ソフトウェア工学　83－18
（１９９２．２．７）

オブジェクト指向と状態遷移モデルによるシーケンス制御用言語

滝田啓司　酒井充　米田政明　長谷博行　松田晃典

富山大学工学部

現在、シーケンス制御のプログラミングにはラダーダイアグラム方式が広く用いられている。しかし、この方式はプログラムの構造化、可読性、保守性などに問題を持っている。そこで我々はオブジェクト指向と状態遷移モデルに基づく新しい言語を作成した。制御対象を階層構造をなすオブジェクトの集合とみなすことでプログラムの構造化が可能となり、制御対象の動作を状態遷移を枠組みとして記述することでプログラムの可読性が増した。

本稿では、この言語の文法とインプリメンテーションについて説明する。例として自動サイロシステムに適用した結果、本言語の有効性が確認された。

A Sequence Control Language Based on Object-Oriented Scheme and State-Transition Model

Keiji Takida　Mitsuru Sakai　Masaaki Yoneda　Hiroyuki Hase　Akinori Matsuda

Faculty of Engineering
Toyama University
3190 Gofuku, Toyama-shi, Toyama 930, Japan

A ladder diagram language is widely used for sequence control programming. But it has such defects that (1) lack in modularizability and readability of a program, and (2) difficulty in behavior analysis and modification of a program. So we designed a new language for better sequence control programming. Our language was designed using (1) object-oriented scheme to describe the structure of controled objects, and (2) state-transition model to describe the behavior of controled objects. We consider that our language overcomes the defects which a ladder diagram language has. In this paper we describe syntax and implementation of our language briefly, and show an example for an automatic silo system.

1　はじめに

近年、工場の自動化が進み人手不足を補うようにロボットや自動機械が導入され、自動化が進んでいる。しかし、その自動化された機械を制御するコンピュータ（シーケンスコントローラ）をプログラミングする人の不足はますます大きくなっている。

シーケンスコントローラはラダー図によるラダープログラミングが主であるが、これはかつてはリレーとコイルによってシーケンサが組み立てられていたころからのもので今のプログラミング環境のように構造化、ブロック化されておらず、その表現方法もけっして分かりやすいとはいえない。

そこで我々は、制御対象の構造化のためにオブジェクト指向の考え方を取り入れ、そしてその状態を容易に把握できるよう状態記述を中心とした言語を考案した。部品をオブジェクトと考えることで構造化を行い、状態記述によるプログラミングで、制御対象がどのような状態の時どのような条件で遷移するかがはっきりと記述できるようにした。

また並列処理を考慮にいれ、並行並列、多重並列（パイプライン処理）を簡単に記述できるようにした。

これらの考え方をもとに、例として自動サイロシステムを記述し、それによる評価を行う。

2　シーケンス制御

「シーケンス制御とはあらかじめ定められた順序または一定の論理に従って制御の各段階を逐次進めていく制御である。」（ JIS-C-0401 ）

現在シーケンス制御のプログラミングに使用されている言語は主にラダー方式であるが、前述のように、ラダー方式はかつてリレー制御によってシーケンス制御がプログラミングされていた頃のリレー回路図に対応するものである。しかし現在ではリレーの代わりに PC （プログラマブルコントローラ）といわれる１種のコンピュータが使用されるようになっている。

この PC はコンピュータであるから当然機械語で動作するが、そのプログラミングに今でもダラー方式が用いられているのである。

以下にラダー図の例をあげる。

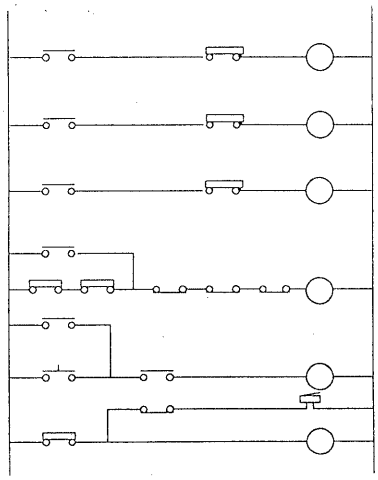


図1　ラダー図

ラダー図によるシーケンスプログラミングは１種の図形プログラミングと言えるが、

- プログラムを構造化できない。このため大規模なもののプログラミングが困難。
- 状態の推移が明示的に表れない。プログラムの動作解析が困難。
- 他人が書いたプログラムを理解することが困難。プログラムを修正することも難しい。

など、問題点が多い。

2．1　自動サイロシステム

シーケンス制御の制御対象の例として自動サイロシステムを取り上げ、それを図２に示す。動作の概要は次の通りである。

原料サイロに貯蔵された２種類の原料から一定量ずつをホッパに貯え、それをミキサに落とし、それらを混合した材料を貯蔵サイロに貯蔵する。

（１）　初期状態

原料ａとｂがそれぞれ原料サイロＡとＢに入っている。

（２）　配合

スタートスイッチが押されると原料サイロＡとＢから原料がそれぞれホッパＡとＢに規

定量になるまで落とされる。

（3） 充填

ホッパAとBに貯められた原料をそれぞれのバルブを開いてミキサに落とす。［この（2）、（3）の状態において、サイロA、サイロBの動作は並行な並列動作をする。両方とも原料をミキサに落とし終えると次の状態に移る。］

（4） 混合

ミキサで一定時間それらを混合する。

（5） 排出開始

混合された材料をサイロ選択スイッチで選択されている方の貯蔵サイロへ排出し始める。

（6） 排出中

排出はミキサの中の混合された材料がなくなるまで続けられる。

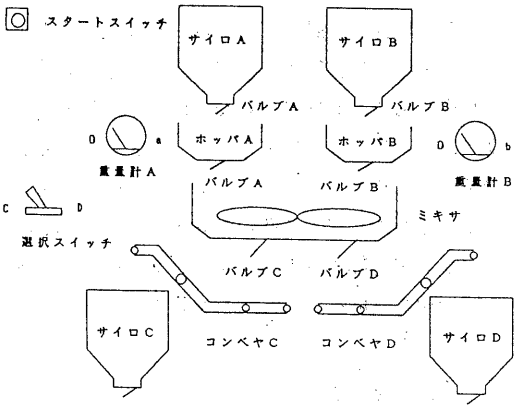


図2 自動サイロシステム

オブジェクト固有のデータと、そのオブジェクトを扱うアルゴリズム（メソッド）を1つにまとめることによるオブジェクトのカプセル化を行うことで、プログラムの記述性を向上し、保守、拡張、再利用の可能性を高めることができる。

本稿では、制御対象となる機械やシステムをオブジェクトとして捉える。またそれらを構成する動作部品などもオブジェクトとして捉える。それらの部品はさらに内部に部品を持っているので、制御対象はそれらのオブジェクトの包含構造（階層構造）となる。

自動サイロシステムは図3のような階層構造を持つオブジェクトとして捉えることができる。

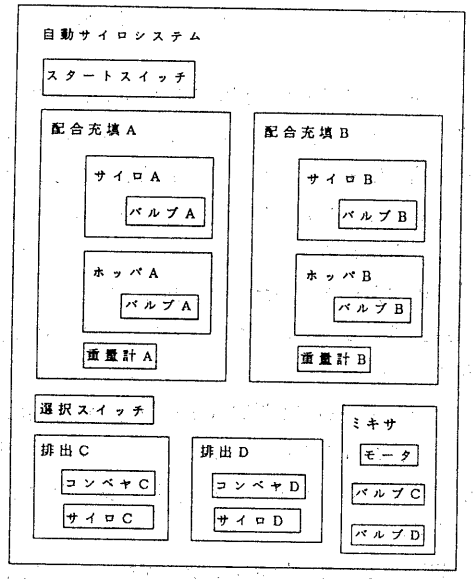


図3 自動サイロシステムのオブジェクト階層構造

3 オブジェクト指向

オブジェクト指向は、80年代からのプログラミングパラダイムであり、今後のプログラミング方式の主流となる考え方であるといえる。

オブジェクト指向をあらわすキーワードには

・抽象化とカプセル化

・インヘリタンス（継承）

などがある。

対象となる物をオブジェクトとして捉え、そのこのように機械の構成単位をそれぞれプログラム単位であるオブジェクトとして捉えることができるので、対象を自然にモデル化しているといえる。

なお、本稿で報告する言語の特徴である状態遷移をプログラムに記述することは、インヘリタンスとはあまり相性がよくない。そのため現段階ではまだ継承を考慮していない。

4　本言語の特徴

本稿で報告する言語は、ラダー方式にかわるシーケンス制御用言語に要求される

・構造化が可能であること
・モジュール化・部品化が可能であること
・シーケンスの状態遷移が的確に表現できること
・可読性に富んで、保守・修正が容易であること

などの要件を満たすように設計した。以下にその特徴を説明する。

4．1　オブジェクト指向

本言語ではオブジェクト指向を取り入れたことで、構造化、モジュール化、部品化が可能となった。

一般にオブジェクト指向では、オブジェクトやクラス、インスタンスなどの用語が使われるが、本言語ではオブジェクトとは部品のような実体をさすもので、具体的なポート番号を持つスイッチなどを意味する。クラスとはオブジェクトが持つ共通の性質をまとめたものであり、プログラムの設計図である。インスタンスとはクラスの中で定義される仮のオブジェクトである。

シーケンス制御の対象は多くの場合物理的にブロック化されており、そのままオブジェクトの単位になるので、オブジェクト指向に適した対象といえる。

前述のように、制御対象はオブジェクトの階層構造を持つとみなすことができるが、オブジェクトは具体的な実体であるので、プログラムの動作中に増えたり減ったりはしない静的なものとして扱う。

4．2　状態遷移モデル

本言語では状態に着目することにより、各オブジェクトの動きを事象駆動型による内部状態の変化で記述でき、時間推移による挙動分類が可能となった。

制御対象としてのオブジェクトは多くの場合内部状態を持ち、メッセージを受けると対応する動作を行い、必要に応じて内部状態を変えると考えることができる。通常のオブジェクト指向言語ではこの内部状態は明示的になっておらず、各メッセージに対する動作記述の中に暗黙的に存在している。しかし制御の分野では、分かりやすさのために状態遷移図を用いて動作記述をすることが多い。そこで、本言語では状態遷移モデルによって個々のオブジェクトの動作記述の枠組みを与えている。

4．3　オブジェクトの動作

本言語ではオブジェクトの動作を上で述べたように状態遷移モデルによって記述した。

オブジェクトの動作には状態に依存する動作と状態によらない動作がある。

状態によらない動作としては関数型メッセージに対する処理がある。

状態に依存する動作としては、その状態に入ったとき1度だけ実行される無条件動作と、条件付き動作がある。条件付き動作部には条件部と実行部があり、条件部ではメッセージが送られてくるのを待ったり、内部オブジェクトからの返事をチェックしたりして、条件が成立すれば実行部を実行する。実行部には内部オブジェクトへのメッセージや次の状態への遷移などを記述する。

4．4　メッセージ

各オブジェクトはメッセージ通信を行う。本言語ではメッセージの種類を以下のように分類した。

・関数型メッセージ

状態によらず受け付け、通常の関数のように値を返す。しかしこれは状態に関してはなんら変化を与えない。

・手続き型メッセージ

状態に依存し、そのオブジェクトの状態を遷移させるトリガーとなるメッセージである。

・自己発信型メッセージ

スイッチが押されたときなど、スイッチオブジェクトが自分を使用している上位オブジェクトに、

「今スイッチが押されました」というメッセージを送ることができれば分かりやすい。このようなメッセージを自己発信型メッセージという。これにより下位のレベルの部品オブジェクトなどにおいて、自分の状態が変化したとき上位のレベルのオブジェクトにメッセージを送ることができるので、プログラムで事象駆動的な記述を行うことが可能となり、上位のオブジェクトがたえず下位のオブジェクトの状態を監視する必要がなくなり、１スキャンにおいてチェックすべき項目が大幅に減ることが予想される。

４．５　並列処理

シーケンス制御においては多くの場合並列処理動作が生じる。代表的な並列処理には２つ以上のオブジェクトが独立に動作する並行並列処理といわゆるパイプライン処理がある。

本言語では、各オブジェクトは独立に定義されているので、基本的に各オブジェクト間の並行並列動作が可能である。各オブジェクトの同期はメッセージ通信で行うこととし、オブジェクト間のメッセージ通信ができるのはオブジェクト階層の階層間だけとした。

４．５．１　並行並列処理

上述のように並行並列処理は容易に記述できた。ただし同期をとる必要があるので、並行並列動作するオブジェクトを含むオブジェクト（上位オブジェクト）でその制御を行う必要がある。

４．５．２　パイプライン処理

本言語では以下のような状態の階層化による拡張で、パイプライン処理を記述できるようにした。

状態の階層化は、１つの状態（主状態）がさらに細かい１つの状態遷移（副状態遷移）を持つことを許すことにより実現する。基本的には主状態と副状態遷移の両方が同時に活性化する考える。

パイプライン処理の部分を PIPELINE というキーワードを付けて副状態遷移として記述する。この副状態遷移にはループの戻りは記述しない。各状態がアクティブになれるが、最初の状態から順にアクティブになるようにする必要がある。そのため、副状態遷移の初期状態を常にアクティブである特別な状態とする。また、一連の動作を終了するには特別なコマンド EXIT を用い、最終状態を用いない。

パイプライン処理全体を終了させるにはいくつかの方法が考えられるが、ここでは副状態遷移の初期状態で EXIT コマンドを用いるとそれが不活性となり、パイプライン部に活性状態がもう増えないことで、パイプライン部は最終的には終了し、最後の EXIT コマンドでパイプライン処理全体が終了するようにした。

状態遷移では基本的には１つの状態しかとれない。複数の状態をとるというのは本来はありえない。しかし、現実には、１つの状態ではそれに関して制御される機械は一部の機械だけであり、それらの機械は他の状態で制御される機械とまったく異なる場合がある。このような場合にはそれらの状態を同時にとることが可能となる。パイプライン処理の記述はこのような場合に機械資源を有効に利用するためのものである。

パイプライン処理（副状態遷移）の各状態間で同時に活性化してほしくない状態がある。それはプログラム中の条件の一部として記述することもできるが、ここでは次のような宣言的な記述により実現した。

```
EXCLUSIVE (n1,n2),(n3,n4,n5);
```

n1とn2は同時にアクティブにはなれない排他状態である。同様にn3かn4かn5のどれか１つのみアクティブになれる。しかしこの時n1とn3は同時にアクティブになれる。

パイプライン処理は、ペトリネット的動作であり、また排他制御を行うので、デッドロックを生じる可能性がある。しかし本言語のパイプライン処理は局所的であり、記述の全体にペトリネットを利用している言語に比べ、デッドロックの計算量が少なくてすむと考えられる。

5　本言語の文法

本言語の特徴は、今まで述べてきたように、クラスによるオブジェクトの部品化、状態遷移記述による部品の動作の記述などであるが、以下に本言語の文法について述べる。

5．1　構造

プログラムは

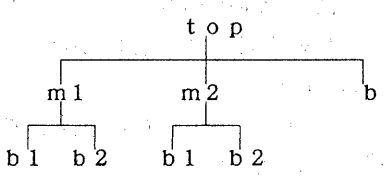
```
CLASS bottom {
        ・・・
}
CLASS middle {
        ・・・
}
CLASS top {
        ・・・
}
```

のようにクラスの集合として記述され、表記上クラスの階層構造は現われない。しかし、

```
CLASS bottom {
        ・・・
}
CLASS middle {
        INSTANCE bottom b1,b2;
        ・・・
}
CLASS top {
        INSTANCE middle m1,m2;
        INSTANCE bottom b;
}
```

のようにクラスの内部で使うクラスのインスタンスを宣言することで、クラスの階層関係を記述することができる。

上の例において、この top クラスからオブジェクトを生成すると、以下のようなオブジェクトの階層関係を持つプログラムとなる。



各クラスは以下のように記述される。

```
CLASS       class_name {
    VAR     type    name;
    PARAMETER       type    param;
    INSTANCE class_name instance_name;
    EXPORT  type    function_name();
    MESSAGE message_name;
    ・・・
    STATE_TRANSITION name {
        NODE node_name1 {
            単純動作部
            条件動作部 {
                ・・・
                TRANSITION nodename;
            }
        }
        NODE node_name2 {
            ・・・
        }
        ・・・
    }
    FUNCTION type function_name() {
            ・・・
    }
}
```

1つのクラスは「CLASS クラス名 {」で始まり「}」で終わる。また大文字で書かれたものはキーワードである。クラスのプログラミングは大きく分けて2つの部分からなる。

1つは、クラス宣言部。もう1つはクラス動作記述部である。

クラス宣言部は、使用される変数、オブジェクト固有のパラメータ、内部のオブジェクトのインスタンス、公開するファンクション（メソッド）、受け付けるメッセージが宣言される。この部分はクラスのプロトタイプ的な部分と言える。

もう1つのクラス動作部がクラスの本体である。この部分も2つの部分に分けられる。1つは状態遷移記述部、もう1つはファンクション定義部である。

状態遷移記述部は、本言語の特徴でもある状態遷移をそのまま記述する部分である。各ノード（状態）を記述し、そのノードにおいて実行される動作部を記述する。単純動作部はそのノードに遷移したとき1度だけ実行される部分である。条件動作部は、次のノードに遷移するための条件と遷移先などが記述される。

このノード内の動作部において内部のオブジェクトへメッセージを送ったり、ファンクションをコールしたりする。

各オブジェクトにおいて現在活性であるノードの実行部だけが実行され、他のノードの実行部は実行されない。

ファンクション定義部では、オブジェクトがどのノードにいても実行できるファンクションが定義される。

メッセージ送信については、

instance_name.message(selectors);

と記述することでどのオブジェクト（ instance_name ）にどんなメッセージ（ message(引数 selectoer) ）を送るかを指示する。

関数型メッセージについては、受信したオブジェクトはいつでもこれを受け付け、そして値を返すので、送信オブジェクトは、

WHEN（メッセージ送信）｛条件動作部｝

のように使用することで返り値を利用することができる。

手続き型、自己発信型メッセージは受信するオブジェクトにおいて、

ACCEPT メッセージ ｛条件動作部｝

と受け付け得るノードに記述し、その状態においてメッセージを受信することができる。

5．2 インプリメンテーション

本言語の動作は、上記のソースコードをコンパイルし、C言語に落とし、そのCソースをコンパイルした実行可能なプログラムによってシミュレートされる。

Cのソースに落とされたクラスは構造体宣言となりオブジェクトは静的でグローバルな構造体変数として定義されている。これは 4.1 でも述べたように、シーケンス制御においてオブジェクトの対象である機械などはプログラム実行中に増えたり減ったりしないからである。

コンパイルされて出てきたCのソースのオブジェクトの固有のパラメータを設定することで、同じクラスからつくられたオブジェクトでも異なる計算を行わせることが可能である。

コンパイラについては、パソコン（ EPSON PC-386 ）上の bison （ yacc ）で構文解析部を、jlex （ lex ）で字句解析部を作成し、コード生成部はC言語（ Turbo-C ）を使用して作成した。

6 自動サイロシステムのプログラミング例

以下に自動サイロシステムのクラスのプログラミングの例（スイッチクラスとトップクラス）をあげる。

```
CLASS swit
{
    INSTANCE    inport    i;
    EXPORT    BOOL on();
    STATE_TRANSITION {
        NODE off {
            WHEN (i.in() == ON) {
                TRANSITION on;
            }
        }
        NODE on {
            WHEN (i.in() == OFF) {
                TRANSITION off;
            }
        }
    }
    FUNCTION
        BOOL on() {
            IF (NODE == on) RETURN TRUE;
            ELSE            RETURN FALSE;
    }
}
```

スイッチのクラスのプログラム例

```
CLASS  auto_silo
{
    INSTANCE    swit        startsw;
    INSTANCE    haigou_jyuuten    hja, hjb;
    INSTANCE    mixer        mix;
    INSTANCE    selectsw    sel;
    INSTANCE    drop        dc, dd;
    /* 副状態遷移部 */
    PIPELINE STATE_TRANSITION tajyuu {
      EXITEVENT  end;
      EXCLUSIVE  (haigou,jyuuten),
                 (jyuuten,mix,/*…*/,drop_end);
      NODE init {
          WHEN ((hja.isstartable() == TRUE) &&
                (hjb.isstartable() == TRUE))
              TRANSITION haigou;
          WHEN (startsw.on() == FALSE)
              EXIT;
      }
      NODE haigou {
          hja.haigou_start(); hjb.haigou_start();
          ACCEPT (haigou_end|hja && haigou_end|hjb)
              TRANSITION jyuuten;
      }
      /* 中略 */
      NODE drop_end {
          ACCEPT end|mix {
              mix.to_init();
              EXIT end;
          }
      }
    }
    /* 主状態遷移部 */
    STATE_TRANSITION main {
        NODE start {
            WHEN (startsw.on() == TRUE)
                TRANSITION pipeline_node;
        }
        NODE pipeline_node {
            STATE_TRANSITION tajyuu;
            ACCEPT    end|tajyuu
                TRANSITION start;
        }
    }
}
```

トップクラスのプログラム例

各オブジェクトのクラスについて、状態遷移を記述したプログラミングを行う。本言語で自動サイロシステムを記述すると、オブジェクト数は52個であるが、クラス18個分（部品化されている低レベルのクラス（ポートやタイマーなど）５つを含むので実際は13個分）の記述でプログラミングできた。

7　シミュレーション結果

上記の自動サイロシステムについてコンパイルして得られたCのソースにパラメータを設定して実行した結果を解析すると以下のようになる。

・０サイクル目

すべてのオブジェクトは初期状態にある。

・１サイクル目

スタートスイッチの状態により、トップの状態が遷移する。

（ONとみなし、トップの状態は配合状態へ）

・２サイクル目

配合充填オブジェクトａ，ｂに start のメッセージを送る。

配合充填オブジェクトは自分の内部のバルブオブジェクトに open のメッセージを送り、その結果バルブがオープンされる。

・・・・・

以下次々にメッセージが送られ、プログラムは実行されていく。

このシミュレーションにより所期の目的は達成されたが、最初の１サイクルでスタートスイッチが押されてからバルブが開くまで遅れが生じる。これは状態が１つ遷移するのに１サイクルかかるためであり、状態が細かく分かれているほど、状態遷移のための時間がかかることになる。

8　まとめ

オブジェクト指向と状態遷移記述を取り入れたことで、シーケンス制御のプログラミングを容易に行うことのできる言語を作成した。

本言語を用いれば、機械をブロック化でき、部品の再利用も可能となり、また機械の動作が把握しやすいので、わかりやすいプログラミングを行うことができるようになったと考えている。

また基本的にオブジェクトの独立性による並行処理記述が可能であり、また複雑にならないレベルで高度な並行処理記述能力を追加できた。

今後の課題としては、本言語を用いて具体的に種々の制御対象を記述し、その動作について調べ、本言語の記述能力の確認を行うことがあげられる。また、インタラクティブなプログラミングが行えるようなエディタや、容易に動作解析ができるように、タイムチャート出力や状態遷移が把握できるデバッガなどの開発を行い、環境を整備する必要がある。

さらに、並列動作システムに起こりうるデッドロックなどを検出できるように、コンパイラの機能を強化することも必要である。

謝辞

本研究を進めるに当り多大なご協力をいただいている立山マシン株式会社高口彰氏並びに株式会社立山システム研究所能島信行氏に謝意を表します。

［参考文献］

[1]電気学会:"シーケンス制御工学 新しい理論と設計法",オーム社

[2]木村陽一:"シーケンサ活用テクニック"、電気書院

[3]岸、岡安、平山、三原:"状態遷移に基づくプログラム部品合成システムの開発"、情報処理学会研究会報告、91-SE-80-22,PP.167-174

[4]福井眞吾:"並列システムの階層的記述に適した通信機構"、情報処理学会研究会報告、91-PRG-3,PP.7-16

[5] 佐野佳克："オブジェクト指向シーケンス制御言語の試作"、富山大学大学院工学研究科修士論文,(1989)

[6]酒井、米田、長谷、滝田、松田:"オブジェクト指向と状態遷移モデルに基づくシーケンス制御用言語の試作"、情報処理学会第43回全国大会予稿