第1章　作業を始める前に

パソコンでリアルタイム制御を始めるにあたって，組み込みプログラム開発技術者が，いま，どんな環境におかれているかについて述べます．

技術がめまぐるしく変化する状況において大事なことは，潮の流れ，変化の方向，将来の姿を読み取ることです．

何が変化しているのか，変化をもたらしている原動力は何か，変化のトレンドはどこを指しているのか，それらについて考察します．

具体的な技術が必要だという読者は，本章をスキップして，第2章へジャンプしてください．

■ 1.1　モデル・ベース開発の発展

大学の研究室や企業の開発部門において，機械の制御装置を開発する際に，モデル・ベース開発（Model Based Design: MDB）あるいはモデル駆動開発（Model Driven Development: MDD）の手法を採用するケースが多くなりました．

モデル・ベース開発を採用すると開発の過程はどのようになるか，その進行過程をスキット風に述べます．

仮にいま，化学プラントの制御装置を製作するとします（**図1.1**）．

プラントには，いくつかの反応器，触媒供給装置，反応器を結ぶパイプライン，調整弁，加熱バーナなどがあります．

まず，反応器内で起こる化学プロセスを分析します．

化学反応器には，いくつかの状態量があります．例えば，原材料の供給量，反応器内温度，圧力，出力物質の組成比などです．状態量の中には，観測可能な量とそうでないものがあります．測定可能な状態量を可観測量（observable）といいます．観測可能でない状態量に対して，必要ならば推定のための数式モデルを作ります．

化学反応器には，いくつかの制御量があります．例えば，器内温度，撹拌速度，触媒添加量，弁の開度などです．制御量を変えると，それにしたがって反応器の出力は変化します．

これらの変数を使って，反応器の数学的なモデルを作ります．これらは通常なら複雑な数式モデル

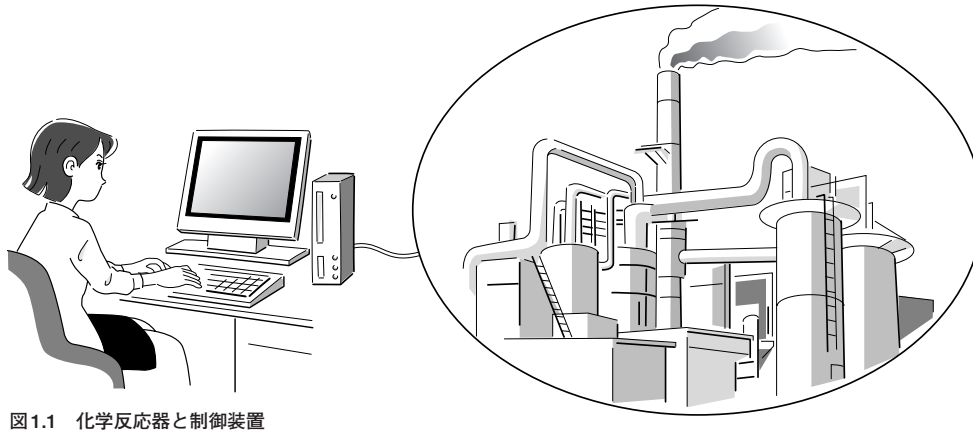

図1.1　化学反応器と制御装置

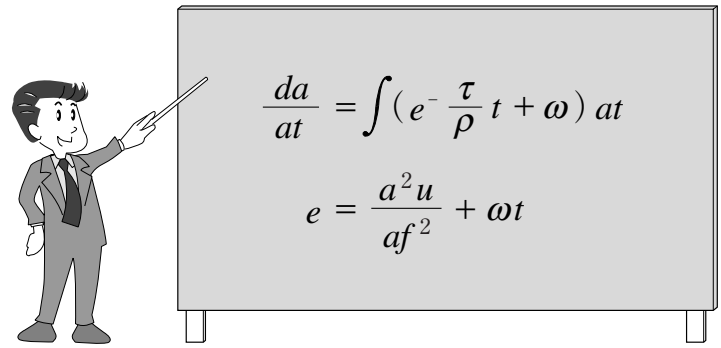

図1.2　数式モデルの例

になります（**図1.2**）．

反応器の数式モデルができあがれば，反応器を一つのブロックとして，コンピュータへインプットします．コンピュータのディスプレイ上にブロックをドラッグ・アンド・ドロップして，そのブロック内に数式モデルを書き込みます．必要ならば，ブロックのプロパティを変更します．

ブロックを書いたら，次はブロックを線で結びます．これらの線は，プラント内のパイプラインに対応します．コンピュータのディスプレイ上に，ブロックの線図ができます．このブロック線図を，本書ではスキーマ（schema）と呼びます（**図1.3**）．

スキーマは，ブロックとそれらのブロックを結ぶ線分から構成します．

ブロックは，入力ポイントあるいは出力ポイントのいずれか，あるいはその両者があります．入力ポイントなし，かつ出力ポイントもなし，というブロックは存在しません．

スキーマは，物理的なプラントをコンピュータ内に論理的に実現するものです．スキーマと言う代わりに，モデル（model），あるいはチャート図（diagram）と言うこともあります．

スキーマができあがったら，コンピュータを使ってシミュレーションを行います．

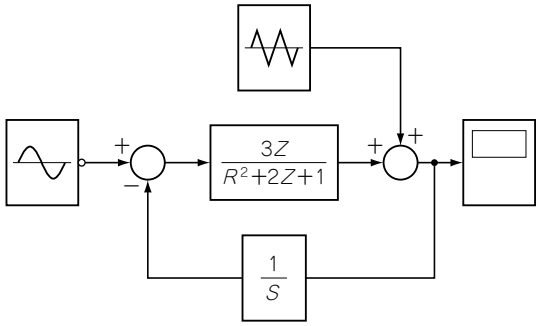

図1.3　化学プラントのスキーマ

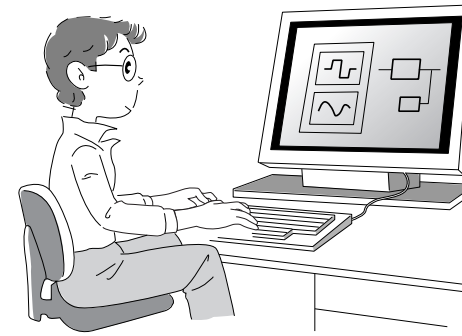

図1.4　シミュレーションのグラフ表示

ウインドウのツール・バー上のボタンをクリックして，プラントのシミュレーションを開始します．シミュレーションが進行すると，反応器内の温度，流量などの状態量がグラフ表示されます（**図1.4**）．

仮にシミュレーションの結果，反応器内の温度が設定値よりも低く，出力は設計値に至らなかったとします．

この場合，加熱ヒータのフィードバックの比例ゲインが不足していたと推測して，このゲインを10%上げて，再度シミュレーションを行います．

今度は，ゲインが大きすぎたために反応器は不安定になり，器内温度が激しく乱高下してしまいました．しかし，これはコンピュータ・シミュレーションなので，心配することはありません．反応器内温度が乱高下しても，反応器は爆発したりはしません．安心してください．

……

と，こんな風に，シミュレーションを進めてフィードバックのゲインのチューニングします．

以上で，制御に必要なフィードバック・ゲインの調整は完了しました．

続いて，実プラントにおける検証過程に入ります．

スキーマから制御ブロックを抽出して，そのブロックをCのプログラムへ変換し，それをコンパイラにかけて実行プログラムをビルドします．ビルドした実行プログラムを制御用のターゲットのパソコンへダウンロードします．

このパソコンは，アナログ入出力ボードなどを介して，プラントと直結します．制御プログラムを実装したパソコンの制御によって，プラントを稼動し，実世界のデータを採取します（**図1.5**）．

プラントの応答を調べて，シミュレーションの数学モデルを修正したり，制御のためのフィードバック・ゲインを再調整したり，あるいは制御アルゴリズムをPID制御から最適制御に変えたり，そういった変更を必要に応じて行います．

仮に，実機による実験データを分析した結果，反応器の寸法を変更する必要があるという判断に達したとします．そんなときでも，プログラムを自ら手直しする必要はありません．

コンピュータのディスプレイ上で，スキーマの数式モデルを変更します．変更したスキーマにビルドをかけると，新しい実行プログラムが自動的に生成されます．このプログラムを制御用のコンピュータ

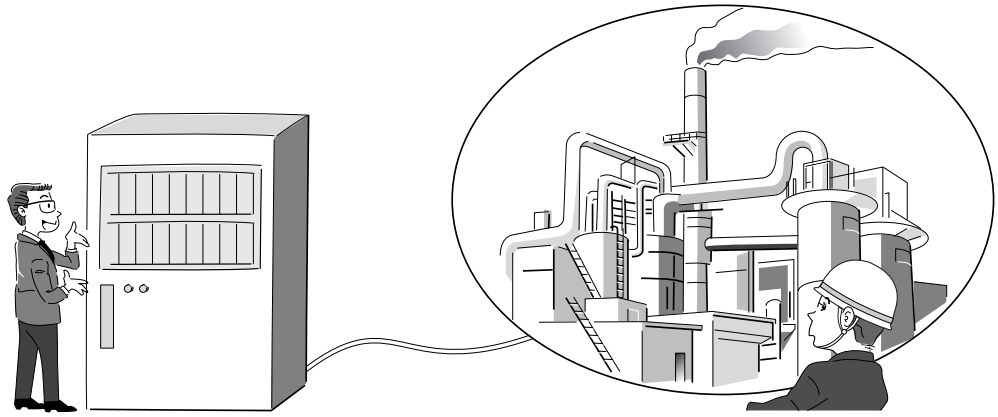

図1.5　実プラントにおける検証

へダウンロードすれば，直ちに実プラントにおけるデータ採取が可能になります．

モデル・ベース開発において，開発を行うホスト・コンピュータとプラント制御が直結しているので，上流のスキーマを変更することによって，すぐに物理プラントにおける実証実験が実施可能になります．

モデル・ベース開発に関して，もう一つのスキットを示します．

舞台は，同じく化学プラントの制御系設計問題のシーンとしましょう．

プロジェクトは順調に進行していたのだけれども，海外から大型プラントの受注が入り，試作部の全スケジュールが組みかえられて，化学反応器に関する試作機の製作が3ヵ月遅れることになったと仮定します．

社外秘の特命プロジェクトなので，試作を外注へ出すことはできません．あくまでも内製する必要があります．

制御装置はすでに完成しており，実機待ちの状態です．

皆さんなら，どうしますか．

これは良い機会だなどと言って，ちょいとハワイへ海外旅行に出かけたりしますか？ まさか，そんなことはないでしょう．若手技術者の提案で，シミュレーションに使用したモデルを制御部とプラント部に分割します（**図1.6**）．

プラント部をC言語に変換して，これをコンピュータ内に実現します．

制御装置の実機とプラントの動作をシミュレーションするコンピュータを接続して，仮想的な実験を実施し，制御プログラム内のバグ取りを行います．

制御装置の実機とプラントのシミュレータを接続して制御プログラムの検証を行う方法を，通常，HILS（Hardware In the Loop Simulation）といいます．HILSによって，制御プログラムの検証が完了するとは言いませんが，実機実験に必要な期間の短縮に貢献することは確実です．

一般に，プラントの規模があまりにも巨大で，試作プラントを作ることができない場合や，制御装置がプラントよりも早期に完成する場合などにHILSを採用します．

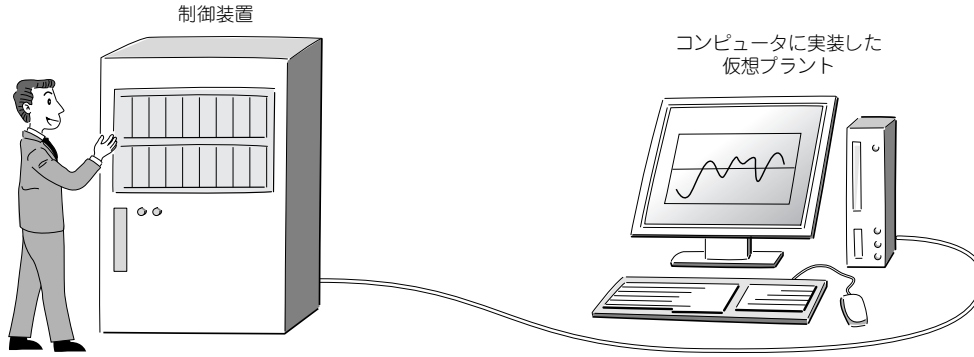

図1.6　HILSの構成

モデル・ベース開発は，HILSとして使うこともできるわけです．

■ 1.2　モデル・ベース開発の導入

モデル・ベース開発は，コンピュータのディスプレイ上にスキーマを描き，これを使ってシミュレーションを行い，その結果をダイレクトにプログラムに変換し，実機実験の実施を可能にします．

モデル・ベース開発は，コンピュータを用いて機械の制御装置の開発工程を上流から下流まで一元化し，開発期間を短縮し，かつ設計変更を容易にします．

しかし，モデル・ベース開発を導入するために，いくつかの条件があります．それらの諸条件について述べます．

モデル・ベース開発のスタート点は，コンピュータのディスプレイ上に描くスキーマです．スキーマは，2次元ディスプレイ上に展開したグラフです．

スキーマのブロックとしてUML（Unified Modeling Language）を使用すべきだ，という意見があります．UMLはグラフ表示を行うための標準言語である，というのです．これは明らかにまちがいです．

UMLのUは英単語のunifiedの頭文字であり，その意味は，複数の言語をまとめて一つの言語にしたという意味であり，具体的にいえば，いくつかのモデリング言語を組み合わせて一つの表記法を作ったという意味です．「標準（standard）」という意味は，どこにもありません．

モデル・ベース開発で使用するブロックは，直接シミュレーションを実行し，かつコンピュータ言語へ変換できるものでなければなりません．これが絶対の条件です．

例えば，制御工学で使用する伝達関数は，

$$G(s)=\frac{a_m s^m+\cdots+a_1 s+a_0}{b_n s^n+\cdots+b_1 s+b_0}$$

という形式になります．

伝達関数の内容は，分子と分母の多項式の係数を指定すれば，一意に決まります．係数の，

$a_m, a_{m-1}, \cdots, a_1, a_0 \quad b_n, b_{n-1}, \cdots, b_1, b_0$

をプロパティとして指定すれば，伝達関数の内容は厳密に決まります．曖昧性は一つもありません．

したがって，伝達関数は，モデル・ベース開発のブロックとして採用することができます．

UMLは，システム分析を行うような漠然とした状況において使うものであり，モデル・ベース開発などのように，数学的に厳密な構造を必要とする状況において使用するべきものではありません．

モデル・ベース開発において，ディスプレイ上にスキーマを描くと，シミュレーションが可能になり，プログラムの自動生成が可能になります．

ここにも注意点があります．ディスプレイ上のスキーマは，本質的に2次元平面上のグラフです．ところで，コンピュータのプログラムは，本質的に1次元上に配置する文字列です．2次元を1次元に変換する過程を，一般にシリアライズ(serialize)と呼びます．

モデル・ベース開発におけるシミュレーション，およびプログラムの自動生成において，スキーマのシリアライズが必要になります．2次元に展開されているグラフを1次元の文字列へ変換する一般的な方法はありません．それが可能という証明もありません．

数学的にいうと，任意の2次元グラフをシリアライズする一般的なアルゴリズムは存在しないので，通常は適当な選択基準を作り，それによって，便宜的に2次元のスキーマを1次元のプログラムに展開します．

すなわち，選択基準の選択によって，シリアライズの結果は異なるものになります．

例えば，**図1.7**に示す伝達関数のブロックがあったとします．

このブロックにおいて，入力を与えれば，出力は計算できます．**図1.8**に示すように，ブロックに対してフィードバック回路がついたとします．

ブロックの入力は，同じブロックの出力を含みます．このブロックの計算は，単純ではありません．ちょっと厄介です．入力の値が決まれば出力が計算できますが，その出力値が入力になるのです．つまり，蛇が自分の尻尾を飲み込むような形式になります．

また，ここでは深く追求しませんが，2次元のチャート上に，デッド・ロック(dead-lock)と呼ばれる図形を描くこともできます．このような図形に関して，シリアライズは正常に行われないので，シミュレーションの結果が信頼できないものになる恐れがあります(**図1.9**)．

このような場合には，人が介入してモデルの図形を別の図形に書き直したりします．

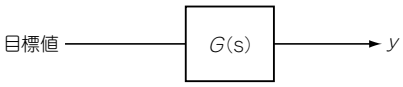

図1.7　ブロック線図

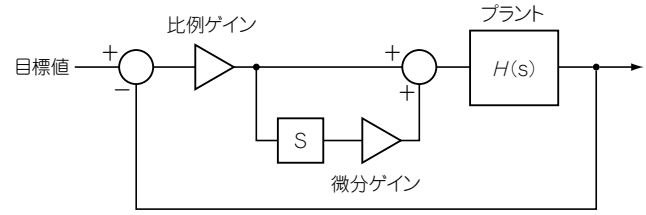

図1.8　フィードバックつきのブロック線図

モデル・ベース開発を導入する際に，人の性格が関係する場合があります．

例えば，A君は組み込みプログラムの技術者の中でも「何でも自分でやらないと気がすまない」というタイプです．本章の冒頭において，CPUとパソコンのBIOSの構造に関するスキットを述べました．このスキットに当てはめると，パソコンを使って制御実験を行う際に，パソコンのBIOSを完全に理解しないとプログラムの作成を始めることができないというタイプの人です．

A君は，自分の技術のレパートリを拡張することを過大に評価し，かつ自分の人件費を過小に評価します．USBを使う前にUSBの規格を徹底的に勉強し，LANを構築する際にはTCP/IPのプロトコル・スタックを作ったりします（**図1.10**）．

A君のようなタイプの技術者を職人気質と呼びます．職人気質の組み込みプログラマは，多くの場合，モデル・ベース開発の導入に抵抗します．「何でも自分でやらないと気がすまない」ので，開発システムのブラック・ボックスが容認できないのです．

また，このような気質の技術者は，開発する制御システムの規模が小さい間は問題を起こさないけれども，開発するシステムの規模が大きくなるにしたがって，システム開発の障害になる場合があります．細部にこだわり，全体を見ない傾向にあるからです．

モデル・ベース開発の導入によって，このようなタイプのプログラマはプロジェクトから排除されることになります．

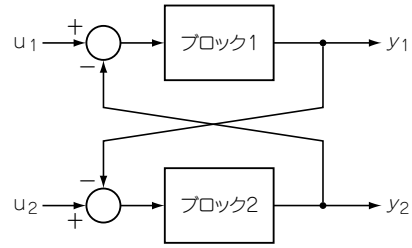

図1.9　デッドロックを含むブロック線図

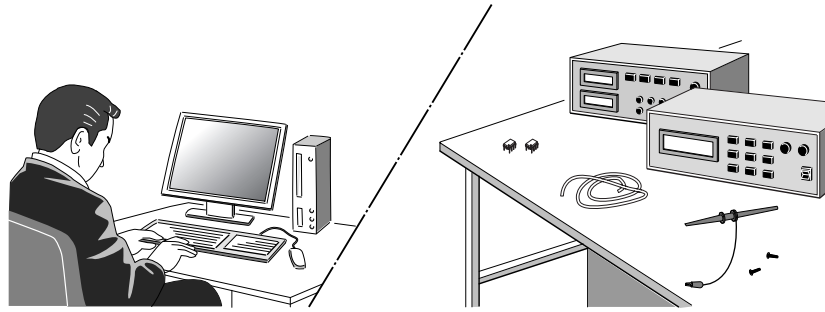

図1.10　職人気質のプログラマ