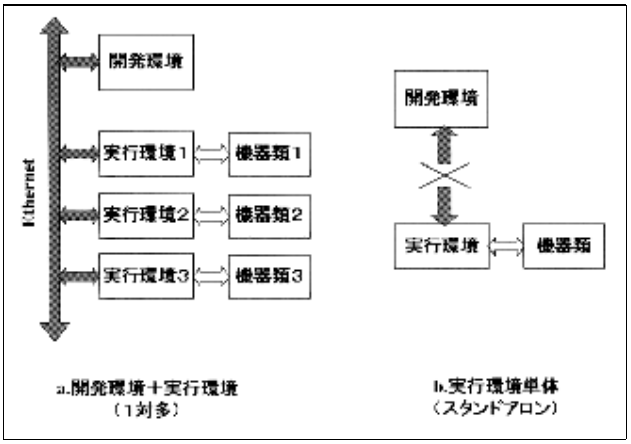
Ethernet
開発環境
実行環境1
機器類1
実行環境2
機器類2
実行環境3
機器類3
a.開発環境＋実行環境
（1対多）
開発環境
実行環境
機器類
b.実行環境単体
（スタンドアロン）

図4 システム構成例

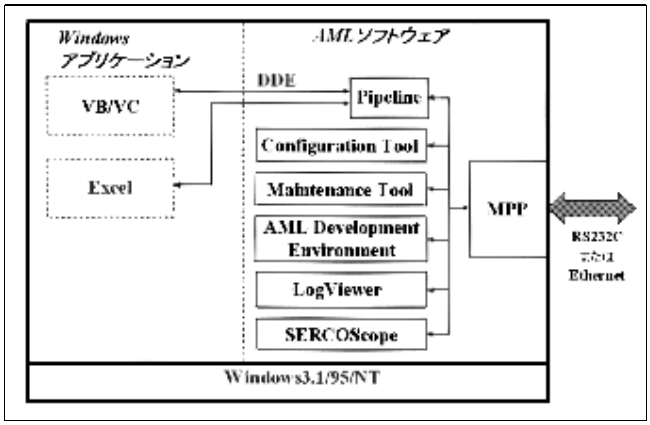

図5 開発環境ソフトウェア群

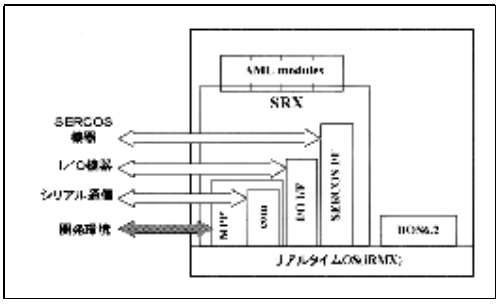

図6　実行環境ソフトウェア群

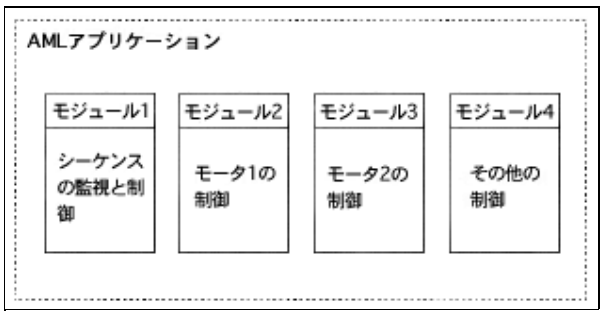

図7　AMLアプリケーションとモジュール

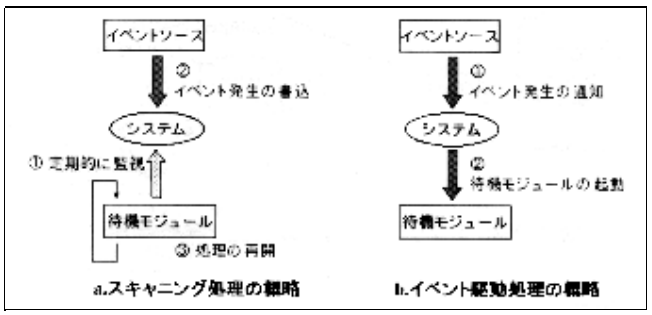

図8 スキャニング処理とイベント処理の違い

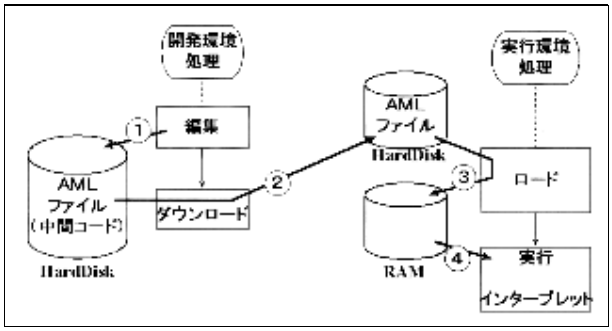

図9 AMLアプリケーションの開発と実行手順

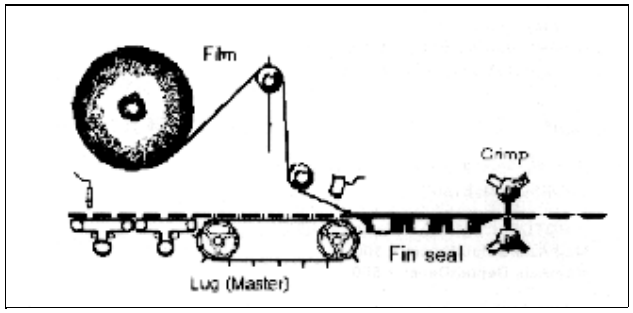

図11 包装機軸構成

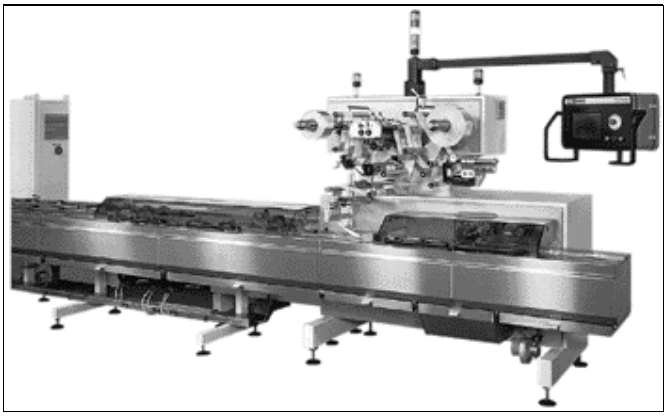

図12　包装機システム概観図

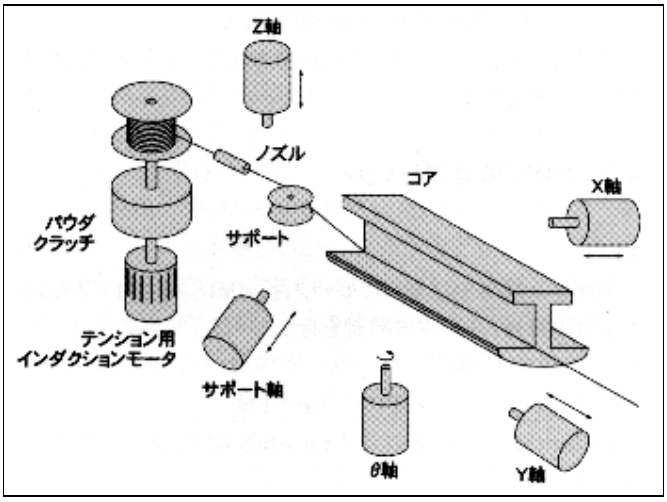

図13 巻線機軸構成

図14　巻線機概観と加工サンプル

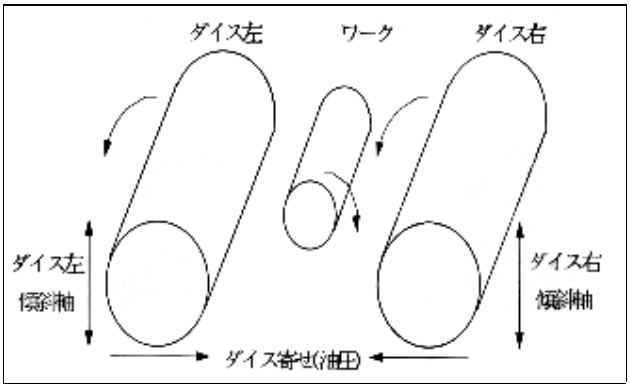

図15 転造盤軸構成

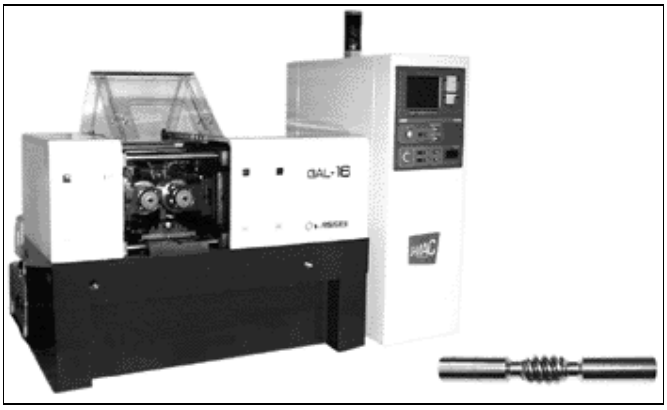

図16　転造盤概観と加工サンプル