ソフトウェア工学 116－4
（1997．9．11）

機械設計のためのオブジェクトモデル

廣田豊彦† 西岡竜大†,☆ 橋本正明†

設計者の知的活動を支援することを目的として，我々は，設計プロセス全体で共有されるプロダクトモデルを構築した．プロダクトモデルは，設計者の知識を反映させるために，属性モデリングの考え方を用いている．さらに，設計者が自身の知識を直接記述できるように，プロダクトモデル記述言語 PMDL を導入した．PMDL はオブジェクト指向モデルを基礎とし，機械設計特有の概念と機構を追加している．例題として，組立図生成ツールを試作した．

Object Model for Mechanical Design

TOYOHIKO HIROTA,† TATSUHIRO NISHIOKA† and MASAAKI HASHIMOTO†

To support the intellegent design activities of designers, we have constructed a product model that can be shared by all processes of a design task.The product model incorporates attribute-oriented modeling in order to represent a given designer's knowledge. Moreover, to directly reflect designer's intentions, we have introduced the Product Model Description Language (PMDL) with which a designer can transscribe his knowledge by himself. PMDL is based on object-oriented modeling and features concepts and mechanisms specific to mechanical design. As an example, we have developed assembly chart generator using our product model.

1. はじめに

機械設計などの工学的分野において，今だに十分な計算機支援が実現していない．従来から図面のドラフティングや CAD システムなどのツールは提供されてきたが，設計者の知的作業である「設計」や「組立図作成」・「スケジューリング」自体を支援するツールなどは皆無に等しい．計算機による設計支援が実現していない理由として，現在の設計ドメインにおける以下の問題点が挙げられる．

- プロセス中心の処理形態であり，各プロセスにおけるデータ共有が不十分
- 設計物のモデルが確立されていないため，設計者の知識が整理されていない

これらの問題の解決策として，我々は以下のアプローチを採っている．

- 設計物を表すモデルとしてプロダクトモデルを構築
- 設計者の知識を表現する手法として，属性モデルを設計物のモデルに反映させる

更に設計者の意図を直接モデルに反映するために，設計者自身が知識を記述できる枠組として，プロダクトモデル記述言語を提示している．

設計プロセス

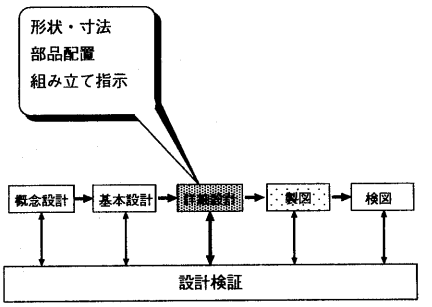

図1 設計プロセス

設計ドメインにおけるプロセスの流れは図1である．我々は設計物のモデルであるプロダクトモデルを構築し，詳細設計プロセスおよび製図プロセスにおける設計支援システムの試作，検証を行なっている．

プロダクトモデルによる設計支援

プロダクトモデルを用いた設計支援システムは図2の様な形態をとっている．設計物の情報，つまり設計物の組立構造，部品の属性，部品間の結合表現などは，設計者がプロダクトモデル記述言語 (PMDL) を用いて記述

† 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence, Kyushu Institute of Technology
☆ 現在，(株) 東芝
Presently with Toshiba Co.

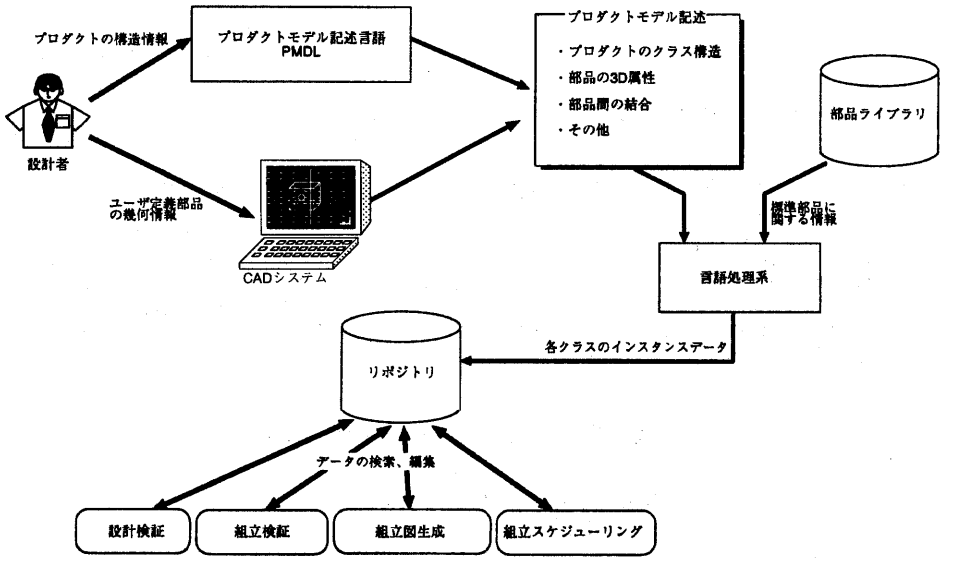

図2　プロダクトモデルを用いた支援システム

する．幾何情報はCADにより入力され，プロダクトモデル記述に変換される．この記述は言語処理系によりリポジトリに設計物データとして保存され，各設計支援システムはこのリポジトリにある設計物の情報をデータとして扱う．このように設計システム全体で設計物のデータを共有することにより一貫した設計支援が可能である．本報告書ではプロダクトモデル及びそれを用いた設計支援システムの例として組立図生成ツールを紹介する．

2. プロダクトモデル

プロダクトモデルは属性モデルの概念に基づいたオブジェクト指向モデルである．設計システムは設計者の意図を反映できるものでなければならないので，プロダクトモデルは，設計ドメインにおける概念を反映したものでなければならない[1]．また，設計知識が開発者により固定されないようにプロダクトモデルは設計者によりPMDLを用いて記述される．そこで我々はプロダクトモデルの構築にあたって，オブジェクト指向モデルの概念に加えて設計ドメイン特有の概念やメカニズム[2]を導入した．以下では，我々が導入した概念やメカニズムについて説明する．

2.1 組立構造におけるインスタンス生成

設計物は図4のような集約による階層構造で表現されている．設計物あるいはその中間生成物であるサブアセンブリは，その構成要素である部品またはサブアセンブリの集約であり，部品は「穴」や，「シャフト」といった部品の機能素である部分形状の集約である[3]．

組立構造において，サブアセンブリは部品の集約であり，部品が存在しなければサブアセンブリは存在し得ない．同様に部品も部分形状なしでは存在し得ないので，サブアセンブリのインスタンス生成時には部品のインスタンスを生成しなければならないし，部品のインスタンス生成時には部分形状のインスタンスを生成しなければならない．そこで，階層の最上位クラス(設計物を表すサブアセンブリ)のインスタンス生成時に再帰的に最下位クラスのインスタンスまで自動生成するようなメカニズムを導入している．

2.2 組立構造における結合表現の伝搬

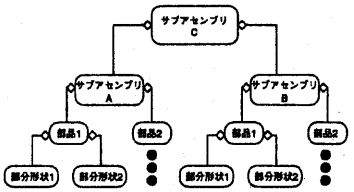

図4　組立構造の階層表現

設計者がサブアセンブリ間の結合を考えるとき，サブアセンブリは1つの部品として扱われるため，サブアセンブリ間の結合は部品間の結合とみなされる．つまり，図3(a)の結合は，サブアセンブリAの穴1とサブアセンブリBの穴2とをネジ1で結合するというふうに考えられている．ここで，部品も部分形状の集約なので，一つの部分形状と考えると，穴1と穴2とをネジ1で結合するとみなすことができる．このことから，部分形状の持つ結合表現が部品へと伝搬され，さらにサブアセンブリへと伝搬されていることが分かる．つまり組立の構造と逆に伝搬されているのである．

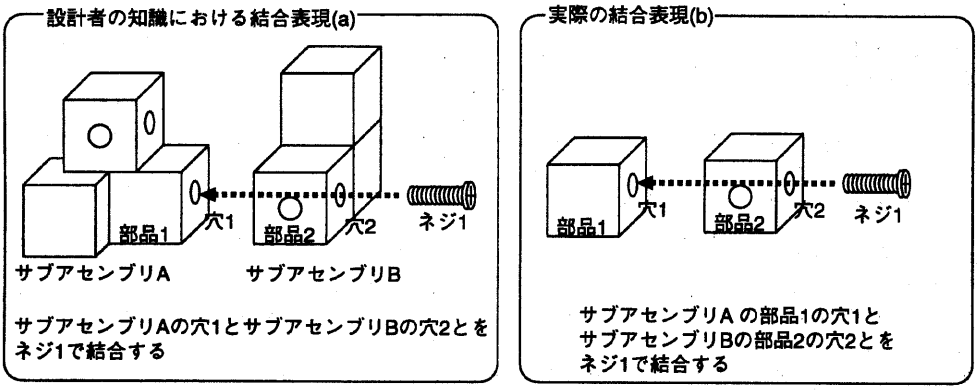

図3　設計者の知識における結合表現 (a) 及び 実際の結合表現 (b)

しかし，通常のオブジェクトモデルでは，このような概念は表現できず，全ての結合表現を部分形状クラスあるいは部品クラスに記述しなければならない (図 3(b)) のでプロダクトモデルの構造が複雑になり，理解性が低下する．

そこで我々は，この結合表現の伝搬を組立構造の集約概念に対して融合し，プロダクトモデル記述言語 (PMDL) に反映させることにより (3.2節参照)，設計者に対するプロダクトモデルの理解性を向上させている．

2.3　設計ドメインにおける継承

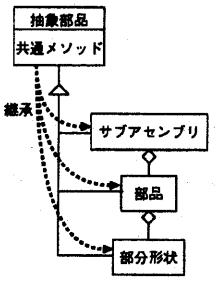

図5　オブジェクト指向におけるメソッドの継承

設計プロセスにおいて，サブアセンブリは，部品と同様に扱われる．従って幾何情報などの属性に対する操作 (メソッド) には，共通なものも少なくはない．これらをオブジェクトモデルで一般化すると図 5のようになり，この共通なメソッドは基本クラスである抽象部品クラスで定義し，それを派生クラスであるサブアセンブリや部品，部分形状が継承する．しかしこの上下関係は設計物の階層構造とは異なっている．

プロダクトモデルは設計者により直接記述されることを前提としているので，メソッドの継承による上下関係と組立構造 (集約) による上下関係とが混在するのは設計者にとっては複雑でありモデルの理解性が低下する要因

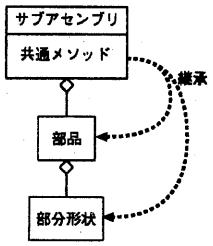

図6　プロダクトモデルにおけるメソッドの継承

となっている．そこで我々は，設計プロセスにおいて，幾何情報の操作や設計物の強度計算などは組立構造の階層により再帰的に求められることがほとんどであるということに着目して，図 5の抽象部品クラスから各派生クラスへのメソッドの継承を組立構造と同じにするために，(図 6) 抽象部品の役割をサブアセンブリクラスに代行させるようにした．これにより組立構造における上位のサブアセンブリクラスで定義したメソッドがその下位クラスに継承される．

継承概念は集約概念とは異なり情報は上位から下位へと継承される．

3.　プロダクトモデル記述

プロダクトモデルは属性モデル[4)5)] の概念に基づいたオブジェクト指向モデルであり，組立構造を階層的に表現し，部品間の結合などを知識として表現する設計物中心のモデルである．設計者は PMDL を用いてプロダクトモデルを記述し，プロダクトモデル記述言語処理系 (PMDL-Processor) はプロダクトモデル記述により定義されたクラスのインスタンスを生成する．生成されたインスタンスは設計支援システムにより共有データとして利用する．(図 2参照)

ここではプロダクトモデル記述言語による組立構造表現，結合表現について述べ，実際の設計物に対する記述

例を挙げる．

3.1 組立構造表現

設計物の組立構造は，オブジェクトモデルによりクラスによる階層構造(図4)で表現される．サブアセンブリはそれを構成する部品の集約であり，部品は部分形状の集約として表現される．サブアセンブリ，部品，部分形状はPMDLによりクラスとして定義され，そのインスタンスは言語処理系により生成される．

クラス記述

プロダクトモデルにおけるクラス記述形式を図7に示す．クラス定義において，クラスタイプ，クラス名，上位

```
<class type> <class name> : <parent class> {
element:
  <class type> <instance>([<attribute>,..]);
attribute:
  attribute(<attribute type>, <default>);
connection:
  connection(<screw>, [<obj1>, <obj2>]);
method:
  <method name>(<arg1>,<arg2>,...); }
```

図7 クラス記述形式

クラス名を記述する．<class type>はaubassembly, part, sub_partのいずれかであり，<parent class>は自分を構成要素として持つクラス名である．クラスの上下関係は，図4で示した階層構造に反映される．

element以下では，構成要素である下位クラス(サブアセンブリ，部品，部分形状)，属性を記述する．下位クラスの宣言ではそのインスタンス名，属性およびその属性値を記述する．属性はデフォルト値を持ち，インスタンス生成時に属性値が指定されていない時にはその値が使用される．

結合表現はサブアセンブリにのみ含まれ，その構成要素の間の結合関係として記述される．

method以下では，座標計算，強度計算などで必要とされる計算を記述する．

インスタンス

設計支援システムは，定義されたクラスのインスタンスを共有データとして動作する．インスタンスの生成はトップダウンで自動的に行なわれるが，これはクラス定義時に記述した構成要素(下位クラス)の宣言を階層的に検索することにより行なわれる，このとき，上位クラスの構成要素に対して指定した属性値が下位クラスのインスタンスに反映される．

メソッド

設計において，座標計算や強度計算などメソッドがあるがこれらのうち，階層間で共通なものはプロダクトモデルにおける継承概念により下位クラスに継承される．したがってこれらのメソッドは最上位クラスの定義時に記述される．また，前節で述べた集約概念により上位クラスは下位クラスのメソッドを参照する．

3.2 結合表現

図8 結合表現伝搬概念の導入による結合表現の変換

前節で説明した結合伝搬のメカニズムを導入した結合表現は構成要素間の関係として上位クラス中に記述されている．つまりサブアセンブリクラスの中では，結合はサブアセンブリ間，あるいは部品間の結合として表現されている(抽象結合表現)この記述は言語処理系により組立構造の階層間を下向きに検索し，サブアセンブリ間の結合に使われる部分形状を見つけることで部分形状間の結合(具象結合表現)へと変換される(図8)．このとき，プロダクトモデルには部分形状間の結合記述がグローバルに定義されており，その記述からサブアセンブリと部分形状との上下関係を推論することが可能であるので抽象結合表現では部分形状名が省略可能される．

3.3 プロダクトモデル記述言語による記述例

ここで，上述したプロダクトモデル記述言語による記述例を挙げる．このモデルを研究するに当たって，まず簡単なエンジンのモデル化に取り組んでいる．ここでは図9に示すシリンダとシリンダヘッドの結合を含むエンジンを例に挙げプロダクトモデルによる組立構造をモデル図及びその言語記述について説明する．

図10はエンジンの部品構造を示している．エンジン(サブアセンブリ)はシリンダヘッド部(サブアセンブリ)と，シリンダ部(サブアセンブリ)，六角ネジ(部品)からなり，シリンダヘッド部はシリンダヘッド(部品)から，シリンダ部はシリンダ(部品)からなる．さらにシリンダヘッド，シリンダは穴(部分形状)を持ち，六角ネジはネジ山(部分形状)を持つ．以下ではエンジン及びその構成要素のクラス記述について説明する．

エンジンのクラス記述

このエンジンの構造をプロダクトモデル記述言語を用いて記述すると 図11のようになる．サブアセンブリ・

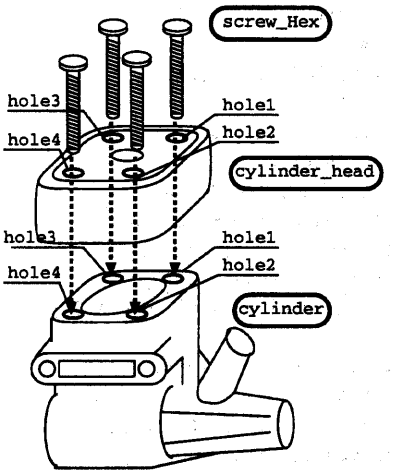

図9 エンジンのシリンダヘッドとシリンダの結合

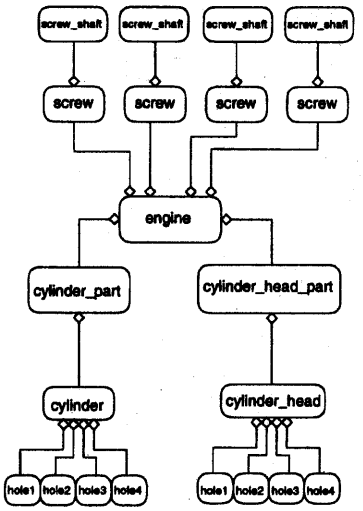

図10 エンジンの組立構造

エンジンは構成要素として cylinder_part, cylinder_head_part, screw を持つ. cylinder_part, cylinder_head_part はそれぞれ cyl_p1, cyl_head_p1 というインスタンス名であり，属性として座標値を設定している．

エンジンの属性としてはグローバル座標系における位置 (origin), 向き (vector), サブアセンブリ内での基準座標系 (サブアセンブリの構成要素はこの座標系をグローバルベクトルとみなす．)(cordinate) がある．これらはデフォルト値をもっており，インスタンス生成時に属性値の指定がない属性に対してはこのデフォルト値が適用される．例えば エンジンの向きはデフォルトではグローバル座標系の z 軸の方向となる．結合記述は，ネジ1～ネジ4を用いてシリンダヘッド部とシリンダ部と

```
subassembly engine {
element:
  cylinder_part cyl_p1([[origin,[0,0,40]]);
  cylinder_head_part cyl_head_p1([[origin,
                                   [0,0,100]]);
  screw screw1([[diameter,3],[length,20]]);
  screw screw2([[diameter,3],[length,20]]);
  screw screw3([[diameter,3],[length,20]]);
  screw screw4([[diameter,3],[length,20]]);
attribute:
  attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]);
  attribute(cordinate,[[1,0,0],[0,1,0],[0,0,1]]);
connection:
  connection(screw1, cyl_p1, cyl_head_p1);
  connection(screw2, cyl_p1, cyl_head_p1);
  connection(screw3, cyl_p1, cyl_head_p1);
  connection(screw4, cyl_p1, cyl_head_p1);
method:
  vector GlobalVector(globalVec,
        CHILD.GlobalVector(vector,
                           PART.vector)); }
```

図11 エンジンのクラス定義

を結合することを定義しており，この結合記述は言語処理系により，具象結合表現に変換される．メソッドとしては，グローバル座標計算を行なう GlobalVector がある．このメソッドはエンジンが持つ部分形状のローカルベクトルをグローバルベクトルに変換するものであり，階層間で再帰的に求められる．

```
subassembly cylinder_part : engine {
element:
  cylinder cyl1([vector, [0,1,0]]);
attribute:
  attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]);
  attribute(cordinate,
          [[1,0,0],[0,1,0],[0,0,1]]); }
```

図12 シリンダ部のクラス記述

エンジンの構成要素であるシリンダ部の記述は図12のようになる．構成要素としては部品シリンダを持つ (実際にはピストン，キャブレタなど色々な部品を持ち，それらの間の結合も定義してあるが，ここでは説明しないので省略してある)．シリンダ部もメソッド GlobalVector を参照するが，エンジンから継承されるので記述されていない．シリンダの定義は図13のようになる．構成要素は穴1～穴4であり，その属性として穴の径，長さ，向きを設定している．部分形状・穴のクラス記述は図14である．穴は，どの部品からも参照し得るので親

```
part cylinder : cylinder_part {
element:
  hole hole1([[diameter,3],[length,10],
              [vector,[0,0,-1]);
  hole hole2([[diameter,3],[length,10],
              [vector,[0,0,-1]]);
  hole hole3([[diameter,3],[length,10],
              [vector,[0,0,-1]]);
  hole hole4([[diameter,3],[length,10],
              [vector,[0,0,-1]]);
attribute:
  attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]);
  attribute(cordinate,
            [[1,0,0],[0,1,0],[0,0,1]]); }
```

図13 シリンダのクラス記述

```
sub_part hole : UNKNOWN {
attribute:
  attribute(diameter, 4);
  attribute(length, 8);
  attribute(screw_pitch, 0.1);
  attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]);
method:
  vector GlobalVector(parent.vector,vector); }
```

図14 穴のクラス定義

クラスを明示していないが，インスタンス生成時に自動的に親クラスが設定される．属性としては，穴の径，長さ，向きのほかに，ネジ山のピッチがある．メソッドGlobalVectorは，他のクラスとは異なり，部品に対する相対ベクトルを返すので，ここで再定義している．部品・ネジ，部分形状ネジ山のクラス記述は図15である．構成要素としてネジ山を持っており，ネジ山は直接構成要素として宣言されないので径，長さ，ピッチなどは，ネジの属性を継承する．実際に部分形状と結合するのはネジ山だが，記述の理解性などからサブアセンブリ内の結合表現は，ネジと部品(またはサブアセンブリ)が結合するように記述し，言語処理系で部分形状の結合に変換する．

4. 組立図生成ツール

我々はプロダクトモデルを用いた設計支援システムの例として，組立図生成ツールを試作している．この組立図生成ツールはプロダクトモデルにより定義されたサブアセンブリの結合表現から組み立ての指示を行なうツールであり．設計者がサブアセンブリを指定すると，そのサブアセンブリ内の結合表現から，サブアセンブリ内の全ての組立図を生成する．

```
part screw : UNKNOWN {
element:
  screw_shaft screw_shaft1([]);
attribute:
  attribute(diameter, 4);
  attribute(length, 8);
  attribute(screw_pitch, 0.1);
   attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]);
  attribute(cordinate,
      [[1,0,0],[0,1,0],[0,0,1]]); }

part screw_shaft : screw {
attribute:
  attribute(origin,[0,0,0]);
  attribute(vector,[0,0,1]); }
```

図15 ネジ(部品)とネジ山(部分形状)のクラス定義

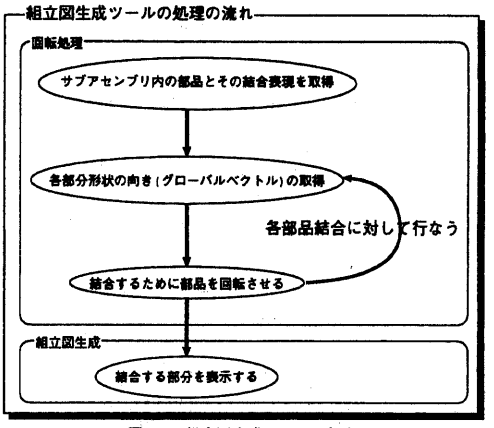

図16 組立図生成ツールの処理

組立図生成ツールの処理の流れは図16の通りであり，以下ではこのプロセスの流れにに沿って説明する．

4.1 回転処理

回転処理は，サブアセンブリから結合表現を抽出し，結合表現に含まれる各部分形状のグローバルベクトルを求め，結合する部品間のベクトルの向きの違いから部品を回転させるといった一連の処理の流れにより行なわれる．

結合表現の抽出

まず，システムは，PMDLにより記述されたサブアセンブリクラス及びその下位クラスのインスタンス(言語処理系により生成される)をリポジトリから抽出し，そのデータからサブアセンブリに含まれる結合表現を階層的に取得する．サブアセンブリの構成要素としてサブアセンブリが含まれるとき異なる階層の結合表現が存在するが，そのような場合，結合表現は階層の下から上に向かって順番に処理される．

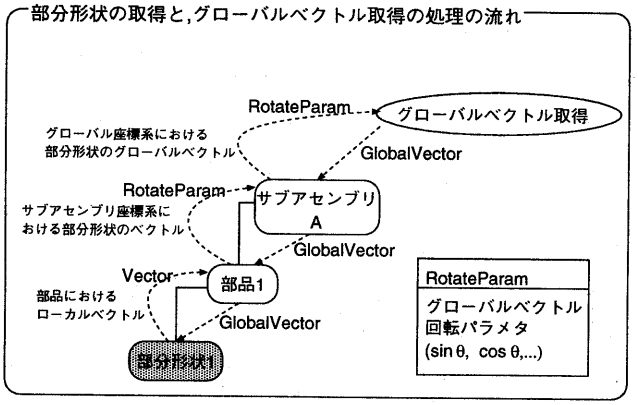

図17　グローバルベクトル取得

グローバルベクトルの取得

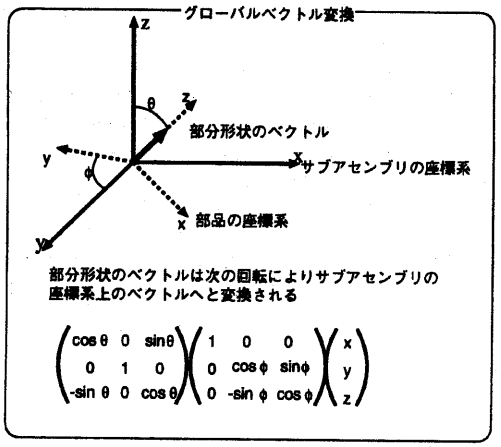

図18　グローバルベクトル変換

サブアセンブリのインスタンスに含まれる結合表現から，サブアセンブリの部分形状を階層を下に向かって検索し，その部分形状のグローバルベクトルを取得する(図17)．グローバルベクトル取得プロセスは，まずサブアセンブリにグローバルベクトルを要求し，サブアセンブリは部品へ，部品は部分形状へとそのローカルベクトルを要求し，グローバルベクトル変換を適用する．

グローバルベクトル変換

グローバルベクトル変換は，サブアセンブリの座標系と部品の座標系の差から，部分形状のベクトルをサブアセンブリの座標系をグローバル座標系とするグローバルベクトルへと変換する．(図18) この処理を階層的に繰り返すことにより，サブアセンブリ階層が深い設計物に対してもグローバルベクトルが取得できる．

回　　転

回転プロセスは，グローバルベクトル取得プロセスが求めた各部分形状のグローバルベクトルの向きの違いから，部品の回転パラメタを求め部品を回転させる．(図19)

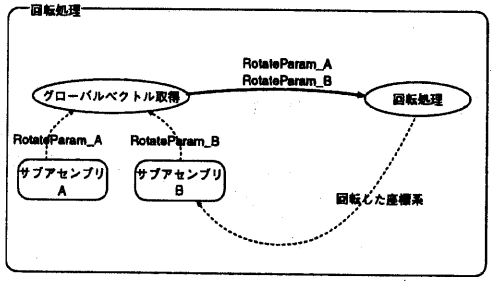

図19　回転処理

この処理を全ての結合に対して行なうと，システムはサブアセンブリの組立図を生成する．

4.2　組立図生成

回転処理が終了すると，システムは組立に必要な部品の3次元幾何情報を抽出する．この幾何情報は部品の属性として部品クラスに含まれている．シリンダの幾何情報は図20のようにCADにより入力されたものであり，幾何プリミティブとしてプロダクトモデル記述に変換されたものが属性としてモデル内に保持される．

エンジンの組立に必要な部品の情報を全て取得すると，それらの幾何プリミティブを回転処理で求めた通りに回転させる．更にサブアセンブリで指定された各部品の位置情報から，部品の位置を決め描画する．全ての部品を描画し終えると，結合の方向を矢印で指示する．この組立図生成ツールを用いて前節でクラス定義をしたエンジンの組立図を生成すると図21のようになる．

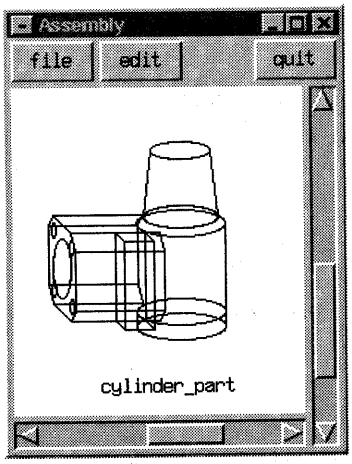

図20　シリンダの幾何情報

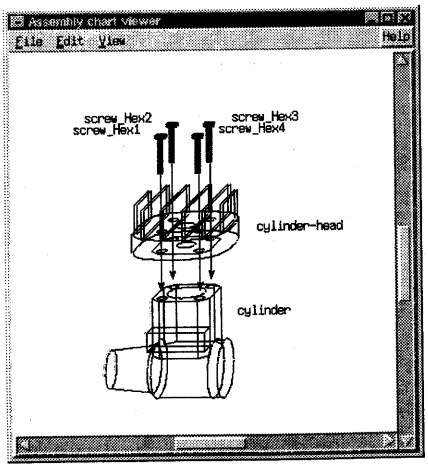

図21　エンジンの組立図

5. おわりに

本研究では現存する機械設計ドメインにおける問題点を解決することを目的として，知的設計支援を実現するための設計物中心モデルとしてプロダクトモデルを構築した．またプロダクトモデルを用いた設計支援システムの例として組立図生成ツールを試作している．プロダクトモデルの構築に当たって，オブジェクトモデルの概念に加え機械設計に特化した概念やメカニズムを導入した．

- 組立構造における部品の集約に対するインスタンス生成
- 設計ドメインにおけるメソッドの継承
- 組立構造における結合表現の伝搬

現在は，幾何情報を手動でプロダクトモデル記述へ変換しているが幾何情報をプロダクトモデル記述として表現するためにCADツールとの連携が必要である．また，プロダクトモデルを用いて他の設計支援システムを構築したり，他のモデルにおける概念を導入したすることでプロダクトモデルの拡張を行なっていく．

参考文献

1) Toyohiko Hirota, Masaaki Hashimoto, and Isao Nagasawa. A discussion on conceptial model description language specific for an application domain. *Trans.IPS.Japan*, 36(5):1151–1162, 5 1995. (in Japanese).

2) Yotaro Hatamura. *Practical Design*. Nikkan kogyo Shinbun-sha, 1988. (in Japanese).

3) John R. Dixon and John J. Cunningham. Research in designing with features. In D. eds. Yoshikawa, H. Gossard, editor, *Intelligent CAD, I*, pages 137–148. North-Holland, Amsterdam, 1989.

4) Masamitsu Mochizuki, Isao Nagasawa, and Masanobu Umeda. A knowledge representation language for tolerance analyses and its programming techniques. *Trans. IPS.Japan*, 35(9):1922–1934, 9 1994.

5) Masamitsu Mochizuki. *Knowledge Representation and Inference Mechanism for Tollerant Analysis*. PhD thesis, Graduate School of Computer Science and System Engineering, Kyushu Institute of Technology, 1996.