ソフトウェア工学 129−13
（2000. 11. 17）

CADシステム生成のリファレンスモデルにおける手続き生成部の一実装

甲斐俊文[*1], 橋本正明[*2], 廣田豊彦[*2], 片峯恵一[*2], Lukman Efendy[*1]

[*1] 九州工業大学大学院情報工学研究科

[*2] 九州工業大学情報工学部

概要

本研究室で研究が進められているCADシステム生成のリファレンスモデルを用いた統合型建築設計支援システムIBDS(Integrated Building Design System)は、仕様からプログラムを自動生成することを目的としている。本研究はこの仕様からプログラムの構造を自動的に生成する過程で現れるドメインに汎用な中間言語から、性能の良いオブジェクト指向プログラムを生成することを目的とし、そのプログラムの手続きを決めるための手続き生成部の作成を行った。手続き生成部はERモデルをベースとした中間言語を、オブジェクト指向プログラミング言語生成のための中間言語に変換する。生成されるプログラムの属性値管理のために、手続き生成部によりTMSを組み込み、計算時間の効率化をはかった。

An Implementation of Procedure Generation on Reference Model for CAD System Generation

Toshifumi Kai[*1], Masaaki Hashimoto[*2], Toyohiko Hirota[*2], Keiichi Katamine[*2] and Lukman Efendy[*1]

[*1] Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology

[*2] Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology

Abstract

This paper describes an implementation of prosedure generation in CAD system generator for IBDS(Integrated Building Design System). The procedure generation converts the domain-free specification language into the procedural intermediate language. These language appear in a process of CAD system generation. To improve the efficiency of attribute calculation, we have adopted Truth Maintenance System in the procedure generation.

1 はじめに

既存のCADシステムにおいて、その中に現れるデータの形式はシステム毎に決まっている。このため、同じデータを扱う関連の深い業種間でデータをやりとりする際に、データの変換が必要となる。データのやりとりを行う業種毎にデータの変換が必要なため、効率が悪く、コストもかかる。また、

ユーザであるドメイン専門家の要求を正確にとらえ、システムに反映することは困難である。そのため、システムを使った作業の生産性が悪くなってしまう。本研究室で研究を進められているCADシステム生成のリファレンスモデル[1]においては、ドメイン専門家の思考に現れる概念を表しているモデルをベースとした、ドメインごとに特化した仕様記述言語を用いることによって、要求を正確にとらえることができる。

また、このドメインごとに特化した仕様記述言語をドメインに汎用な中間言語に変換し、そこからシステムを生成することによって、異なるデータモデルの統合を行う。

本研究室ではこのリファレンスモデルを用いた統合型建築設計支援システムIBDS(Integrated Building Design System)の研究を行っている。IBDSは構造設計ドメインに特化した仕様記述言語BDL(Building Design Language)や意匠設計に特化した仕様記述言語を規定している。また、ドメインに汎用な中間言語として、ERモデル(Entity-Relationship Model)[2]を採用したドメインフリー中間言語も規定している。

このような言語は計算機で直接実行できないので、我々は仕様から手続き型プログラムの構造を自動設計する方式を考案し、C++プログラムを生成するコンパイラを研究中である[3, 4]。

本研究では、仕様からプログラムの構造を自動的に生成する過程で現れる中間言語から、性能の良いC++プログラムを生成することを目的とし、手続き生成部の作成を行う。この手続き生成部は、ERモデルをベースとしたドメインフリー中間言語をオブジェクト指向プログラミング言語生成のための手続き用中間言語に変換し、必要な手続きを生成する。

本稿の第2章では研究の位置付けについて述べる。第3章ではプログラム生成方式について述べる。第4章では中間言語の定義や手続き生成部のプログラムについて述べる。第5章では本研究の考察を述べる。

2 研究の位置付け

2.1 CADシステム生成のリファレンスモデル

CADシステム間のデータ統一とドメイン知識を持った使いやすいシステムの生成を実現するために、本研究室ではCADシステム生成のリファレンスモデルの研究が進められている。

図1のように、GUI Descriptions, Display Model Descriptions, Domain Model Descriptionsの3つのモデル記述から、CADシステムが生成される。

このうち、Domain Model Descriptionsは構造設計や意匠設計といったような各ドメインごとにドメイン特化仕様記述言語(DS-SL)があり、ドメイン毎にドメイン専門家の頭の中に現れる概念に近いモデルをベースとしている。これにより、正確な要求仕様を記述できる。

これらのドメイン特化仕様記述言語は、ドメインに依存しない言語(DF-SL)に変換される。この言語でドメイン間の差異を吸収し、データモデルを統一することによって、各ドメインのシステムにおけるデータ形式の共通化を可能にする。

また、DF-SLによりプログラミング言語への変換も、同じ方式で行うことが出来る。各プログラミング言語毎にDF-SLから手続き用言語(PIL)、プログラミング言語(PL)への自動変換を行うジェネレータを1つ用意しておけば、どのドメイン特化仕様記述言語からもDF-SLを経て同じジェネレータでプログラミング言語への自動変換を行うことができる。

2.2 建築構造設計用の仕様記述言語ＢＤＬ

現在、本研究室ではCADシステム生成のリファレンスモデルを用いた統合型建築物設計支援システムＩＢＤＳ(Integrated Building Design System)の研究を行っている。

IBDSは、建築構造設計というドメインに特化した仕様記述言語ＢＤＬ(Building Design Language)を規定している。IBDSは建築設計の専門家自身が仕様を記述することによって、CADシステムの生成を行うことを目指している。そのため、BDLは建築設計の知識を建築設計者自身が整理し、記述することができる言語となっている。

建築物の構造の概念モデルは、設計作業に必要な建築物そのものに関する情報を、建築物の属性の集合として表すための枠組である。ＢＤＬのモデルは、ＥＲモデルをベースにリレーションシップを部材の依存関係に特化したものである。以下に、ＢＤＬの概念を示す。

建築物の構造は、建築物の属性の集合であり、属性は建築物を構成するそれぞれの部材に属している。各部材は接続関係、位置属性などの属性を持つ。

属性は相互に依存関係を持ち、設計対象の編集によって一部の属性値が変更されると、その属性に依

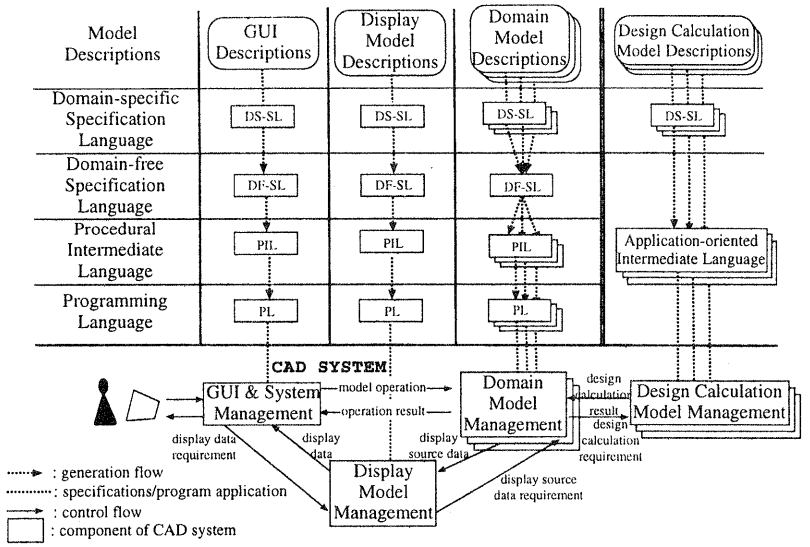

図 1: CAD システム生成のリファレンスモデル

存する属性値を再計算する必要がある。これを属性の従属性と呼ぶ。また梁が二本の柱に支えられているように、部材間には存在の依存関係があり、これを存在の従属性と呼ぶ。

部材間の存在の従属性は、プラグとソケットと言う部材間のリレーションシップのそれぞれの役割として特化されている。プラグとは、ある部材が存在するために必要な部材を指すものであり、ソケットとは、依存される部材を指すものである。

2.3 仕様記述言語の言語処理と中間言語

仕様記述言語は計算機で直接実行できない。このため、仕様記述言語でかかれた仕様を計算機上で実動化するには、言語処理を施してプログラミング言語に変換する必要がある。IBDS では、この言語処理を CAD システム生成のリファレンスモデルに沿った流れで行う。

ドメインに特化した仕様記述言語に字句解析及び構文解析、意味処理を行い、ドメインフリー中間言語 (DF-SL) を生成する。これに手続き生成を行い、手続き用中間言語 (PIL) に変換する。最後にコード生成により実装言語で記述されたプログラムが生成される。

ドメインフリー中間言語は ER モデルをベースにしており、存在従属性制約や属性の従属性制約の概念を持つ。手続き用中間言語はオブジェクト指向プログラミング言語への変換のための中間言語であり、制約を実現する手続きやオブジェクトを管理するための手続きなどが記述される。

2.4 手続き生成部の作成

要求仕様はユーザからの要求を正確に記述することが重要であり、仕様記述言語では記述性や理解性に主眼がおかれている。そのため、仕様記述言語で記述された仕様を計算機上で実働化するためには、言語処理によるプログラミング言語への変換が不可欠である。

本研究室では、仕様記述言語の言語処理について研究がなされており、その一環として先に述べた言語処理の過程における、手続き用中間言語から C++プログラムを生成するコード生成部が作成されている。

そこで本研究では、仕様記述言語の言語処理における手続き生成部の作成を行う。これによりドメインフリー中間言語から手続き用中間言語を生成し、コード生成部により効率の良い C++プログラムを生成することを目的としている。

3 プログラム生成方式

手続き生成部が受け取るドメインフリー中間言語には、ER モデルにおけるエンティティタイプとリレーションシップタイプの定義が記述されており、

その中に属性の計算式も記述されている。しかしこれを実際に計算機に処理させるための手順は含まれていない。このため、手続きを手続き生成部において生成する。

手続き生成部では、主にエンティティやリレーションシップの管理や属性値の管理のための手続きを生成する。

3.1 エンティティ及びリレーションシップの管理

オブジェクトモデルでは、エンティティタイプとリレーションシップタイプはクラスとして扱い、エンティティやリレーションシップはそれぞれのクラスのインスタンスと見なす。ここで、エンティティ型のインスタンスとリレーションシップ型のインスタンスを,それぞれオブジェクトとリンクと呼ぶことにする。図2にこのモデル変換の例を示す。

接続されたオブジェクトとリンク相互の参照は、ポインタによって実現する。このポインタ名にはロール名を用いる。このようにして、接続されたオブジェクトとリンクは、互いにポインタで参照可能となる。リンクによって関連を持つオブジェクト同士は、このポインタをリンクを介して辿っていき、相手を参照することができる。また、それぞれのオブジェクトやリンクをそのクラス毎にまとめて管理する必要がある。このため、それぞれのクラスについて管理リストを用意し、生成されたインスタンスはこのリストにつながれる。

このようにクラスを定義し、必要な手続きを生成することで、エンティティ及びリレーションシップの管理は実現される。ここで、必要な手続きとは、オブジェクトとリンク間の接続や切断、関係を持つオブジェクトの参照、管理リストへの連結や削除などの処理を行うものなどである。

3.2 属性の管理

属性の値が、他の属性値を用いる計算式によって求まる場合、その属性は従属属性である。従属属性は、属性の従属性制約により、計算式に含まれている他の属性値が変化すると、それにともなって値が変化して、正しい値に更新される。ドメインフリー中間言語 では各従属属性に計算式が与えられているので、従属性制約を実現するような手続きを与えなければならない。

手続きを実現するにあたって、属性値の値をいつ再計算するのかが問題となる。関連する属性が更新される度に、その属性を用いて値が決定される全ての従属属性を再計算する、という方法がある。また逆に、従属属性が参照されたときに、再計算を行う、という方法もある。しかし、これらの方法では再計算した値が参照されないまま次の再計算が発生したり、変更されていないのに再計算を行ったりするため、無駄な計算が多くなってしまう。

そこで本研究では、効率化のためにTMS (Truth Maintenance System) を用いて従属属性の管理を行うことにした。

TMS TMSは知識集合において、知識同士の論理的整合性を維持するための機構である。TMSにおいて知識は信念と呼ばれ、それぞれの信念が現在、信じられているかどうかを記録している。ここでは、信じている状態を in 状態、信じられていない状態を out 状態と呼ぶことにする。TMSはそれぞれの信念について、その信念が in 状態になるための条件を表す弁明も記録している。知識集合内の信念の状態が変化したり、新しい信念が加わると、TMSはその変化によって生じる矛盾を解消する働きをする。

TMSを用いた属性値の管理 それぞれの属性の現在の値が信念となり、in 状態なら正しい値、out 状態なら計算をしなおす必要があるとみなす。

ある属性が変化し、新しい値が in 状態とみなされると、その属性の従属属性の値が、以前計算された値のまま in 状態であれば、矛盾が生じる。そこでTMSは矛盾を解消するために、変化した属性に影響を受ける全ての属性の状態を、out 状態に書き換える。このとき、再計算は行わないで、属性は元の数値を保持したままになる。再計算は、属性が参照されたときにはじめて行われる。属性が参照されると、属性の状態がTMSによって out に書き換えられているかをチェックし、in 状態のままであればそのまま属性の値を使い、out 状態であれば再計算を行う。再計算を行って属性が正しい値に更新されると、属性の状態を in に戻す。

4 実装

4.1 中間言語

手続き生成部はドメインフリー中間言語を入力として与えられ、手続き用中間言語を出力する。このドメインフリー中間言語と手続き用中間言語について順に説明する。

(1) ドメインフリー中間言語

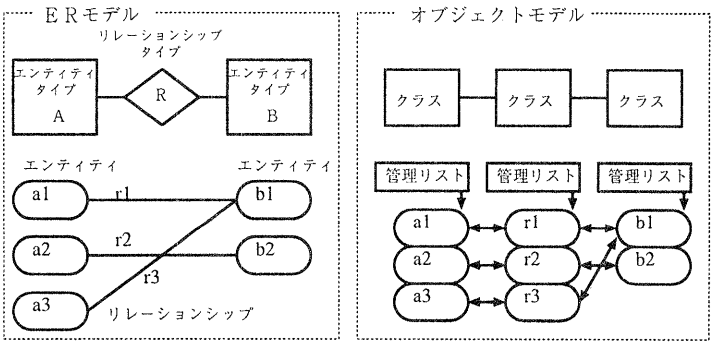

図 2: ERモデルからオブジェクトモデルへの変換

- **bdl_class**(*classType, className, classNo, cardinal*).
  クラスの定義。

- **bdl_attribute**(*className, classNo, attriName, attriNo, attriType, value, type, unit*).
  各クラスの持つ属性の定義。

- **bdl_computeAt**(*className, constNo, '', obSelf, attriName/virtualName, expression*).
  従属属性の計算式の定義。

- **bdl_constraintParam**(*constNo, paramName, roleName, className, virtualName,nil*).
  bdl_computeAt で用いられる一時変数の定義。

(2) 手続き用中間言語

- **bdl_class**(*ClassName*).
  クラスの定義を行う。

- **bdl_classUsage**(*ClassName, b, 1/n*).
  クラスの特性の定義を行う。

- **bdl_superClass**(*ClassName, SuperClassName*).
  継承関係を定義する。(スーパークラスの定義)

- **bdl_subClass**(*ClassName, SubClassName*).
  継承関係を定義する。(サブクラスの定義)

- **bdl_att**(*ClassName, AttributeName, DataType, DataUnit*).
  属性を定義する。

- **bdl_attUsage**(*ClassName, AttributeName, c/b/d, 1/n*).
  属性の特性を定義する。

- **bdl_attValue**(*ClassName, AttributeName, DefaultValue*).
  属性の初期値の定義。

- **bdl_mtd**(*MethodType, 11/1n/nn, ClassName, AttributeName, ViewID, Parameters, Expressions*).
  メソッドを定義する。

- **bdl_mtdUsage**(*MethodType, 11/1n/nn, ClassName, AttributeName, purevirtual/virtual/real*).
  メソッドの特性を定義する。

- **bdl_mtdPar**(*MethodType, 11/1n/nn, ClassName, AttributeName, ViewID, ParamName, ParamType*).
  メソッド内のみに使用される変数を定義する。

- **bdl_mtdRet**(*MethodType, 11/1n/nn, ClassName, AttributeName, ViewID, ReturnName, ReturnType*).
  メソッド内のみに使用される戻り値を保持する変数を定義する。

- **bdl_mtdVar**(*MethodType, 11/1n/nn, ClassName, AttributeName, ViewID, Variable-*

Name, VariableType).
メソッド内で使用される変数を定義する。

bdl_mtd() の Expressions で扱われる文法(手続き用中間言語)

- **seq**(*[elem ...]*)：順次処理(制御構文)
  リスト内の要素(処理)を順番に実行する。
- **sel**(*[pre_elem ...][expretion][true_elem ...][false_elem ...]*)：条件分岐文(制御構文)
  前処理として1番目のリストの処理を行い、2番目のリストにある条件式を計算し、式が真なら3番目、偽なら4番目のリストの処理を行う。
- **acs**(*MethodType, 11/1n/nn, ClassName, AttName, [RolePtr,Value]*)：メソッドを呼び出す。
  引数によって特定されるメソッドを呼び出す。このとき、RolePtr(ポインタ)を使って相手のオブジェクトにアクセスする。Value はメソッドの引数になる。
- **call**(*MethodType, 11/1n/nn, ClassName, AttName, [RolePtr,Value]*)：ライブラリに用意されたメソッドを呼び出す。
- **funcn**(*FunctonName, ParamName, Variable*)：関数を呼び出す。
- **exc**(...)：特殊な処理を行う。
  オブジェクトの生成や削除などの特殊な処理を行う。
- **substExpr**(*Variable, Type, Expretion*)：
  値の代入。
  Expretion の値を Variable に代入する。
- **selExpr**(*Expretion*)：条件式。
  Expretion を評価して、真か偽を返す。制御構文 sel の動作に影響する。

4.2 処理手順

ドメインフリー中間言語に記述されたそれぞれのエンティティタイプやリレーションシップタイプについて、次の2つのクラスを生成する。そのエンティティタイプ(またはリレーションシップタイプ)自身を表すオブジェクトクラス(またはリンククラス)と、そのクラスを管理する管理リストクラスである。この管理リストクラスは、コード生成部で組み込まれるリストクラスのサブクラスとして定義する。

それぞれのクラスについて以下の処理を行い、手続き用中間言語 を出力する。

- オブジェクト / リンク管理リストクラス
  - クラス定義
    クラス名を定義する。また、インスタンスを1つだけ持つことを定義する。list クラスをスーパークラスとすることも定義。
  - オブジェクト / リンク管理のためのメソッドの生成
    オブジェクトの場合は、管理するオブジェクトの生成や、複数のオブジェクトに対し、それらに関係を持つオブジェクトやリンクを取得するメソッドを生成する。リンクの場合は、管理するリンクの生成及び生成や削除によって影響を受ける属性へのTMS用フラグの更新を行うメソッドを生成する。そして、複数のリンクに対し、それらに関係を持つオブジェクトを取得するメソッドを生成する。
  - それぞれの属性に関するメソッドの生成
    複数のオブジェクトの属性に対して、値の取得と設定、定義された計算式の実行、TMS用のフラグの設定を行うメソッドを生成する。
- オブジェクト / リンククラス
  - クラス定義
    クラス名を定義する。また、インスタンスを複数持つか1つだけ持つかを定義する。
  - 属性の定義
    オブジェクトの場合は、ドメインフリー中間言語で定義された属性の定義及び、関係を持つリンクへのポインタ、TMSのためのフラグを定義する。リンクの場合は、関係を持つオブジェクトへのポインタを生成する。
  - オブジェクト / リンクに関するメソッドの生成
    オブジェクトの場合は、属性の初期化、そのオブジェクト自身の削除、関係の接続と切断、関係するオブジェクトやリンクの取得を行うメソッドを生成する。リンクの場合は、

オブジェクトとの関係の接続と切断、それによって影響を受ける属性へのＴＭＳ用フラグの更新、関係するオブジェクトの取得を行うメソッドを生成する。

- それぞれの属性に関するメソッドの生成
  オブジェクトの場合のみ、ドメインフリー中間言語で定義された全ての属性について、値の取得と設定、定義された計算式の実行、ＴＭＳ用のフラグの設定を行うメソッドを生成する。

4.3 処理例

手続き生成部の入力として与えられるドメインフリー中間言語の中に、例えば以下のようにcustomer エンティティが定義されていたとする。

```
bdl_class(o,'customer',2,us).
bdl_plugList('customer',[]).
bdl_socketList('customer',['kounyuu']).
bdl_attribute('customer',2,'kounyuu_price',
  1,v,['kounyuu','price'],'nil','nil').
bdl_attribute('customer',2,'function1',2,
  function,['nil','obSelf'],'nil','nil').
bdl_attribute('customer',2,'highestprice',
  3,d,['nil','obSelf'],'int','nil').
bdl_computeAt('customer',2,'','obSelf',
  'function1',['max','V1']).
bdl_constraintParam(2,'V1','','obSelf',
  'kounyuu_price',nil).
bdl_computeAt('customer',1,'','obSelf',
  'highestprice','V1').
bdl_constraintParam(1,'V1','','obSelf',
  'function1',nil).
```

手続き生成部によって手続き用中間言語で customer クラスとその管理クラスである customerList クラスが定義される。その行数は合わせて 400 行以上になる。

以下にその生成されたドメインフリー中間言語のファイルから、customer クラスの定義部分の一部を示す。

```
bdl_class(customer).
bdl_classUsage(customer,b,n).
bdl_att(customer,function1,(int,nor),nil).
bdl_attUsage(customer,function1,function,n).
bdl_attValue(customer,function1,[nil,obSelf]).
bdl_att(customer,highestprice,(int,nor),nil).
bdl_attUsage(customer,highestprice,d,n).
bdl_attValue(customer,highestprice,[nil,
  obSelf]).
bdl_mtd(getTmsAtt,'11',customer,function1,v0,
  [outVal],
  sel([
    acs(getFlag,'11',(customer,ptr),
      function1,[self,att_flag])],[
    selExpr('att_flag == flag_NG')],[
      acs(attDepn,'11',(customer,ptr),
      function1,[self]),
    acs(getDrtAtt,'11',(customer,ptr),
      function1,[self,outVal])],[
    acs(getDrtAtt,'11',(customer,ptr),
      function1,[self,outVal])
    ])).
bdl_mtdUsage(getTmsAtt,'11',customer,
  function1,real).
bdl_mtdVar(getTmsAtt,'11',customer,function1,
  v0,att_flag,(tmsflag,nor)).
bdl_mtdRet(getTmsAtt,'11',customer,function1,
  v0,outVal,(int,nor)).
                    .
                    .
                    .
```

5 考察

5.1 生成されるプログラムの評価

本研究では、手続き生成部の作成を行う前に、ドメインフリー中間言語から生成されるＣ＋＋プログラムを人間の手で作成することにより、生成の処理に必要な知識を整理する作業を行った。この作業では、販売在庫管理の例題を、ドメインフリー中間言語からＣ＋＋プログラムに変換した。このときにハンドコーディングによるＣ＋＋プログラムと、本研究で作成した手続き生成部を用いて自動生成したＣ＋＋プログラムの性能の比較を行った。

(1) 処理速度比較

顧客オブジェクトを１つ生成しておく。ファイルから値段と商品番号を入力として受取り、その属性値をもつ新しい商品オブジェクトを生成し、それを顧客オブジェクトに関係づける処理を行った。この商品オブジェクトを生成し、リンクも生成して関連づけを行う処理にかかった時間を計測したところ、オブジェクト 1000 個あたりハンドコーディングでは 0.128 秒、自動生成では 0.125 秒だった。

続いて、始めに生成した顧客オブジェクトの属性である支払代金の計算を行った。支払代金の値は、リンクを経由してつながっている全ての商品オブジェクトの値段の値を合計する計算により得られる。

この属性は商品オブジェクトが追加されることにより、TMSによって再計算が必要であると判断される。このため、参照してから、再計算がなされ、値が得られるまでの時間を計測した。結果は、オブジェクト数 1000 個あたりハンドコーディングでは 0.041 秒、自動生成では 0.041 秒かかった。

以上の結果より、処理速度は人の手によって記述されたプログラムと同等の性能であると言える。

(2) 静的ステップ数比較

C++プログラムをアセンブラに変換し、その静的ステップ数を調べた。静的ステップ数は、アセンブラのステップ数に相当する。その結果、最適化前はハンドコーティングが 8030 ステップ、自動生成が 7230 ステップとなった。最適化後はハンドコーティングが 3382 ステップ、自動生成が 2908 ステップだった。

静的ステップ数は、最適化する前は 1.20 倍、最適化をかけると 1.26 倍となった。

以上の結果より、人の手で作成されたプログラムとほぼ同等の性能のプログラムが生成されたことが確認できた。したがって、本研究で作成した手続き生成部と処理の追加や変更を行ったコード生成部は、性能のよいプログラムを生成できるといえる。

5.2 今後の課題

データの構造不一致 プログラムにおいて、計算に用いられるデータの列が、計算される順番通りに入力されるとは限らない。このような状態のことを構造不一致と呼ぶ。

プログラムを組む人は頭の中で構造不一致を検出し、入力データを配列等にバッファリングすることで、この問題を解決する。

本研究で作成した手続き生成部では、全てのデータ(オブジェクト)をリストにつないで保持することで、構造不一致の問題を回避している。

しかし、この方法では構造不一致でない構造に対してもデータのバッファリングを行っており、メモリを無駄に使っていることになる。

メモリ効率を改善するために、構造不一致を検出し、それに合わせたプログラムを生成するための機構を、手続き生成部に組み込むことが必要となる。

データフローによる属性値計算 本研究で作成した手続き生成部では、属性値の計算をTMSによって管理することで効率化をはかった。しかし、構造不一致の問題と同様に、人は計算のタイミングを頭の中で考え、効率の良いプログラムを実現する。

このため、データフローによる属性値計算を行えるように手続き生成部へ機構を組み込み、どちらの方式で属性値の計算を行うかを選択できる様にすることが、処理時間の効率化のために必要である。

5.3 おわりに

本研究では、ドメインフリー中間言語から性能の良いC++プログラムを生成する、手続き生成部の作成を行った。生成されるプログラムの属性値管理のために、手続き生成部によりTMSを組み込み、計算時間の効率化をはかった。

性能評価に用いた販売在庫管理のプログラムの例では、作成した手続き生成部を通して生成されたC++プログラムの性能は、ハンドコーディングで作成したC++プログラムとほぼ同程度であった。このことにより、生成されたC++プログラムの性能の良さは確かめられた。

本研究で作成した手続き生成部では、構造不一致の検出を行っていない。また、属性計算の管理についても処理時間の無駄がある。このため、手続き生成部に構造不一致の検出とそれに合わせた手続きを生成する機構と、データフローによる属性値の管理を行う機構を組み込むことが今後の課題である。

参考文献

[1] L.Efendy, M.Hashimoto, K.Katamine and T.Hirota. A referrence model of CAD system generation from Various object model-based specification description languages specific to individual domains. In *IEICE TRANS. INF. & SYST., VOL.E83-D, NO.4 APRIL 2000.*

[2] P. P. Chen. The entity-relationsship model-toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[3] M. Hashimoto and K. Okamoto. A set and mapping-based detection and solution method for structure clash between program input and output data. In *Proc. IEEE Computer Software and Application Conference*, pages 629–638, 1990.

[4] 山崎 充彦, 廣田 豊彦, 橋本 正明. 仕様記述言語PSDLからオブジェクト指向言語への変換. 情報処理学会第 *48* 回全国大会 *(1G-10)*, 3 1994.