プログラミング-言語・基礎・実践- 21－11
(1995. 3. 9)

オブジェクト指向に基づくシーケンス制御用言語とIL言語へのコンパイラの作成

酒井貞亮　　酒井充　　米田政明　　長谷博行

富山大学工学部電子情報工学科
〒930 富山市五福 3190

あらまし　シーケンス制御用言語として広く用いられているラダーダイアグラム方式には「プログラムの可読性が乏しい」、「プログラムをモジュール化できない」など様々な問題点が存在する。そこで我々の研究室ではオブジェクト指向と状態遷移図モデルを採用した新しいシーケンス制御用言語を開発した。本稿では本言語の基本概念と本言語により記述されたプログラムをプログラマブルコントローラと呼ばれる制御装置で採用されているIL言語に変換するコンパイラについて説明する。

キーワード　シーケンス制御、IL言語、オブジェクト指向、状態遷移図モデル、コンパイラ

An Object Oriented Sequence Control Language and Implementation of the Compiler to IL Language

Sadaaki Sakai　　Mitsuru Sakai　　Masaaki Yoneda　　Hiroyuki Hase

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

Abstract　In the field of sequence control language, ladder diagram is in general use. But it has some problems, for example, lack of readability, modularity, etc. Therefore, we have designed a new sequence control language on the concept of the object oriented theory and state transition diagram model. In this paper, we explain the basic concept of our language and the compiler which translates the language into IL language used in general programmable controllers.

key words　sequence control, IL language, object oriented, state transition diagram model, compiler

1.はじめに

シーケンス制御は古くからある技術であり、生産現場においてはフィードバック制御に劣らぬ重要な技術である。従来のシーケンス制御はリレー制御盤を使用していたが、現在ではコンピュータ技術を導入したプログラマブルコントローラ（以降、PC）を用いるようになった。

PC を用いて機械を制御するシステムの製作にあたっては、ハードウェアの設計・製作とともに、ソフトウェアの設計・製作も重要な作業である。ソフトウェアを記述するための言語にはフローチャートによる記述、論理回路図による記述、テキストによる記述などがある。これらの中で、最も広く利用されている方式は論理回路図の一種であるラダー図である。

ラダー図は記述が特殊であるため専門家以外には分かりにくく、モジュール化ができない、デッドロックの検出が困難であるなどの欠点を持っている。これらの欠点を改善するため、最近ではペトリネットを基本としたグラフセ（GRAFCET）や SFC などが開発され使用されている。

本研究室では、それらと同様、ラダー図の持つ欠点を解決する目的でオブジェクト指向と状態遷移図モデルを導入した新しいシーケンス制御用言語（以降、本言語）を設計した。本言語では並行動作記述にオブジェクト間の並行性を利用し、各オブジェクトの自律的動作記述に、基本的には並行動作の無い状態遷移図の枠組みを利用することにより、非同期な並列動作の記述やモジュール化、可読性のよさを実現している。

本稿では本言語の設計概念と本言語に基づき記述されたプログラムを PC で用いられる命令リスト（以降、IL）へ変換するコンパイラについて述べる。

2.シーケンス制御

シーケンスとは現象の起こる順序をいい、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よって定められる順序にしたがって、制御の各段階を逐次進めていく制御である（JIS C 0401）。

本節ではシーケンス制御に用いられる PC、ラダー図について簡単に説明する。

2.1.プログラマブルコントローラ

シーケンス制御に主として使用されている PC について簡単に説明する。PC とは入出力部を介して各種装置を制御するものであり、プログラマブルな命令を記憶するためのメモリを内蔵した電子装置である。PC はシーケンス制御用の特殊な CPU であり、一般的な CPU とはかなり異なる動作形態を持つ。

PC の内部命令は多くの場合 IL で記述される。IL の命令は、ロード（LD）、AND、OR などの機能を示す部分（命令語）と、入力リレー、出力リレーなどの要素番号よりなる。IL によるプログラムはこのような命令を実行する順序に配列した一連のリストで構成される。

PC はステップ番号 0 から順に命令の内容を実行し、プログラムの最後までいくと再びステップ番号 0 へ戻る動作をする。このような演算をサイクリック演算と呼び、ステップ番号 0 からプログラムの最後までの実行を PC の 1 サイクルと呼ぶ。

ステップ番号（アドレス）	命令	
	命令語（オペコード）	要素番号（オペランド）
0	LD	X000
1	ANI	M7
2	OUT	T0
	SP	K20
5	LD	T0
6	OR	M0
7	ANI	M7
·	·	·
·	·	·
·	·	·
·	END	
1999		

サイクリック演算

要素番号は不要

図 2-1 IL（命令リスト）

2.2.ラダー図

現在シーケンス制御用言語には、主にラダー図が採用されている。それは、かつて使用されていた表現方法である電磁リレーの展開接続図によく似ているからである。

ラダー図とは、接点とコイルを一定の規則に従って接続、配置したものであり、IL 命令とほぼ 1 対 1 に対応している。

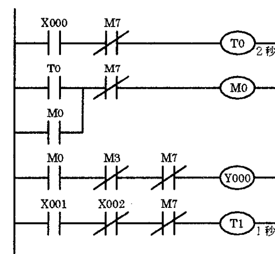

図 2-2 ラダー図

3.本言語の基本概念

本言語の特徴はオブジェクト指向と状態遷移モデルを採用していることである。本節ではそれらを中心に本言語に基本概念を説明する。制御対象の具体例として自動サイロシステムを取り上げる。

3.1.自動サイロシステム

自動サイロシステムとは、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自動化システムである。

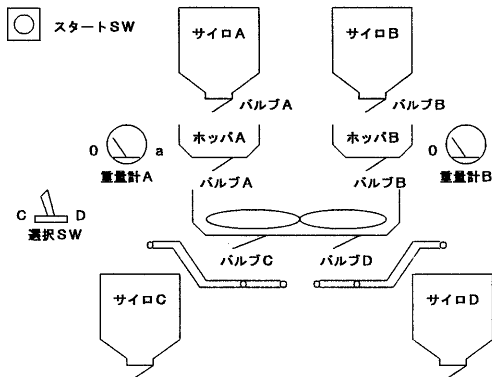

図 3-1 自動サイロシステム

3.2.オブジェクト指向

シーケンス制御の制御対象は、システム全体を1つのオブジェクトとして捉え、またそれを構成する動作部品、更にその内部の部品などをオブジェクトとして捉えていくことにより、オブジェクトの階層構造として捉えることができる。それらのオブジェクトは具体的な実体であるため、プログラムの動作中に増えたり減ったりしない静的なものとして取り扱う。このようにして対象システムの構成単位である機械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、対象を自然な形でモデル化することができる。

自動サイロシステムは次の図のような階層構造として捉えることができる。

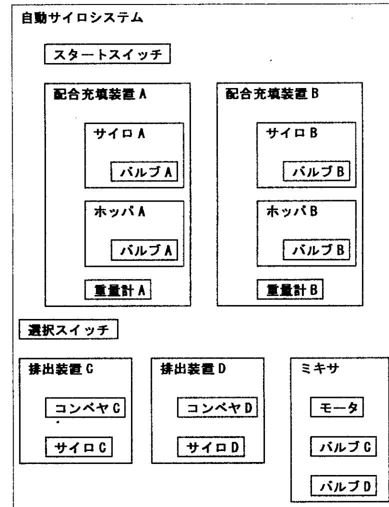

図 3-2 オブジェクトの階層構造

3.3.状態遷移図モデル

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し、時間推移による挙動分類が可能となった。次の図は自動サイロシステムのバルブにおける状態遷移図である。

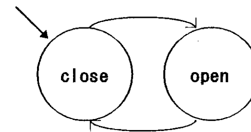

図 3-3 バルブの状態遷移図

制御対象に対応するオブジェクトの多くは内部状態を持ち、メッセージを受け取るとそれに対応する動作を行い、必要に応じて内部状態を変える。通常のオブジェクト指向型言語ではこのような内部状態は明示的にはなっておらず、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行うことが多い。そこで、本言語では状態遷移図モデルによって個々のオブジェクトの動作を記述する。

3.4.オブジェクトの動作

シーケンス制御においては、多くの場合、並列処理が生じる。本言語では、それに対応するための並行プログラミングが可能となっている。プログラムは複数のオブジェクトからなり、それぞれのオブジェクトは独立して動作しているものとみなす。シーケンス制御ではオブジェクト間の同期が重要であるが、メッセージ通信でそれを行う。

メッセージには、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と状態に依存せず通常の関数のように値を返すメッセージがある。前者は値を返さないので「手続き型メッセージ」と呼び、また後者は「関数型メッセージ」と呼ぶことにする。

オブジェクトの自律的動作を記述するために状態遷移図の枠組みを用いるが、その遷移のトリガーをここではイベントと呼ぶ。オブジェクトは状態遷移図の中のどれか1つの状態にあり、その状態に記述されたいくつかのイベントの中の発火可能なイベントで、もっとも優先度が高いイベントが発火し、そのイベントに対応した動作をする。

イベントには、上位オブジェクトからのメッセージ、下位オブジェクトからのメッセージ（自己発信型メッセージと呼ぶ）、下位状態遷移図からの内部イベント、WHEN 文による条件式イベントがある。特に上位オブジェクトからのメッセージ以外のイベントに対応した処理は遷移図内でのみ記述する。上位オブジェクトからのメッセージによるイベントに対応した処理は通常のオブジェクト指向言語と同様に遷移図外でも記述できる。

通常のメッセージは送り先のオブジェクト名を指定して送信する。そのため、メッセージはクラス内のインスタンスに対して送信することになる。一方、自己発信型メッセージはそのオブジェクトをインスタンスとして定義しているオブジェクトに対して送信するメッセージである。

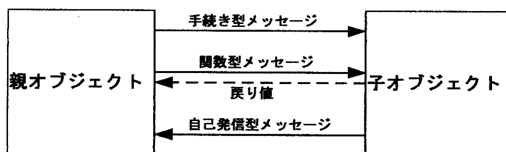

図 3-4　メッセージの種類

また、ある一定の働きをするいくつかの状態からなる状態群が状態遷移図の複数の場所に現れる場合がある。このような場合にはそれらを一つの副状態遷移図と定義して、あたかも状態遷移のサブルーチンのように用いながら状態遷移図を作成していけば、プログラムがコンパクトになり分かりやすくできる。

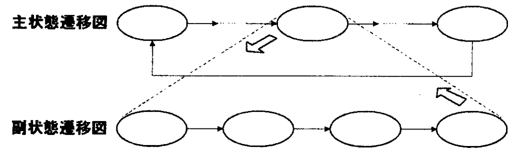

図 3-5　副状態遷移図

3.5.パイプライン処理

パイプライン処理とは、例えば工場のラインにおける流れ作業のように進む処理である。次の図のように、最初は部品を用意し、次にそれを組み立て、そして塗装をするという工程を考える。

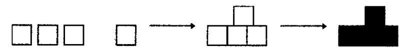

図 3-6　流れ作業の工程

このような工程を繰り返し行う場合、工程に空きができないように次のような処理を行えば、効率よく製品を製造できる。

時間	用 意	組み立て	塗 装	完 成
	材料1			
	材料2	材料1		
	材料3	材料2	材料1	
	材料4	材料3	材料2	製品1
	材料5	材料4	材料3	製品2
	材料6	材料5	材料4	製品3
			・	
			・	

図 3-7　効率のよい処理

制御プログラムを開発する場合、まず逐次的な状態遷移図によって記述できるパイプライン処理の無いシステムを考え、次にそれをパイプライン処理のあるシステムへと拡張する段階的方法が考えられる。このとき、状態遷移図記述を拡張することによりパイプライン処理を記述できれば、パイプライン処理記述が容易になると考えられる。上記のオブジェクトと状態遷移図モデルの枠組みのみでもパイプライン処理の記述は可能ではあるが、パイプラインのない逐次的な状態遷移記述を大きく変更する必要がある場合が多いと予想され、必ずしも容易ではない。そのため本言語では状態遷移図記述を基にパイプライン処理を記述できるようにしている。

パイプライン処理は副状態遷移図の先頭でパイプライン宣言することで定義される。これをパイプライン図と呼ぶことにする。パイプライン宣言以外はほとんど通常の状態遷移図と同様であるため、パイプライン図でも状態という用語を用いる。また、活性化している状態にトークンがあり、状態遷移はトークンの移動で考えるものとする。パイプライン図では、初期状態で次々にトークンが発生し、最終状態でトークンは消滅する。初期状態ではトークンを繰り返し発生させるために、トークンが無くなり次第再びトークンを発生させるものとする。

パイプライン図では、いくつかの状態が同時に活性化できるが、あるトークンが遷移しようとする時、遷移先の状態にトークンが存在する場合がある。この場合は現在の状態に止まり遷移待ちモードに移る。遷移待ちモードは遷移先の状態が空くまで続く。

また、ある状態が活性化している時、他のある状態を活性化したくない場合も考えられる。そのような場合、それらの状態は排他的であると宣言することができる。

3.6.デッドロックへの対処

並行動作システムではデッドロックの防止と検出が重要である。本言語では並行動作を記述するためにオブジェクト間の並行性とオブジェクト内でのパイプライン記述を利用しているが、それらによって生じる可能性のあるデッドロックへの対処について考える。

オブジェクト間の並行性によるデッドロックの原因の一つはオブジェクトが相互に同期メッセージを通信し合うことから生じる。このことは本言語でメッセージのパラメータとしてオブジェクトを使用することを許していない理由の一つである。

次にパイプライン記述によるデッドロックについて説明する。各状態間に排他制御が必要となる場合があるが、これはプログラム動作記述中にその制御を埋め込むことにより実現可能である。ところが、その制御は動作記述の中に埋もれているため、そのことに起因するデッドロックを検出することが難しい。このため、本言語では状態間の排他制御を明示的に宣言できるようにした。これにより、デッドロックの検出が容易になると考えられる。

4.クラスの構成

プログラムの記述はクラス単位に行う。クラス定義の概略は次のとおりである。

```
CLASS クラス名 {
        宣言部
        状態遷移図定義部
        関数定義部
}
```

次に自動サイロシステムにおけるバルブのプログラム例を示す。

```
CLASS valve {
        INSTANCE outport o;
        MESSAGE open();
        MESSAGE close();
        MESSAGE BOOL on();

        STATE_TRANSITION valve {
                NODE close {
                        o.out(FALSE);
                        ACCEPT open() TRANSITION open;
                }
                NODE open {
                        o.out(TRUE);
                        ACCEPT close() TRANSITION close;
                }
        }

        FUNCTION BOOL on() {
                IF(NODE == open) RETURN TRUE;
                ELSE FALSE;
        }
}
```

宣言部

クラスにおいて使用するインスタンス、変数、公開するメッセージなどを宣言する。

状態遷移定義部

本言語の特徴の１つである状態遷移図を中心に動作を定義する部分であり次のように記述する。

```
STATE_TRANSITION 状態遷移図名 {
        宣言部
        状態定義部
}
```

状態定義部は状態群の動作を定義する部分であり、各状態は次のようなブロックで構成される。

```
NODE 状態名 {
        宣言部
        単純動作部
        条件付き動作部
}
```

宣言部はその状態に局所な変数を宣言する。単純動作部とはその状態に遷移したときに１度だけ実行される部分であり、その状態における初期動作を定義する部分である。また、条件付き動作部はその状態が活性化している間有効となるイベントと、それに対する動作を定義する部分である。

またパイプライン宣言はキーワード「PIPELINE」によって行う。

```
PIPELINE STATE_TRANSITION 状態遷移図名 {
        宣言部
        状態定義部
}
```

パイプライン図の宣言部では排他宣言を行うことが可能である。排他宣言は次のような形式によって行う。

```
EXCLUSIVE ( 状態名 , 状態名 [ , 状態名 ・・])
          [ , ( 状態名 , 状態名 [ , 状態名 ・・]) ・・];
```

「(」と「)」の間に列挙された状態群の中の各状態は同時に活性化できない。

また、パイプライン図の状態定義部で最初に定義された状態は初期状態となり、最後に定義した状態は最終状態となる。

関数定義部

オブジェクトがどの状態にあっても実行できる関数を定義する部分である。ただし、再帰的な呼び出しはできない。

```
FUNCTION 型名 関数名 ([ 型名 引数名 [, 型名 引数名 ・・]])
{
        宣言部
        動作定義部
}
```

5.実装

本節では本言語によるプログラムを IL コードに変換するために採用した方法を示す。まず、実装を考える上で問題となった点を説明し、次に変換方法について説明する。

5.1.実装上の問題点

本言語によるプログラムを IL 言語に変換する上でいくつかの問題が存在する。

まず、IL 言語の命令は PC のメーカー、機種によって微妙に異なるため、多くの PC への移植性を重視すると、使用できる命令が制限される。

また、PC では下位ステップへジャンプする命令は備えているが、基本命令として上位ステップへ戻るような命令を備えていない。応用命令として備えていても入れ子にできなかったり、入れ子に制限があったりする。このため、サブルーチンの効率的な実現が困難となっている。

更に、メモリは固定的に割り当てなければならないため、ローカル変数なども固定的にメモリを占有する。

5.2.状態遷移の実現

本言語における状態遷移図には、通常の状態遷移図（主状態遷移図）と任意の状態のサブルーチン的な状態遷移図（副状態遷移図）の２種類があるが、生成されるコードは基本的には同じである。

まず、各状態に対してそれぞれ PC の補助リレーを対応させ、それがオンである時、その状態が活性化しているものとした。状態を遷移させるには、現在活性化している状態に対応する補助リレーをオフにし、遷移先の状態に対応した補助リレーをオンにする。

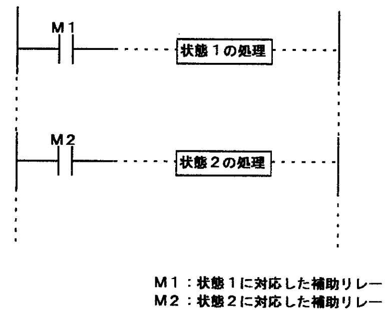

図 5-1　状態遷移の実現

また状態には単純動作部と条件付き動作部が存在する。そのため、単純動作部であることを表すフラグとして補助リレーを用意し、それがオンである時には単純動作部が、オフである時には条件付き動作部が実行されるようにした。

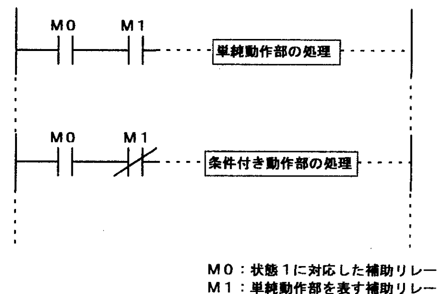

図 5-2　単純動作部と条件付き動作部の区別

副状態遷移図は１つのクラスに複数個定義できるため、それぞれの副状態遷移図に補助リレーを対応させ、それがオンである時、その副状態遷移図が活性化しているものとした。従って、副状態遷移図を活性化させるには、それに対応した補助リレーをオンにすればよいし、副状態遷移図を抜けるにはそれをオフにすればよい。

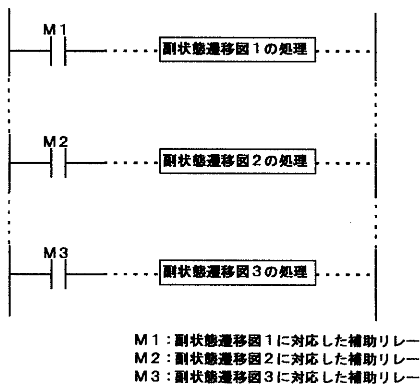

図 5-3　副状態遷移図の区別

5.3.メッセージの実現

本言語では手続き型、自己発信型、関数型の３種類のメッセージが用意されている。以下にその実現方法を示す。

手続き型メッセージ

手続き型メッセージの受信は各メッセージごとに深さ１のバッファを用意している。すなわち各メッセージに対して補助リレーを対応させ、送信時にそのリレーをオンにし動作保持させ、受信時にそれを解除することで実現している。

このような処理を行う場合、メッセージ送信の際に、同一のメッセージが既に送信されており、それがまだ受信されていない場合を考慮する必要がある。その様な場合には、先に送信されたメッセージが受信されるまで送信を待ち、その間他の処理を行わない。

その実現のために、メッセージ送信待ちを表すフラグとして補助リレーを用意する。それがオンであることがメッセージ送信待ちの状態であることを示す。従って、メッセージ送信の際にそのメッセージに対応した補助リレーの状態を確認し、もしオンであればメッセージ送信待ちを表す補助リレーをオンにする。そしてメッセージに対応した補助リレーがオフになり次第メッセージ送信待ちを表す補助リレーをオフにし、メッセージの送信を行う。すなわち、メッセージに対応した補助リレーをオンにする。

自己発信型メッセージ

自己発信型メッセージは手続き型メッセージと同様に、各メッセージに対して補助リレーを対応させ、送信時にオンにする。しかし、手続き型メッセージと異なりその補助リレーの動作保持は行わない。このため、メッセージの存続期間は PC の１サイクルとなる。また、メッセージ送信待ちも行わない。

関数型メッセージ

関数型メッセージは、PC でのサブルーチンの実現が困難であるため、関数自体をインライン展開することにより実現している。

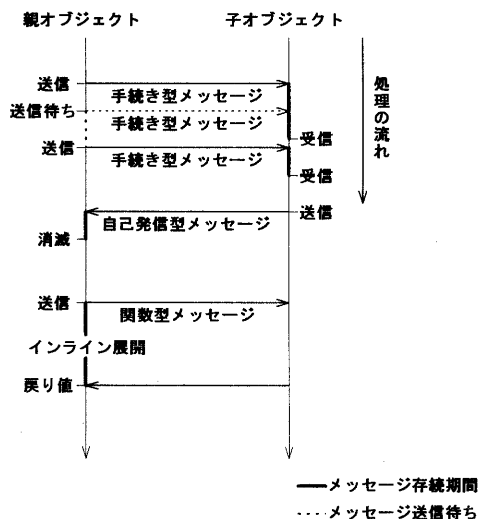

図 5-4　メッセージの送受信

5.4.　パイプライン処理の実現

パイプライン図の初期状態は、トークンを繰り返し発生させるために、不活性になり次第活性化する。すなわち、初期状態に対応した補助リレーはオフになり次第オンになる。

また、遷移待ちモードの実現には、遷移待ちの状態であることを表すために補助リレーを用いる。任意のトークンが状態を遷移させる際には、遷移先の状態に対応した補助リレー及び同時に活性化できない状態が存在するならそれらに対応した補助リレーの状態を調べる。調べた結果、それらが全てオフであれば遷移が行われるが、そうでない場合は、遷移待ちを表す

補助リレーをオンにし動作保持させ、遷移待ちモードに入る。

6.検証

本言語で自動サイロシステムを記述すると、オブジェクトの総数は53個であるが、クラスは16個(クラスライブラリを除くと12個）となった。

本コンパイラを用いて自動サイロシステムのプログラムをILコードに変換し、ILエミュレータを用いてエミュレートした結果、正常に動作することが確認された。

しかし、スタートスイッチが押されてからバルブが開くまでに遅れが生じるなど、入力リレーの変化が直ちに反映されない場面が存在した。

この原因は、本コンパイラでは上位オブジェクトから順に処理されるようにコードを生成するため、あるサイクルにおける下位オブジェクトの状態の変化が上位オブジェクトに伝わるのは、既にそのサイクルでの上位オブジェクトの処理は終了しているため、次のサイクルになるからである。従って、オブジェクトの階層構造が深いほどそのような遅れが大きくなる。PCの1サイクルは数ミリ秒で処理されるため自動サイロシステムではさほど問題にならないが、微妙なタイミングが要求されるシステムにおいては問題となるであろう。

このような遅れを生じさせずに上位オブジェクトに入力リレーの変化を伝える方法としては、関数型メッセージを駆使する方法が考えられる。ただし、関数型メッセージを使用する場合、関数自体はインライン展開されるため生成されるコードの量は増える。

7.まとめ

オブジェクト指向と状態遷移図モデルに基づくシーケンス制御用言語のIL言語へのコンパイラを作成した。このことにより、本言語で作成したプログラムをPCで実行させることが可能となった。

本言語ではオブジェクト指向を取り入れたことにより、システムをモジュール化でき、部品の再利用などが可能となった。また、状態遷移図モデルを導入したことにより、システムの動作を把握しやすくなった。

今後の課題としては、種々の制御対象について本言語により制御プログラムを作成し、今回作成したコンパイラによりILコードを生成してその動作確認を行うことにより、本言語の記述能力や本コンパイラが生成するコードの安定性などを検討することが挙げられる。

また、オブジェクト指向言語には欠かせない継承の枠組みを追加するなどの言語の拡張や、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必要もあるだろう。

本言語や本コンパイラは試作段階であり、今後、言語の変更、拡張、コード生成方法の見直しが行われることが予測されるため最適化を行っていない。また本コンパイラは他のPCへの移植性を重視して作成したため、特定のPCの命令を使った最適化を避けた。そのため、本コンパイラが生成するコードは実際にIL言語で記述した場合に比べ、メモリ効率、実行効率ともかなり悪いものになった。今後、最適化などの処理を施し、効率を向上させなければならないであろう。

参考文献

[1] 電気学会：「シーケンス制御工学　新しい理論と設計方」，オーム社（1988）

[2] 浅野哲正：「図解　シーケンサ百科」，オーム社（1990）

[3] BERTRAND MEYEAR 著，二木厚吉 監訳／酒匂寛・酒匂順子 共訳：「オブジェクト指向入門」，アスキー出版局（1990）

[4] 石川裕，所真理雄：「オブジェクト指向並行プログラミング言語」，情報処理，Vol.29，No.4，pp.325-333（Apr.1988）

[5] 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」，情報処理学会研究報告,91-SE-80-22,pp.167-174（1991）

[6] 福井眞吾：「並列システムの階層的記述に適した通信機構」，情報処理学会研究報告，91-PRG-3，pp.7-16（1991）

[7] 滝田，酒井，米田，長谷，：「オブジェクト指向と状態遷移図モデルによるシーケンス制御用言語」，情報処理学会研究報告，92-SE-83，pp.139-146（1992）

[8] 酒井貞亮，米田政明，長谷博行，酒井充：「オブジェクト指向に基づくシーケンス制御用言語のコンパイラの作成」、1994年電子情報通信学会秋季全国大会予稿（1994）