2004－SE－145　(11)
2004／8／20

状態遷移表記言語ObCLからの中間コード生成について

千葉岳史
公立はこだて未来大学

様々なソフトウェア生産方法, 方法論などが混在する中, 実際に企業間で利用されているシステムはあまりない。そんな中でCASEツールは, 使われるものの一つである。CASEツールには, プログラムを自動生成してくれるものがあるが, どのような形式が自動生成されれば良いのか。そのシステムがどのような構成であれば, 実用化されるのか。システムがどのように構築されるのか。本研究では, ツールによる自動生成に着目し, 上記のような問題について考察した。自動生成される仕組みを, 構築するのに当たって, 状態遷移表記言語ObCLを使用し, 中間データに置き換えて, 他言語へと変換する過程とした。

Making Middle-Code with ObCL

Takefumi CHIBA
Future University-HAKODATE

There are methods to develop software programs in large numbers. Case-Tool is one of software, used for developing software programs. What do we use auto-generating system? How do we make the aut-generating system? In this study, focused attention on auto-generating, I examined about those question. Making auto-generating system, I used ObCL and transformed ObCL into middle-data.And I added process which transforming middle-data into other languages.

1　はじめに

様々なソフトウェア生産方法, 方法論などが混在する中, 実際に企業間で利用されているシステムはあまりない。それは企業において, 現在あるソフトウェアがどれほど利益をもたらすかどうかが問われているためである。現状行っている開発方法より, 良い方法があったとしてもそれを導入するのには, コストがかかりすぎる場合が多い。そのため, 企業は今の導入している方法よりは, より良い方法があるとは解っていても, 結局導入できずにいるのが現状である。

そんな中ソフトウェア開発において,CASEツールを使用している企業がある。しかし, 現状に存在しているCASEツールは, データ形式がそのツール固有のものであったり, その環境が閉じたものであるので, そのツール利用者の期待どおりの作成がされないことがある。期待どおりに出力させようとしても多くの場合, 変換過程に手を加えることが出来ない事が多い。そのため, どのような変換過程を経ているかも, 明確には解らないため, デバッグもまた困難となっている。

企業において, 扱われるCASEツールとは何か, それらのツールに必要な部分はどんなものであり, 不必要な部分とはどのようなものであるのか, 開発者にとってどのような機能が使いやすいのかということに着目して, 本研究では, まずCASEツールの自動生成に焦点を絞った。今

回作成するにあたって, 状態遷移図からプログラムを自動生成するツールということを目標において進めていく。その過程で, 中間データを作成する手順を追加する。その中間データは, データベース形式で登録される。そして, 個々のツールから自由アクセスできるようにして, オープンな環境を目指す。それによって, 拡張性を持たせるほか, デバッグを行いやすく, 開発者にとって扱いやすい環境となるようにする。

2　研究概要

自動生成において, 状態遷移表記言語 ObCL を,Lex&Yacc を使用して変換し, 中間データを生成させる。中間データからは, 他言語への変換を考慮している。この中間データとして, 今回はデータベース形式で表記するようにした。そこで, 他の中間データにする形式として, いくつかと比較した。

3　状態遷移図 ObTS

ObTS とは, 状態遷移を図で表記したもので, ObTS モデルでは, システムの構造はオブジェクトの階層構造で表され, 個々のオブジェクトは局所データとしての属性, 動作を表す状態遷移図, 動作委譲のための内部オブジェクトを持つ。また, システムの動作は個々のオブジェクトの内部動作と, オブジェクト間の属性付きイベント通信によって表される。個々のオブジェクトの動作は状態遷移で表され, 遷移の際にイベントの送受信、関数的な属性計算を行う。したがって,ObTS によるモデル化とは, 対象システムからオブジェクトやイベントや属性を抽出し, オブジェクトの動作をイベント通信や属性計算を含む状態遷移図に表すことになる。ObTS でモデル化したシステムを整理/表現するために図的な記述方法を用いるのが一般的であるが, これを実際に計算機上で動作させるためには次に述べる ObCL 言語で記述する。

3.1　オブジェクトと状態遷移図

状態遷移図とは, 何らかの記述対象 (ObTS の場合はオブジェクト) の動作を状態と, 状態から状態への遷移という 2 つの概念によって表したものである。

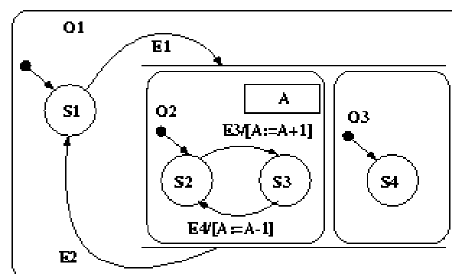

図 1: ObTS の例

ここで黒丸からの矢印はこの状態遷移図の初期状態を表しており, 通常の状態遷移を表す他の矢印には, その遷移の要因となる事象を付記する。ObTS で用いる状態遷移図では状態遷移の要因となる事象は, オブジェクトとオブジェクトの間でやりとりされるイベントで表現され, 状態遷移を規定する遷移規則ではその要因となる入力イベントの他にその結果生じる出力イベントを記述する。また、単に状態が遷移するだけでなく, それに伴って遷移規則の記述にしたがって計算が行われる。

4　状態遷移表記言語 ObCL

ObTS はモデル化しやすいが, 言語的に記述をすることはできない。ObCL とは,ObTS を言語化したものであり,ObTS に対して言語的な記述体系を与えると同時に, 記述の再利用性や可読性をもたせている。

ObCL の言語体系を, 簡単に説明する。ObCL 言語による仕様記述は, 属性クラス, イベントクラス, フィールドクラス, オブジェクトクラス, および, 対象システムのトップレベルにあるオブジェクトを指定するシステム記述からなっている。トップレベルオブジェクトとは, 対象システムが動作を開始する瞬間から起動されているオブジェクトのことで, システム中のほかの全てのオブジェクトは, どれかのトップレベルオブジェクトの子孫オブジェクト (内部オブジェクトか, 内部オブジェクトの内部オブジェクトか…) でなければならない。トップレベルオブジェクトが複数してされている場合, それらは並行行動する。

ObCL 記述のトップレベルの文法は以下のようになる。

```
      記述  ::=  (クラス記述)*システム記述
クラス記述  ::=  オブジェクトクラス記述
              |  フィールドクラス記述
              |  イベントクラス記述
              |  属性クラス記述
```

ここで,()*の記法は記述を 1 個以上繰り返してかけることを意味している。以下にて,ObCL を構成する各クラスの説明を行う。

4.1　クラス

属性クラスとは, 通常のプログラミング言語で言えば, データや属性の型に相当する概念である。例えば, 通常の言語での整数型が ObCL の整数型属性クラスに対応する。後に述べるイベントクラスやオブジェクトクラスの中で属性を宣言する場合, その型は全て属性クラスとして宣言されていなければならない。

イベントクラスとは, イベントの記述とその再利用のためのもので, イベントクラス名とイベント属性とイベント属性に対する操作 (イベント手続き, または, イベント関数) を持つ。操作は, イベントクラスの持つ属性を隠蔽するためのものであるが, 現在はオブジェクトがイベント操作を用いずに直接イベント属性を使用することを許している。

フィールドクラスは, フィールドの記述とその再利用のためのもので, そのクラスに属するインスタンス (つまりフィールド) に流れるイベントを記述する。イベントはイベント名とイベントクラス名を用いて宣言する。

オブジェクトクラスはオブジェクトの記述とその再利用のためのものであり, オブジェクトクラス名, フィールド参照, オブジェクト属性宣言, 操作宣言, 状態宣言, 内部オブジェクト宣言, および, 状態遷移規則からなる。

4.2　システム記述

システム記述は対象システムのトップレベルにあるオブジェクトをしてする記述である。ObTS ではオブジェクトの木構造によってシステム全体を記述するため, 木構造の根に当たる 1 つのオブジェクト (root オブジェクトと呼ぶ) をシステム記述で記述する。ただし,root オブジェクトには属性が不要で, 内部オブジェクトを起動すること以外の動作記述が不要な場合には,root オブジェクト自体の記述を省略して, その代わりに root オブジェクトの内部オブジェクト (並行動作するオブジェクト群であることが多い) をシステム記述に記述しても良い。

以上のクラス群を組み合わせて記述するのが,ObCL 言語の体系である。ObCL の具体的なプログラム例を以下に示す。

ObCL プログラム例

```
field F
   event e1,e2:GENERIC_EVENT
end

class C1
   field f:F
   state s1
   transition
   start is
```

```
      source init
      destination s1
   end
   t1 is
      source s1
      input f.e2
      do f.e1.val:=1
      destination s1
      output f.e1
   end
end

system S
   object o1:C1
end
```

5　提案するシステムの概要

研究しているシステムの説明とそれに対する実装の過程を説明する。まず, 利用者はこれから作成するであろうプログラムの流れを, 状態遷移図にて表記する。その後, できた状態遷移図を状態遷移表記言語 ObCL にて表記する。そして, ここでその作成したプログラムが, 変換されて, 中間コードが生成される。ここで, 変換の過程に使われているのが,Lex&Yacc であり, 中間コードに使用されているのが, データベース言語 SQL である。Lex&Yacc は, こういった文法を解釈, 分解, 変換といった処理をするのに最適な言語であり, 当研究に最適であると判断したので, 利用に至った。

中間となるデータをあえて経由してから, 利用者の目標である他言語への変換に至る過程をとり,SQL を使っている目的は二つある。1 つは, デバッグしやすいであろうということである。それは, 中間コードをわかりやすい言語, つまり, ここでデータベース言語を使うことによって, 表記が簡潔であるので作成したプログラムの特徴と違いをすばやく理解でき, すぐに間違いを正せるということである。

もう一つは, 他言語への変換が容易になるということである。特徴だけを, データ化してあるので, それを読み取りさえすれば, 他言語へ応用は簡単になる。

そして, これは最終的にツールという形をとるので, ツールを利用する上での問題点, ツールにあるべき機能, 使いやすい構成といったことを, 考えていかなければならない。

以下は, システムの概要図である。

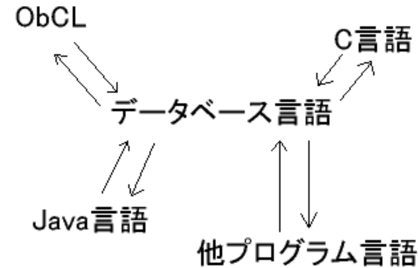

図 2: システム概要図

6　システムの実装にあたって

前に説明したようなシステムを持つ,CASE ツールを作成するべく, 研究と行動に至った。まず, 自動変換の根幹, システムの根幹といえる lex&yacc の実装から始まった。

中間コードを, 生成するのに必要な情報を取り上げるとすると,「現在の状態」,「次に遷移する状態」,「遷移条件」,「遷移するときの処理文」であると考え, それらのデータを取り上げるようにプログラムを作った。

抽出される中間コードは, 以下の表のようなものになる。

```
table1:
状態 1　 状態 2　 遷移条件文 (式)　 処理文 (式)
----------- ----- ------------- ---------
```

```
 init     S1              null     a=0,b=0
  S1      S2               a>b          X=a
  S1      S3           b==a+c        X=b-a
  S1      S1              a<=b          a+1
  S2      S3              c==b        X=b+c
```

また, 前述のプログラム記述例にて, 示した例をこの表のようにすると, 以下のようになる。

```
table2:
状態１　状態２　遷移条件文（式）　処理文（式）
-----------  -----  ------------  ---------
 init     S1              null         null
  S1      S1              null   f.e1.val:=1
```

「状態 1」は, 遷移前の状態を示し,「状態 2」は遷移後の状態を表している。「遷移条件文」は, 状態移動が行われる条件を示し,「処理文」はその状態遷移が行われるとき, 処理される動作を示している。しかし, 実際の ObCL の構成から,「状態 2」の項目は, 最後に表記される。そこは, プログラムを応用すれば, 予定通りの表にすることが出来るので, 特に問題視しなかった。また, 当初予定していたデータの取出しには, オブジェクトクラスの部分だけを検査すれば良いと考えていたが, 実際は条件文、処理文ともに他の記述部分で動作が詳しく設定されているので, その部分を含めて解析しなければならなかった。それは, 二通りの手段が取れると考えられる。あらかじめ, その動作を設定されている部分を解析する過程を追加するか, もしくはその設定をそのまま抽出して, データとして書き出すか, である。

このデータベースを利用すると, 状態 2 の項目にある S3 に関連する内容だけを, 抜き出したいと思った場合,

```
SELECT * FROM table1
WHERE 状態２ = S3
```

といったように,SQL を理解している人なら, 使うことが出来る。また,SQL といったデータベース言語を習得してない人でも使えるようなシステムを加えるということも考えられる。

途中, 中間コードをデータベース言語以外で表すとどうなるかを考えた。例えば,C 言語に変換する過程をとるとして, 問題なのが, どのように変換するかである。そこで, 状態遷移を関数にて表すようにすると, データベース言語として中間コードを設定したときと, 同じように遷移条件をどのように解釈するかが問題となった。プログラム解析を行って, 遷移条件を詳細に抽出した後, それから読み取れる動作処理を C 言語へと直す。後は, プログラムの全体の流れを, そのまま C 言語の処理へ反映していくといった手順となる。

実装に関しては, 中間コードを生成するプロセスの段階で終わっており, 中間コードを解析して, 他言語へと変換するプロセスは, 実装している段階である。中間コードからの先の変換過程の, 起こりうるであろう問題点, 障害を予測し, それに対する対策を考えた。後の章にて述べる。

7　評価

データベース言語の評価として, 表では, 現在の状態と遷移先の状態の区別がすぐにでき, また各状態の遷移状況を明示的に理解できるのだが, 遷移条件を設定するのが困難となる。しかし, 言語自体は単純なため, 変換自体はさほど難しくはない。遷移条件を単純に文字列として置き換える変換方法をとることとした。データベース型であらわす利点は, 表であるのでそのデバッグのしやすさと仕様の変更の容易さにあるだろう。そして,SQL などといったデータベース言語に対応できるようにすれば, プログラマーの知りたい情報に関連するものだけを, 引き出すことも出来る。また, 端的にプログラムの特徴を示せるので, 他言語への変換も容易である。途中の変換処理を, 大幅に無視できるのは, かなり大きい特徴と

いえる。

ObCL から C 言語へ変換することは, 容易とはいえない上に, 遷移状況が明示的に理解できないが, 記述面では優れている。言語体系が違うとはいえ, そのまま実行可能であるためだ。しかし, C 言語に変換する場合, 元の ObCL で記述した量の倍以上になる上, 状態遷移条件を解析し C 言語に変換する過程が非常に難しい。さらに, 状態の追加を行う際に生じるデバッグが困難であるため, 可逆変換を前提とするとさらに複雑になる。C 言語へ変換する際, どの程度まで C 言語に対応させるかという問題もある。

データベース形式の中間コードのモデルを簡単に示すと以下のようになる。

モデル：状態 1　状態 2　遷移条件文（式）　処理文（式）

これは, 端的に表示できる代わりに, 状態が多ければ多いほど, 記述量が多くなっていく。また, 実装の段階で説明したが, 遷移条件を詳細に抽出するのが難しく, 現状では記述を抽出するだけにとどまっている。しかし, 遷移条件等の中身について, 詳細に構文解析を行えば, 中間コード上でのより詳細なデバッグや分析が可能であるが, 状態遷移図の特性上, この部分の構文解析なしに文字列で保存しても, 相応のデバッグは可能である。問題は, 中間コードから先の変換の際に, どのように遷移条件を解析させるかである。

8　考察と結論

記述性を考えると,C 言語のようにプログラム言語への変換がより好ましいが総合性から考えると, データベース型を中間コードとしたほうが良いとわかる。また, プログラミング言語では, その使用される言語の知識が必須であり, 自分の使いたい言語へ変換させるためにその言語を学ばなければならないは不便である。結果,ObCL から C 言語を生成するのは, プログラムとしての違い以外には, 中間コードを生成するかしないかだけの違いしかないとわかった。

中間コードとして, データベース型に置き換えるのであれば, 状態遷移条件文をどのように処理するかが鍵となる。ObCL の特徴上, ただ置き換えるだけでも, ある程度は利点が見込めるが, 単純であるがゆえに情報量に不安が残る。可逆性を考えるとしたら, 逆変換できるだけの情報量を持たせなければならない。それを考えると中間コードに変換する際, 状態遷移条件文はなるたけ詳細に記す必要があるだろう。

状態遷移図表記言語 ObCL から中間コードとなるデータベース型への変換過程の一部を作成した。プログラミング言語への変換との評価により, 中間コードをデータベース型で表すことの利点が良く理解できた。また, 中間コードの形式も現状の形で納得せずに, もっと思案し, 様々な形を試すべきだ。Lex&Yacc のプログラム記述で, もっとデバッグしやすい環境を構築するというのも良い研究題材になるかもしれない。例えば,Lex と Yacc の連携が取れているのを, 自動で判別行ったり, Lex,Yacc それぞれのデバッグするべき部分を明示してくれたりといった機能を持つものを考え出してみるということだ。本研究において, 行えたことは小さなことであるが, 今後の研究において役立てることを発見できた。

9　今後の方針

今回, 中間コードに変換する過程を重点的に実装していたので, 今後は中間コードから変換する過程を研究, 実装していく。また, 自動生成する際に, どの程度まで自動生成するかを考えなければならない。全て自動生成して, 動作が保証されるようなものではなく, 部分的に生成する過程にするのであれば, ツール利用者にどの程度まで負担させるのか。生成する部分は, どのような部

分があれば利用者にとって使いやすいのか, という点である。

また, データベース形式ということを利用して,SQL といったデータベース言語を使用できるような環境の構築のほかに, それら言語を理解していない人にも使えるような環境を構築するといったことも考えられる。

自動生成する内容にも焦点を当てなければならないだろう。例えば,C 言語において状態遷移をどのように表すか。状態遷移ごとに, 関数を設定して, 関数から関数へ移動するということを状態遷移とするか, もしくは一つの関数の中に全ての状態遷移を含めてしまうか。以上を踏まえて, 中間コードからの生成過程を実装する。

中間コードから, 他言語への自動生成が形になったら, 次は逆変換の過程に取り掛かる。逆変換においては, その言語における状態遷移をどのように見出すかが問題だろう。C 言語であっても,Java 言語であっても状態遷移を示すような記述があるわけではない。それを見出すのは, プログラムを読み込む側になる。ObCL のように, 状態遷移が明記されているのであれば, 簡単であるが, 明記されてない上に, プログラムの記述には, 様々な形がある。同じような処理内容であっても, 同じような記述でなくても良い。その様々なプログラムの記述を, どのように解析して, どのように状態遷移に落とすか。現状では, 一定のプログラム記述であるという前提のもとで行う方法を考えている。いずれは, その方法以外で, 変換が可能かどうか実験するべきだろう。

この変換の成功後は, ツールとしての機能を検討していく。デバッグしやすく, 扱いやすい環境を目指すために, どのような機能であればよいのか。どのような構成であるべきなのか。幅広く使われるツールとは何かといったことを研究し組み込む。また, 自動生成される内容を利用者の思うとおりに, 変更ができるようなシステムを加えるとして, どのような変換であればよいのか。これは, これまでにやってきたことの応用となるだろう。

参考文献

[1] 伊藤　恵, 「ObTS/ObCL/ObML 利用マニュアル」, 2000.11.9

[2] John R.Levine,Tony Mason,Doug Brown 共著/村上列訳　,　lex & yacc プログラミング　1994.10.20

[3] J.J. パトリック　,　訳:株式会社コムサス,　SQL プログラミング　Oracle と Access で学ぶ SQL の基本　2000.3.30

[4] ZIPC : http://www.zipc.com/

[5] RationalRose : http://www-6.ibm.com/jp/software/rational/