4V-6 オブジェクト／制約指向パラダイムによる設計言語Ｄｅｌｐｈｉ

石橋 勝典* 小笠原 秀人* 石川 孝** 菅野 文友*
* 東京理科大学 工学研究科
** ぺんてる株式会社中央研究所

１．はじめに

コンピュータを設計支援に用いようとするとき，対象に関する知識をコンピュータに持たせる必要がある．この知識は，たいてい汎用のプログラム言語と同レベルで書かれるため，知識を記述するためには，その言語をマスターしているか，あるいはマスターしている人に翻訳してもらう必要がある．対話的に知識を取り込むという方法もあるが，編集などの点で問題がでてくる．

また，設計知識の記述に際しては，“設計のノウハウ”も記述する必要がある．これは，宣言的な知識とは種類が異なり，一緒に記述することが難しい．

本稿では，上記の点を考慮し，総合設計支援環境の一環として設計した設計言語“Ｄｅｌｐｈｉ”の概要と，これで書かれた知識を計算可能な内部表現に変換するパーザについて，紹介する．

２．概要

設計言語Ｄｅｌｐｈｉは，設計者自身が設計知識を記述できるようにすることを目的とした，設計知識表現用言語である．想定した総合設計支援環境におけるＤｅｌｐｈｉの位置づけを，図１に示す．

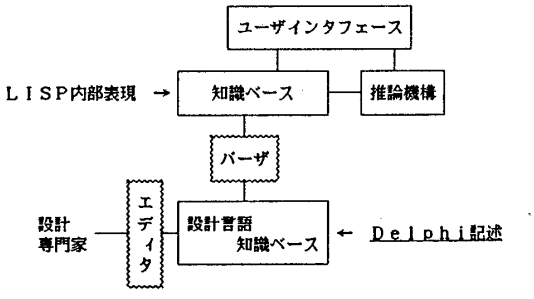

図１ 設計支援環境の概要

３．設計言語“Ｄｅｌｐｈｉ”

３．１ 設計方針

Ｄｅｌｐｈｉを設計するにあたり，以下の点に留意した．

① 設計者自身が知識を記述できるようにすること．

② 設計ノウハウを設計対象間の制約として記述できること．

①を満足させるため，知識の記述に「オブジェクト指向」の考え方を取り入れた．設計対象を，変数の値によって特徴づけられるもの（オブジェクト）の集まりとして記述することによって，対象の構造を反映した知識のモジュール化を進めることができる．そして構文は，できるだけ簡単にし，記述上の制約をあまり多く設けないようにした．

②を満足させるため，「制約指向」の考え方を取り入れた．制約指向では，対象間の制約が宣言的に記述される．Ｄｅｌｐｈｉでは，変数の値に対して，制約を変数の間の関係として記述することとした．これによって，各変数間の因果関係だけを明示することで，計算機自身が実行順序をたどり，計算を実行することができる．

３．２ 仕様

Ｄｅｌｐｈｉ（Design Language ZERO）の構文（部分）を 図３．１に，力変換器の設計知識[1]の記述例を図３．２に示す．

```
<Delphi>::=
    <クラス名定義文> {<代入文>}* {<インスタンスデータ>}*
<クラス名定義文>::=
      <クラス名> [< {<上位クラス名>}*]
<代入文>::=
      [<変数記号>:] @<変数名>=<計算式> [//<条件文>]
```

図３．１ Ｄｅｌｐｈｉ構文（部分）

```
力変換器
    @ビーム=ビーム // @ビーム@WT<MWT & LD<MLD
    @ストレインゲージ=ストレインゲージ
                // strain<@ストレインゲージ@maxstrain
    F：@最大測定力
    MWT：@最大重量
    MLD：@最大横変位
    ML：@最大長さ
    LD：@横変位=F*@ビーム@L*@ビーム@L
                /@E*@ビーム@断面@I
    strain：@ひずみ=F*SGA/S*E
    SGA：=@ストレインゲージ@長さ
    S：=@ビーム@断面@断面係数
    E：=@ビーム@材質@E

    (nil,nil,100,20,0.5,50,nil,nil)
    (nil,nil,150,50,1.0,100,nil,nil)
```

図３．２ Ｄｅｌｐｈｉ記述例

Design language "Delphi" based on object/constraint oriented paradigm

Katsunori ISHIBASHI*, Hideto OGASAWARA*, Takashi ISHIKAWA**, Ayatomo KANNO*
* Science University of Tokyo, ** PENTEL CO.,LTD.

Ｄｅｌｐｈｉにおける計算は，値の選択または算術計算によって，インスタンスデータの値を決定することである．　Ｄｅｌｐｈｉの特徴の１つに，“@（アットマーク）”演算子と，“／／（ダブルスラッシュ）”演算子の導入がある．

@演算子：変数評価演算子

@＜変数名＞

変数名に対応するインスタンスデータの値を返す．もし，その値がｎｉｌだったら（値がなければ），代入文を評価することによって，変数名に対応する値を返す．

//演算子：選択演算子

＜計算式＞//＜条件文＞

計算式を評価して，結果の１つを選択し，条件文を評価．その結果が真だったら計算式の評価結果を返す．偽だったら，別の結果を選択し，条件文を再評価する．さらに別の結果がなければ，ｎｉｌとする．

４．パーザの作成

Ｄｅｌｐｈｉで書かれた知識を，計算機上で計算可能な形（内部表現）に変換する必要がある．今回は，この内部表現をｌｉｓｐ表現と設定し，その表現へＤｅｌｐｈｉを変換するパーザを作成した．内部表現の仕様を図３．３に，図３．２を内部表現に変換した変換例を図３．４に示す．

```
<内部表現>::=(<クラス名>
                (super ( {<スーパークラス名>}* ) )
                (variable ( {<変数定義>}* ) )
                ( {<インスタンスデータ>}* ) )
<変数定義>::=(<変数名><代入文><変数記号>)
<インスタンスデータ>::=( {<値>}* )
```

図３．３　内部表現仕様

```
(力変換器 (super nil)
  (variable
   (ビーム (// (and (< (@ 'WT (@ 'ビーム self))(@ 'MWT self))
                    (> (@ 'LD self)(@ 'MLD self)))ビーム) nil)
   (ストレインゲージ (// (< (@ 'strain self)
                            (@ 'maxstrain (@ 'ストレインゲージ self)))
                         ストレインゲージ) nil)
   (最大測定力 nil F)
   (最大重量 nil MWT)
   (最大横変位 nil MLD)
   (最大長さ nil ML)
   (横変位 (* F  (* (@ 'L (@ 'ビーム self))
                    (* (/ (@ 'L (@ 'ビーム self))(@ 'E self))
                       (@ 'I (@ '断面 (@ 'ビーム self))))))) LD)
   (ひずみ (* F (* (/ SGA S) E)) strain)
   (SGA (@ '長さ (@ 'ストレインゲージ self)) nil)
   (S (@ '断面係数 (@ '断面 (@ 'ビーム self))) nil)
   (E (@ 'E (@ '材質 (@ 'ビーム self))) nil))
   ((力変換器 (nil nil 100 20 0.5 50 nil nil))
    (力変換器 (nil nil 150 50 1 100 nil nil)))
```

図３．４　変換例（図３．２の内部表現変換結果）

５．評価

第一の目的が「書きやすさ」であったが，この点に関する評価はつけにくい．今回のＤｅｌｐｈｉ記述例は，ある程度まとまっている知識をもとに記述してみたが，まとまっていない知識についても記述してみる必要がある．

オブジェクト指向，宣言的な記述という点で，従来の手続き的なプログラミングに慣れている人にとっては，書きにくい点があるかもしれない．

また，知識が競合した場合などの対策を設定していない．これについては，競合解消演算子を導入し，式評価時のルールを書き込めるようにすることによって，競合を解消しようと考えている．

今回は，あくまで構文と評価規則を決めたにすぎず，今後は，構築した知識の管理も考えていかなければならない．

６．おわりに

総合設計支援環境の一環として設計した設計言語“Ｄｅｌｐｈｉ”の概要と，このＤｅｌｐｈｉで書かれた知識を内部表現に変換するパーザについて，事例的に紹介した．

今回設計，作成したものは，まだ試行途中のプロトタイプ的なものである．今後は，いろいろな設計知識をＤｅｌｐｈｉで記述し，その有効性を確認すると共に，機能の追加・修正を行なっていく予定である．

以上

[参考／引用文献]

[1] Agustin A.Araya and Sanjay Mitall：
"COMPILING DISIGN PLANS FROM DESCRIPTIONS OF ARTIFACTS AND PROBLEM SOLVING HEURISTICS.", Proc. Int. Jt. Artif. Intell. Vol.10th No.Vol.1, (1987).

[2] 相場　亮："制約論理プログラミング", bit, Vol.20 No.1, (1988).

[3] 長澤　勲 "設計エキスパートシステム", 情報処理, Vol.28 No.2, (1987).