動的構造変更機能を有する
基本設計支援用言語に関する研究

今村　聡

目次

1 序論 1

1.1 研究の背景 1

1.2 研究の目的 3

1.3 研究のアプローチ 3

1.4 論文の構成 5

2 研究の位置づけ 7

2.1 設計用語の定義 7

2.2 パラメトリック／コンフィギュレーション設計の数理的定義 9

2.3 設計の分類 10

2.4 関連研究 12

2.4.1 概念設計支援 12

2.4.1.1 ボンドグラフ 12

2.4.1.2 コンフィギュレーション空間の応用 12

2.4.1.3 定性物理モデル 13

2.4.1.4 機能展開 13

2.4.1.5 その他の手法 15

2.4.2 基本設計支援 15

2.4.2.1 アセンブリ設計 15

2.4.2.2 自動設計（設計エキスパートシステム） 16

2.4.3 パラメトリック／コンフィギュレーション設計 18

2.4.3.1 パラメトリック形状モデリング 18

2.4.3.2 パラメトリック設計 18

2.4.3.3 コンフィギュレーション設計 19

2.4.4 設計言語 19

2.4.5 創発型設計 20

2.5 設計研究における課題 20

2.6 パラメトリック／コンフィギュレーション設計に有効な研究アプローチ 24

2.7 FDL 設計言語の位置づけ 28

2.7.1 本研究のとった研究アプローチの理由 28

2.7.2 FDL の特徴と従来研究との比較 28

2.8 2章のまとめ 30

3 基本設計支援用言語の概要 32

3.1 FDL-I 32

3.2 FDL-II 35

3.3 3章のまとめ 41

4 制約対象指向言語ＦＤＬの基本機能 42
4.1 FDL のクラス定義の概要 42
4.2 対象指向言語としての機能 43
4.2.1 データとその操作手続きのモジュール化 43
4.2.2 継承関係による階層構造(taxonomy) 44
4.2.3 部分と全体の関係による階層構造(partnomy) 45
4.2.4 クラスーインスタンス 45
4.2.5 メソッドとブロードキャスティング 46
4.2.5.1 クラス class 46
4.2.5.2 メソッドの記述形式 46
4.2.5.3 クラスメソッド 47
4.2.5.4 インスタンスメソッド 48
4.2.5.5 ブロードキャスティング 48
4.3 論理型言語としての機能 49
4.4 制約言語としての機能 50
4.4.1 単一属性制約 49
4.4.2 同一性制約 51
4.4.3 多属性制約 52
4.4.4 コマンドファイル 54
4.4.5 生成検証法 55
4.4.6 制約解決支援のための why 機能 57
4.5 モデルのアセンブリ 58
4.6 関係データベース機能 58
4.7 インタプリタ 61
4.8 構造化スプレッドシート型インタフェイス 63
4.9 FDL による設計手順 64
4.10 簡単なモデリングの例 65
4.10.1 等分布荷重を受ける片持ちばりのモデル記述 65
4.10.2 FDL による実行 68
4.11 FDL の問題点 70
4.12 4章のまとめ 71

5 動的構造変更機能の導入 72
5.1 メタ操作の導入 72
5.1.1 部品スロットの追加 73
5.1.2 パラメータスロットの追加 73
5.1.3 スロットの削除 73
5.1.4 単一属性制約の追加／更新 75

5.1.5 多属性制約の追加／削除 75
5.1.6 ローカル述語の追加／削除 75
5.2 構造融合操作 75
5.2.1 構造融合の定義 76
5.2.2 構造融合の意義 77
5.3 構造交換操作 81
5.4 アセンブル操作 83
5.5 動的オブジェクトの管理機構 85
5.5.1 was_a リンク 86
5.5.2 was リンク 87
5.5.3 米国特許との比較 87
5.6 配置演算子 90
5.7 動的構造変更のためのグラフィックインタフェイス 91
5.8 ５章のまとめ 93

6 例題による評価 94
6.1 旋盤設計（構造固定） 94
6.1.1 旋盤設計の概要 94
6.1.2 FDL による旋盤設計（構造固定） 97
6.2 旋盤設計（構造可変） 100
6.2.1 ユーザ操作による構造構築 101
6.2.2 配置演算子による自動構造構築 105
6.3 エレベータ設計 107
6.3.1 エレベータ設計問題の概要 110
6.3.2 要求仕様 117
6.3.3 パラメータ値の生成と検証 117
6.3.4 設計計算結果 118
6.3.5 考察 118
6.4 実験評価 120
6.5 ６章のまとめ 123

7 結論 124
7.1 言語の特徴 124
7.2 パラメトリック／コンフィギュレーション設計の課題への対応 125
7.3 今後の課題 127

参考文献 128
発表目録 136
謝辞 139

付録

1　FDL の文法　140
2　FDL 演算子順位表　143
3　FDL の内部表現　144
4　FDL による旋盤設計モデル記述　148
5　FDL によるエレベータ設計モデル記述　165

第1章　序論

本章では、**1.1** で本論文の研究背景を概観したあと、**1.2** でコンフィギュレーション設計とパラメトリック設計に関わる様々な課題の解決を研究目的として設定し、**1.3** で目的を達成するための本研究アプローチを示す。

1.1　研究の背景

機械設計は、要求仕様（機能）から機能の実現手段（実体）へのマッピングの行為ととらえることができ、そのための理論として一般設計学[Yoshikawa, 79, 81]、公理的設計法[Sue, 88]などが構築された。これらの設計論は設計の本質を理解する上で重要な役割を果たしたが、実際の設計一般に適用可能な強力な設計方法論を提供した訳ではなかった。一方、実際の設計支援をターゲットとした設計方法論の研究は、実用性が高いという強みはあるが、対象分野、設計段階に依存する傾向が強く、個別的で適用範囲が限られるものが多い。

ここでは機械システムを主な対象とし、設計方法論、設計技術を簡単に整理してみることにする。設計過程は概念設計、基本設計、詳細設計の設計段階から構成されると考えられている[Arbab, 91], [Hattori, 87], [Nagae, 87], [Yokoyama et al., 97]。これらの各設計段階における技術内容は大きく異なり、計算機支援には、それぞれ異なるアプローチが必要になると考えられる [Finger & Dixon, 89], [Bracewell et al., 95]。

概念設計の内容は、対象によってかなり異なる。要求仕様もその実現手段も明確に与えられない建築、意匠設計などの分野では、それらを明確にする作業が行われる[Tweed & Bijl, 90]。一方、要求仕様が要求機能として明示される場合は、要求機能に対応する実現手段の概略構造を構築していく作業が行われる[Chakrabarti & Bligh, 94], [Kannapan & Marshek, 91], [Kota, 91], [Rao & Lu, 93], [Ulrich & Seering, 89]。これは機能設計と呼ばれ、機能の明示的なモデリングが必要となる[Winsor & MacCallum, 94], [Tomiyama et al., 93]。

詳細設計は、設計対象の製造が可能なレベルまで詳細を決める段階で、製造関連情報の扱いが重要になる[Arbab, 91]。製品モデルの詳細データ構築[Kimura et al., 86, 87]、設計図面の出図[Anderl & Mendgen, 95]などが行われる。

本研究は基本設計支援を主な対象としているので、ここでは、基本設計についてより詳細に検討する。基本設計は、要求仕様を満たす実現手段を定量的に評価し、実現手段の詳細を決める段階である。機械設計では、部品構成と機能に関係するパラメータのほとんどがここで決定される。シミュレーション、コンフィギュレーション設計（configuration design）、パラメトリック設計（parametric design）の三つの計算機支援方法が基本設計に対して多く用いられている。このうち、コンフィギュレーション設計及びパラメトリック設計は、設計解を求めるための技術であるが、シミュレーションは設計解の解析評価技術である。

シミュレーションは設計対象の基本構造を与えて、その挙動を求める技術である。構造解析、機構解析、流体解析、電気電場解析、塑性変形解析、論理解析など様々なシミュレ

ーション技術がある。集中定数系シミュレーション（機構解析、論理解析、電気回路解析など）や、線形分布常数系シミュレーション（電場解析など）では高い解析精度が期待できるが、非線形分布常数系シミュレーション（流体解析、塑性変形解析など）は現在でも解析精度の確保が困難な問題である。また、シミュレーションは設計対象物の評価に用いる技術であり、設計解そのものを求める能力はない。知識処理システムと組み合わせたり、可変形状モデルと組み合わせて最適形状を求めるなど、設計解の自動探索を行う方法も追求されているが（逆問題）、依然研究途上の課題である。

コンフィギュレーション設計は、あらかじめ与えられた構成要素の集合から、制約を満たす組合せを求めるもので、二つのレベルがあると考える。第１レベルは機械の構造は固定されており、構成部品の選択のみ行う設計、第２レベルは機械の構造、構成部品ともに決まっておらず、両者を並行的に決めていく設計である。従来のコンフィギュレーション設計システムの多くは第１レベルのものにとどまっており[Brown, 93]，[Marcus, 88]，[MacCallum, 92]、第２レベルをサポートできるものは多くない。1980年代に開発された設計エキスパートシステムのいくつかは第２レベルのコンフィギュレーション設計を行っているともいえるが[Dixon, 84]，[Mittal, 86]、これらもあらかじめライブラリに登録されたサブアセンブリレベルのモデルを選択することにより構成の変更を行っているのであって、真に自由に構造を変更することはできていない。最近は、遺伝アルゴリズムを適用して、第２レベルのコンフィギュレーション設計を試みる例が見られる[Bennett, 97]，[Skalak et al., 98]。しかし、複雑な構造を持つ対象のストリング列による表現が個別的な工夫に頼っており、対象の設計評価の自動化が難しいなどの理由から、適用例は限られている。

パラメトリック設計は、対象の特性を表すパラメータ集合とそれらの関係を表す数式、論理式により定義されるパラメトリックモデルを用いて行う設計である。第１レベルのコンフィギュレーション設計に類似しているが、集合からの選択に依存しない値の決定も行われる点が異なる。パラメトリック設計は、あらかじめ適切な設計式を登録しておけば、設計対象の構造が確定している定型的設計を効率的に処理することができる[MacCallum, 87]，[Nagasawa, 84]，[Nagasawa, 86]，[Yokoyama, 89]。しかし、パラメトリックモデルはモデル構造、パラメトリック間の関係式を必要に応じて動的に変更する仕組みを一般には持たないので、設計対象の構造が未確定ないし変化する可能性のある設計への適用は困難である。

以上述べたように、コンフィギュレーション設計、パラメトリック設計においてモデルの柔軟性の欠如が問題になっているが、同様な問題はコンピュータ言語一般にもいえる。現在のソフトウエアの構造・アーキテクチャでは、ソフトウエアモジュール同士がお互いに複雑な関係を持ったまま部分的な変更を行うと、その影響が計り知れないため、ソフトウエアの開発、変更、メンテナンスに多くの人手、期間、コストが費やされることが問題になっている。工業技術院ではこの問題を解決することを目的として、産業科学技術研究開発「新ソフトウエア構造化モデル」プロジェクト（平成２年度～平成９年度）を実施している。本研究は当該プロジェクトのサブテーマ「機械設計におけるエージェントの構成法」として実施された。このテーマでは、巨大ソフトウエアの開発、変更、メンテナンスの困難性の解決を、機械設計用言語に対象を限定しつつも、実用性の高い自己再編メカニ

ズム、環境適応メカニズムの開発により実現し、もって新ソフトウエア構造化モデルを実現する上で有効な概念を提供することを目指した。設計モデルを設計言語により記述し、設計モデルの柔軟性を設計言語の柔軟性（動的構造変更能力）により獲得することとした。

１．２　研究の目的

設計を、概念設計、基本設計、詳細設計に分けて、設計支援技術とその課題について概観した。設計は要求機能を満たす構造を求めていく過程であるが、それは直線的に進むとは限らず、設計対象の概略構造がほぼ固まっているべき基本設計段階でも、軽微な条件変更で、構造を大幅に変更する必要が生じることがある。あるいは、概念設計段階で設計対象の概略構造を決定できるとは限らず、機能の定量的検討作業を相当進めてからでないと適切な構造を決定できない場合もある。このような性格を持つ設計に対しては、パラメトリック／コンフィギュレーション設計による設計支援は困難であった。

一方、個別のパラメトリック／コンフィギュレーション設計を見てみると、パラメータ間の関係記述力が不十分、設計対象の記述性が低い、設計解決能力が不十分、設計対象のモジュール構造の組み立てが煩わしいなどの問題がある（**2.5** 参照）。

以上の背景から、本研究の研究目的を次のように設定した。

（１）基本設計支援技術に位置づけられる、パラメトリック／コンフィギュレーション設計技術に関わる以下の課題を総合的に解決する。

(a)パラメータ間の関係記述力

(b)設計対象の記述性

(c)設計解探索能力

(d)設計対象のモジュール構造と組立

（２）設計モデルの構造が未確定ないし動的に変化しうる設計に対して適用できるパラメトリック／コンフィギュレーション設計技術を開発する。

１．３　研究のアプローチ

本研究のアプローチは、**1.2** に掲げた研究目的を達成することが可能な設計支援用計算機言語 FDL(Flexible Design Language)を開発することによる。設計言語は次に示すような技術を導入したものになる。

(1) オブジェクト指向言語

物理実体との対応がとりやすいといわれる、オブジェクト指向言語の形式を採用する。オブジェクト単位でモジュール構造を持たせ、is_a 継承、has_a 関係の効果的な活用により、プログラムの再利用性を高める。データタイプも設計で必要となる多様なタイプを用意する。

(2) 多様な制約表現

オブジェクト間の情報交換に制約伝播を利用する。制約表現は方程式、不等式、ルール（起動条件付き制約記述）、Prolog 述語が使えるものとし、制約表現の多様性を確保する。

また制約の宣言的記述の能力とオブジェクト指向の特徴を生かして、高い記述性を確保する。複雑な制約問題を効率よく解くことのできる制約解決系を開発する。

(3) 関係データベース機能

標準、カタログデータなどの表形式データを扱いやすいように関係データ検索機能を、制約と連動させた形で提供する。

(4) 動的構造変更用メタ操作

設計モデルの構造変更操作を動的に（プログラム実行系の上で、それまでの設計結果の連続性を保ちつつ）行うため、構造共有、構造交換、構造追加／削除などのメタ操作を導入する。構造融合はオブジェクト同士を融合することにより、モデルの組み立て作業を容易にする。構造交換はシャフトや軸受けを異なるタイプのモデルに交換するなど、オブジェクトの交換に使う。また、オブジェクトの追加や、削除をデータ整合性を維持しつつ行うことができる。

(5) 動的オブジェクトの継承関係管理メカニズム

クラス、インスタンスオブジェクトが動的に変更されてもオブジェクト間の継承関係に矛盾が生じないよう、was_a リンク、was リンクを導入する。was_a リンクはインスタンスオブジェクトの動的構造変更により親クラスと対象インスタンスの間の is_a リンクの代わりに使われるもので、構造変更操作の履歴情報を使って、元親クラスからの選択的な情報継承を可能にする。was リンクはクラスオブジェクトの継承情報管理のために使われるもので、メカニズムは was_a リンクと同等である。

(6) 動的構造変更を自動的に行うための配置演算子

起動条件と動的構造変更操作列の組み合わせで定義される配置演算子を導入し、あらかじめ想定できる設計モデルの動的構造変更の自動化を可能にする。

研究のアプローチについては、より詳細な内容が３章で明らかになるので、ここではこれ以上詳しく述べない。ここで掲げた項目のうち(1)－(3)については、従来にもあった技術であるが、特に制約解決については高度な処理能力を確保しており、これらの技術を有機的に統合することにより、高度なパラメトリック／コンフィギュレーション設計解探索能力を得ることができたと考えている。(4)－(6)の項目は、本研究で開発したオリジナル技術であり、設計対象を表現するオブジェクトの動的構造変更を可能にしているものである。設計モデルの動的構造変更は重要な技術であるが、困難でもあるので、従来は簡略ないし抽象的な設計対象モデルを使っているものがほとんどであった[Finger & Rinderle, 89],[Hoover & Rinderle, 87], [Kota & Chiou, 92],[Murthy & Addanki, 87], [Rao & Lu, 93], [Skalak et al., 89], [Tweed & Bijl, 90]。FDL のように大規模で複雑な設計モデルを対象に高度な制約解決能力を提供しながらも、設計の継続性と整合性を保持しつつ設計モデルの動的構造変更を相当自由に行うことができる設計言語ないし設計システムは、関連研究をサーベイした限り、従来は存在しない。1.2 で掲げた研究目的すべての項目を達成した類似研究は存在しないことから、本研究アプローチは有力といえる。

１．４　論文の構成

本論文は次のような構成をとっている。

第２章では、用語の定義、関連研究サーベイを行い、FDL 研究の位置付けを行う。

第３章では、FDL の概要を開発の過程を追って、紹介する。FDL の開発は基本機能を開発した第１期と動的構造変更機能を開発した第２期からなり、第１期に開発したものを FDL-I、第２期に開発したものを FDL-II と呼び区別する。

第４章では、FDL に関する基本機能について述べる。FDL のプログラム書式、制約、関係データベース等の定義方法、設計手順などが紹介される。

第５章では、FDL に導入された動的構造変更機能について述べる。構造融合、構造交換などのメタ操作、動的オブジェクトの継承関係を管理するための was_a、was リンク、状況を判断してオブジェクトの構造を自動的に変更するための配置演算子などを紹介する。

第６章では、旋盤及びエレベータの設計実験例を報告する。旋盤設計については構造固定の場合を **6.1** で、設計の進展に伴って構造の変更がありうる場合を **6.2** で示す。**6.3** に示すエレベータ設計はウエスチングハウス社で実際に行われた設計問題に取り組んだもので、大規模複雑な上にパラメータ間の関係が条件に応じて動的に変化する。

第７章では、本研究の結果を整理し、今後の課題を示す。

図 1.1 に本論文の構成をフローチャートとしてまとめたものを示す。第３章で示した第１期開発の FDL-I が第４章の基本機能につながり、その評価用例題として第６章の旋盤設計（構造固定）とエレベータ設計が取り上げられた。一方、第２期開発の FDL-II は第５章で述べられる動的構造変更が特徴となっており、その評価用例題が第６章の旋盤設計（構造可変）につながる。

付録では、FDL の文法、演算子順位、インプリメンテーションの内容、及び、旋盤設計、エレベータ設計の FDL コードを記載する。

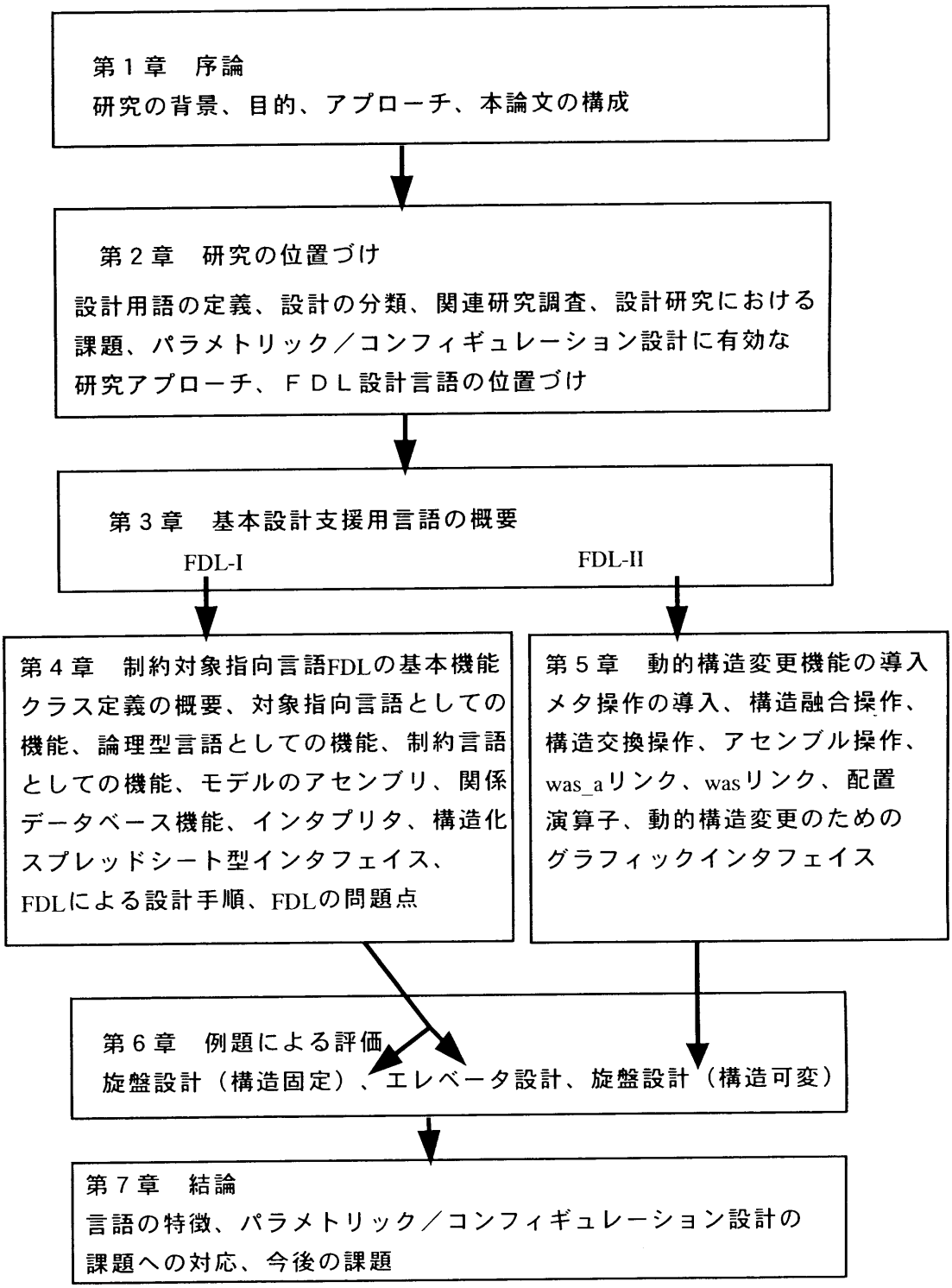


Fig. 1.1 Flow chart of the thesis

第２章　研究の位置づけ

本章では、国内外で行われている設計関連研究におけるFDL研究の位置づけを試みる。その準備作業として、**2.1** では機能、挙動、構造、状態、副作用などの用語を定義し、それらを使って設計行為を再定義し、**2.2** では設計過程を上流から下流に沿って各段階で行われる行為を概観する。**2.3** では本研究が主な対象とする概念設計、基本設計段階で用いられる設計支援技術とその関連研究を整理し、**2.4** で、FDLの研究を位置づける。

2.1　設計用語の定義

設計行為は、機能、挙動、構造などの用語を用いて定義される例が多い。ところが、これらの用語の意味するところは論文により微妙に異なり、統一されていない。そこで、本節では用語を再定義し、それを基に設計行為、設計関連技術の定義を行う。

以下、本論文における設計関連用語の定義を示す。

（１）機能

設計者が意図した、あるいは、目的とした対象物の働き。挙動を伴って発現されるものが多いが、そうとは限らない。例えば、灰皿の機能は灰を保持することであるが、挙動を伴わない。従って、多くの文献が採用する「設計者が意図した対象物の挙動」という定義は、ここではとらない。

（２）構造

対象物を構成する要素すべてとそれらの間の関係。

（３）状態

構造が持つあらゆる物理変数の値が与えられたものを状態と呼ぶ。物理変数とは、力学、電気、熱力、光学、などあらゆる物理現象を表現する変数をさす。計算機上で構造のもつすべての物理変数を表現することは不可能なことが一般的であり、その場合代表的変数で表現する。

（４）挙動

時間遷移にともなって対象物の状態が変化すること。

（５）副作用

意図していない、あるいは、目的としていない対象物の働き。

（６）環境

対象物を取り囲むあらゆる要素。対象物の周囲にある人工物、自然物、人間すべてが対象物の環境である。対象物の機能、挙動、副作用は、環境が与えられることにより求める

ことが可能となる。

（7）要求仕様

対象物の設計の際、対象物が満たさなければならない要求の集合。多くは、機能ないし挙動に関する要求として表現されるが、構造、副作用に関する要求も含まれる。また生産数量、コスト、製造方法も設計（特に詳細設計）に影響を与えることが多く、その場合は要求仕様に含まれる。

（8）基本設計

基本設計は、概念設計段階で決まった要求仕様の実現手段を定量的に評価し、実現手段の機能にかかわる詳細を決める段階である。機械設計では、部品構成と機能に関係するパラメータのほとんどがここで決定される。シミュレーション、コンフィギュレーション設計（configuration design）、パラメトリック設計（parametric design）の三つの計算機支援方法が基本設計に対して多く用いられている。

（9）コンフィギュレーション設計

あらかじめ与えられた構成要素の集合から、要求仕様を満たす組合せを求めるもので、二つのレベルがある。第１レベルは機械の構成（要素間の接続関係）は固定されており、構成要素の選択のみ行う設計である。第２レベルは機械の構成、構成部品ともに決まっておらず、両者を並行的に決めていくものである。

（１０）パラメトリックモデル

パラメータ集合とそれらの関係で構成される設計用モデルである。パラメトリックモデルとパラメータの集合及びそれらの関係で構成される上位のパラメトリックモデル（アセンブリモデル）を定義してもよい。パラメトリックモデルは設計対象のみでなく、要求仕様、使用環境、コスト、リサイクル性、組立分解性など、設計に関わるあらゆるファクターを表現してよい。

（１１）パラメトリック設計

パラメトリックモデルを使って行う設計。設計対象の概略構造がすでに固まっている基本設計段階で用いられることが多い。また、詳細設計段階に相当するが、図面（または３次元モデル）をパラメータ表現し、寸法のバリエーションを与えるだけで図面を完成させる作業もパラメトリック設計と呼んでいる。いずれにしろ、パラメトリック設計は信頼性の高いパラメトリックモデルを何度も再利用することにより、設計効率、設計品質を上げることが目標となるので、主に定型的（ルーチン）設計に適用される。設計が定型的でない場合は、設計変更のたびにパラメトリックモデルの構成、内容自体も変更しなければならないので、適用しにくい。

（１２）自動設計

設計対象モデルと設計解探索アルゴリズムを持つシステムにより、要求仕様を満たす設

計解を自動的に求めること。

（１３）会話型設計

自動設計と異なり、設計者の判断を重視し、設計解探索の自動化を求めない。部分的な設計解探索の結果を必要に応じて表示し、途中設計解の評価、解候補からの選択、別解探索の指示などの判断を設計者に求めることにより、設計システムと設計者が協調して設計を進める。このようなシステムは会話型設計支援システムと呼ばれる。

（１４）シミュレーション

設計対象の構造と初期状態、及び使用環境の情報を与え、数理モデルに基づいた計算により挙動を求めること。設計対象の評価に使うが、シミュレーション自体に設計解を求める能力はない。

２．２　パラメトリック／コンフィギュレーション設計の数理的定義

パラメトリック／コンフィギュレーション設計の定義は前節ですでに与えたが、本研究で中心となる課題であるので、ここではより厳密な数理的定義を試みる。パラメトリック／コンフィギュレーション設計で用いる設計モデルを次のように表現する。

$$A=\{p_1, p_2, \ldots, p_m \mid p_1\in U_1, p_2\in U_2, \ldots, p_m\in U_m, f_1, f_2, \ldots, f_n\} \qquad (2.1)$$

p_i はモデルを表現するパラメータを表し、数、記号、ストリング、あるいは任意の構造をもつデータである。集合U_iの要素でなくてはならない。f_jは$\{p_1, p_2, \ldots, p_m\}$の中から任意の２個以上のパラメータ間の関係を記述した制約で、数式、不等式、述語論理などにより表現される。設計モデルに与えられる設計要求は、いくつかのパラメータ値の指定、集合U_i、制約f_jの指定（変更、追加など）により表現される。

ここで、制約条件$f_1, f_2, \ldots, f_n$を満たすパラメータ$p_i\in U_i$（$i=1, \ldots, m$）を求める問題がパラメトリック／コンフィギュレーション設計問題である。その中で、コンフィギュレーション設計とは$U_1, U_2, \ldots, U_m$がすべて有限集合である場合、あるいは制約との関係で、すべてのパラメータについて有限個の解候補しか存在しない場合をさす。それ以外はパラメトリック設計と呼ぶ。

一般的にパラメトリック設計ではモデル構造の変更（パラメータ集合、制約の変更）は行われない。コンフィギュレーション設計についても第１レベルではモデル構造の変更は行われないが、第２レベルではモデル構造の変更（組立、構成）も行われる。

設計モデルは他のモデルを構成要素として持つ、アセンブリモデルとして表現される場合がある。その場合も、上記の議論がそのまま適用できることを示す。アセンブリモデルは次のように表現される。

$$A = \{p_1, p_2, \ldots p_{n1}, C1, \ldots, Cn2 \mid p_1 \in U_1, p_2 \in U_2, \ldots, p_k \in U_{n2}, f_1, f_2, \ldots, f_k, \ldots, f_{n3}\} \quad (2.2)$$

p_i はモデルを表現するパラメータ、Cj はモデルの構成要素となる下位のモデルを表す。制約は f_k は p_i （$i = 1, \ldots, n1$）に加えて Cj（$j = 1, \ldots, n2$）内に定義されているすべてのパラメータの中から任意の２個以上のパラメータ間の関係を記述する。ここで、Cj（$j = 1, \ldots, n2$）内に定義されるパラメータ及び制約をそれぞれ $p_{n1+1}, p_{n1+2}, \ldots, p_{n1+n4}$ 及び $f_{n3+1}, f_{n3+2}, \ldots, f_{n3+n5}$ と表すことにすると、上記のモデルは次のように書き換えられる。

$$A = \{p_1, \ldots, p_{n1}, \ldots, p_{n1+n4} \mid p_1 \in U_1, p_2 \in U_2, \ldots, p_{n1+n4} \in U_{n1+n4}, f_1, f_2, \ldots, f_k, \ldots, f_{n3+n5}\} \quad (2.3)$$

これは（2.1）式と同形式の表現であり、アセンブリモデルはサブモデルをもたないモデル表現に変換できることがわかる。よってアセンブリモデルにおいても、パラメトリック設計、コンフィギュレーション設計に関する定義を同様に定めることができる。

２．３　設計の分類

要求仕様から、製品製造用図面を作成するまでの過程全体をさす。一般に (1)製品企画、(2) 概念設計、(3) 基本設計、(4)詳細設計の各段階から構成されるが、すべての段階を経るとは限らない。新規製品を開発する場合、製品企画、概念設計の段階が重要だが、設計手法が確立している定型的設計や既存製品の修正設計の場合、これらの段階を省略し、基本設計ないし詳細設計からスタートする。

（１）製品企画

マーケッティング、企業戦略、社会要請などのファクターから製品開発を企画する段階である。出力は要求仕様、製品構想（製品を構成する方法）、販売予測などである。

（２）概念設計

概念設計は、大きく分けて仕様の具体化、機能展開、実現手段の構成の３段階からなる。

(a) 仕様の具体化

要求仕様が、工学的に明確に与えられていない場合、より具体化、明確化する作業が必要となる。例えば、建築物に対する「オープンで明るく心地よい」、自動車に対する「スポーティーでラグジュアリー」といった感性による要求は、実現手段を検討し、より具体的な要求仕様に作り替える。出力は、工学的に、より明確な要求仕様である。

(b) 機能展開

要求機能（要求仕様に包含される部分集合）を直接実現できる物理実体（機構、電子回路など）が存在すれば、それで設計は終了であるが、そうでない場合は要求機能をそれと同等な能力を持つサブ機能の集合に変換する作業を行う。これを機能展開と呼ぶ。機能展

開は機能の木構造を作り出す過程に相当し、リンクでつながった下位の複数の機能が上位の機能を実現するという関係にある。機能展開では解が無数に存在しうる。解の選択は次の段階で行われる「実現手段の構成」の可能性、さらに進んで基本設計段階における要求仕様、性能、コストの定量的評価によって確定される。

(c) 実現手段の構成

機能展開により作られたサブ機能の集合を実現する物理実体の集合を探し、それを組み上げる段階である。サブ機能から物理実体へのマッピングは、機能でインデクシングされた物理実体のライブラリを用意し、そこからの検索により行う。サブ機能－物理実体間マッピングは１対１とは限らず、１対多、多対１、多対多と多様であり、サブ機能集合－物理実体集合マッピングにも無数の解がありうる。この段階では、原理的な実現可能性を検討し、解を絞る。

図 2.1 に機能展開と実現手段の構成の関係を示す。

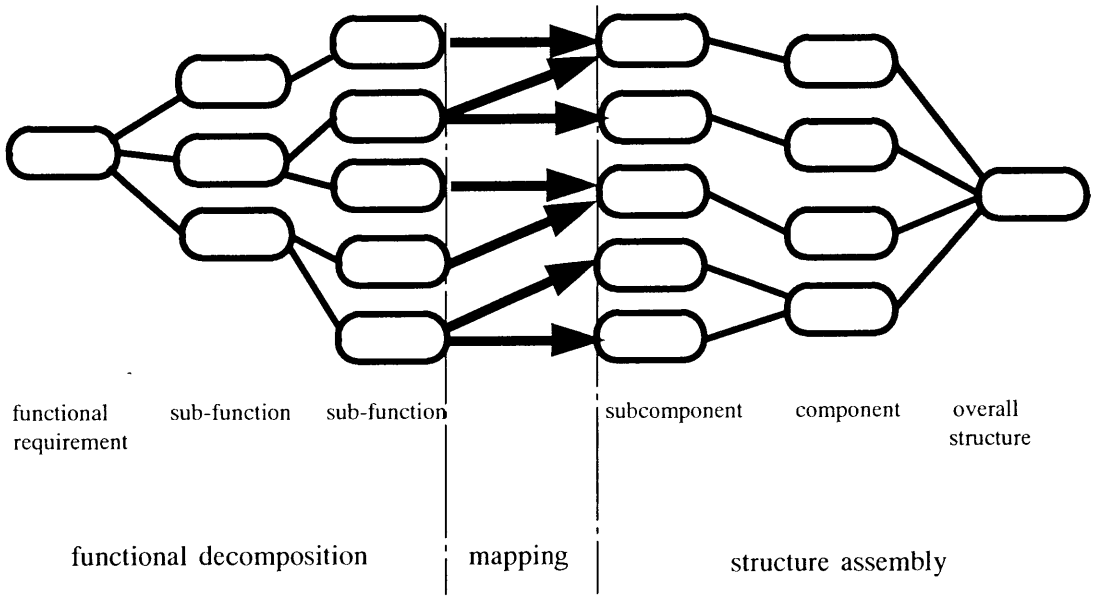


Fig.2.1 Functional decomposition and structure assembly

（３）基本設計

概念設計段階で決定された概略構造に基づき、構造の詳細を定量的に決めていく段階である。主要機能に関わる詳細構造は決まるが、主要機能を支える周辺構造は詳細設計段階に譲る場合が多い。この段階における設計支援技術には、パラメトリック設計、コンフィギュレーション設計、シミュレーション、形状モデルなどがある。設計評価のための試作実験もこの段階で行うことが多い。

（４）詳細設計

基本設計段階で固まってない、主要機能を支える周辺構造の詳細を決めていく。また、対象物を製造するために必要となる情報すべてをこの段階で決定する。対象物の詳細形状以外に、寸法公差、表面荒さ、表面処理などの製造関連情報も決められる。製造個数、製

造設備、コスト、信頼性、環境負荷、組立分解性など、多くの項目が考慮され、設計結果に反映される。

2．4 関連研究

本節では、概念設計支援、基本設計支援、パラメトリック／コンフィギュレーション設計、設計言語、創発設計の５項目をたて、関連研究のサーベイを行う。

2.4.1 概念設計支援

概念設計支援関連研究の関連研究を紹介する。概念設計支援のための有力な手法として、ボンドグラフ、コンフィギュレーション空間による設計対象表現があり、また機能の操作の方法として機能展開が多くとられている。また、これらのいずれのカテゴリーにも属さない概念設計支援の方法も多く提案されている。

2.4.1.1 ボンドグラフ

機械、電気、油圧／空圧、熱、生体などの動的システムを統一的に表現するためのもので、[Paynter, 61]により提案された。ボンドグラフはパワーの伝達に関する基本要素を結合し、パワーの伝達経路のグラフとして対象を表現するもので、電気回路と同等なネットワークとして表現される。ボンドグラフではボンド（エッジ）を通してパワーを伝達し、１ポート要素でパワーの消費、蓄積、供給を行い、２ポート要素でパワーの変換を行い、ｎポート要素によりキルヒホッフの法則と同等の法則を使ってポートの結合を行う。

[Finger & Rinderle, 89]は設計対象をボンドグラフで表現し、要求仕様及び制約を満たすように対象の構成を作り上げていく。歯車箱設計を例題に、歯車ペアを２ポート要素で表現し、最終的に設計制約条件を満たすように減速比を配分し、２ポート要素のシリーズ結合の構成を求めている。[Hoover & Rinderle, 87]も対象表現以外は類似の方法をとっている。[Ulrich & Seering, 89] は１入力１出力の動的システム(dynamic system)を対象に自動設計を試みている。設計要素としては並進機構、回転機構、流体部品、電気部品を扱い、これらを共通の形式（微分方程式またはボンドグラフ）で表現する。設計要求も同様の形式で表現し、設計要求を満たすように設計要素をシリーズに接続したあと、ボンドグラフ（論文では電気回路で表現）領域で表現の簡素化を行った後、よりコンパクトな設計解を導き出す。[Bracewell et al., 95]は、概念設計から部品レベルの詳細設計までをカバーする統合設計システム SchemeBuilder を提案し、その中で点滴システムを例題にボンドグラフを使って設計対象の挙動シミュレーションを行っている。

2.4.1.2 コンフィギュレーション空間の応用

コンフィギュレーション空間 (configuration space) とは、ロボットの経路探索問題や機構の運動解析問題などにおいて利用されている方法で、対象物（機構）が持つ運動自由度

ごとに座標軸を設定したn次元の空間をさす。すると、コンフィギュレーション空間内の一点は、機構を構成する各部品の位置、姿勢の一状態を表すことになる。機構の取りうる状態の集合をコンフィギュレーション空間上に表現することにより、機構の静力学的な運動パターンを表現することができる。

[Murakami & Gossard, 92]は、機構設計問題において実現しようとする運動及び機構ライブラリ内の各モデルの運動をコンフィギュレーション空間上の軌跡で表現し、軌跡のマッチングにより、適切な機構モデルを検索するシステムを提案している。[Joskowicz & Sacks, 91]は、リンク機構及び固定軸機構を対象にキネマティクス（機構の部品間の干渉や相互の動き）解析を自動的に行うシステムを開発した。システムへの入力はソリッドモデル（B-Reps）による形状データ及び初期配置であり、出力は到達可能なコンフィギュレーション空間領域の集合とそれらの隣接関係を表したリンクで、機構が可能な挙動を表現することができる。

2.4.1.3 定性物理モデル

[Tomiyama et al., 89]はメタモデルと呼ぶ定性物理モデルを核に、ソリッドモデル、メッシュモデル、機構モデル、ブロック線図モデルなど設計支援に必要な様々なコンピュータモデルを関係付け、有機的な設計支援環境を構築することを目指すとしている。定性情報のみではモデル間の関係付けは困難であり、後にメタモデルに定量情報を付加している。

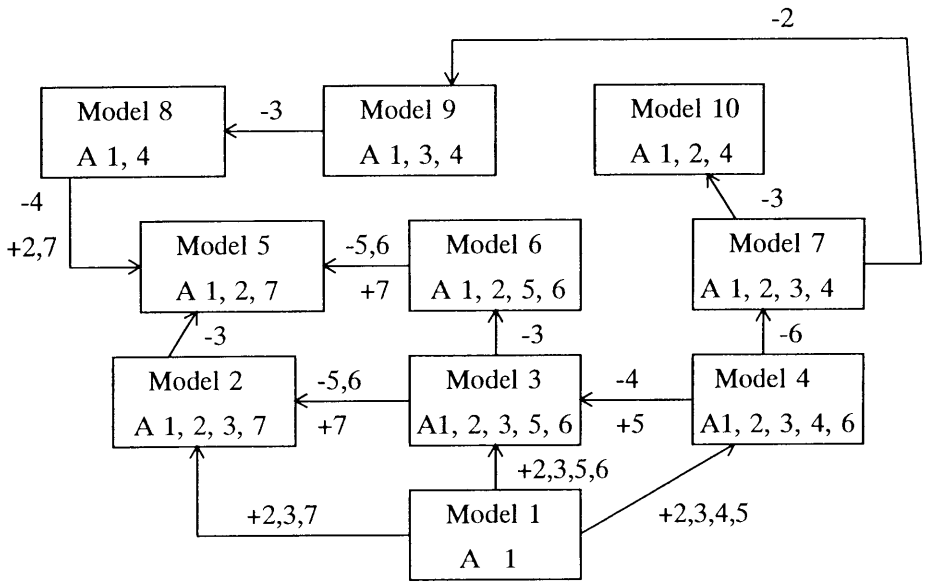
[Umeda et al., 90], [Tomiyama et al., 93]は設計対象物を機能-挙動-状態とそれらの関係で表現する FBS(Function-Behaviour-State)モデルを提案し、機能設計を試みた。設計は FBS モデルで表現した要求機能からスタートし、機能展開することにより、より詳細な FBS モデルを構築し、これに定性推論を適用して必要な挙動が得られるかシミュレーションを行う。設計結果として、対象物の基本構造(basic structure)を出力する。

[Kota & Chiou, 92]は機構のビルディングブロック（歯車ペア、リンク機構、ラックとピニオンなど）の挙動の特性を運動変換マトリックス（並進・回転運動の入出力関係を表現）と運動制約マトリックス（繰り返し、速度可変など、運動の性質を表現）で表現し、要求挙動を満たす機構をマトリックスの積のシリーズとして求める。挙動の定性的シミュレーションが可能で、設計解候補の評価が可能。

[Gupta & Jakiela, 92] は歯車ペアなどの機構対偶部品を微少な分子(molecule)の集まりとして表現し、駆動部品の運動と両部品の輪郭の分子間干渉チェックにより、被駆動部品の運動及び形状の詳細を求めている。機構部品の自動最適形状設計や、汎用性の極めて高い機構シミュレーションへの適用を可能としている。

2.4.1.4 機能展開

機能展開とは、要求機能を部分機能に再帰的に分解していくことであるが、これを研究アプローチに取り入れているものとして、[Bracewell et al., 95], [Hoover & Rinderle, 87], [Kannapan & Marshek, 91], [Tomiyama et al., 93]などがある。機能そのものの表現方法はそれぞれ異なるが、機能をサブ機能に分解していく過程を採用している点で共通点がある。



1 Slender member
2 Constant cross-section
3 Elastic material
4 Circular symmetry
5 Rectangular cross-section(a,b << L)
6 Solid beams
7 Hollow thin-walled shaft

The assumptions are listed under the model name. The equations describing the behaviour of a beam described by Model 7 are listed below .

$$\tau_{\theta z} = G\gamma_{\theta z}$$
$$\gamma_{\theta z} = (d\phi/dz)r = (\phi/L)r$$
$$df = \tau_{\theta z}\, dA$$
$$dM = rdf$$
$$M_t/\phi = (1/L)\int_A rGrdA$$
$$W = \int\int DdA\, dz$$

Fig. 2.2 The graph of models for beams under torsional loads.

2.4.1.5 その他の手法

[Hoover & Rinderle, 87]は、形状－挙動グラフ(form-behavior graph)と呼ぶパラメータ集合で対象を表現し、分解(decomposition)、集成(aggregating)、再配置(redistributing)などの変換により、要求仕様及び構成上の制約を満たすように、対象を構成する。歯車列を例にとり、歯車ペアの構成と減速比の配分を繰り返し計算により行う。

[Tweed & Bijl, 90]は、フレームベースの設計対象記述システム MOLE を提案し、建築設計への適用を試みている。対象表現の自由度が高く、対象表現の逐次的構築、動的変更が容易にできるが、対象間の制約関係を解決するメカニズムがなく、設計解を求める能力に乏しい。概念設計モデル表現システムと位置付けられる。

[Ward & Seering, 89] は、定量推論(quantitative reasoning)という概念を導入し、区間値の演算を定義し、その演算により組み合わせの適性を判断している。カタログからの部品検索と、結合を行い、その結果を区間値演算で評価することを繰り返し、適切な構成を求めることができる。

[Chakrabarti & Bligh, 94] は、システムへの入力、出力の種類（力、速度、変位など)、位置、方向を要求仕様として与え、入力-出力間の変化を達成できる機構の組み合わせをライブラリからの探索と結合により求めている。鍵の機構を例題に、本方式を試みている。

[Neville & Weld, 93]は、設計対象に要求される挙動及びコンポーネントの挙動を互いに影響しあうパラメータ（微分も含む）の集合群で表現し、コンポーネントの結合により、要求挙動と同等のパラメータ集合群をつくることにより、解候補を生成する。真に要求を満たすか否かは、より詳細なシミュレーションを行う必要がある。

[Murth & Addanki, 87]は、梁を例題にモデルの構造変更を自動的に行うシステム PROMPT を開発した。荷重、長さ、断面２次モーメントなどに関する要求に応じて、梁モデルの断面形状や材質を変更するために、Graph of Models と Modification Operators の二つの方法を提案した。Graph of Models は**図 2.2** に示すような様々な梁モデルのネットワークモデルで、要求に応じて適切な梁モデルを探し出すことができる。Modification Operators は応力解析の結果に応じて、質量の移動を行い梁断面の質量分布を変更することができる。PROMPT は適切なモデルを探し出したり、モデルの内容自体を変更する方法について従来にない新しい提案を行ったと評価できる。

2.4.2 基本設計支援

ここでは、アセンブリ設計、自動設計の項目をあげ、関連研究を紹介する。

2.4.2.1 アセンブリ設計

機械アセンブリの設計、組立／分解手順生成、組立性検証などの研究がこの項に属する。コンフィギュレーション設計に属するともいえるが、コンフィギュレーション設計が機能レベルのアセンブリを主要な対象としているのに対し、アセンブリ設計は形状レベルの部

品組立を問題にする。

[Sata et al., 85]は製造のために必要なすべての情報を表現することを目標としたプロダクトモデル研究の一環として、ソリッドモデルを構成する面同士の接続関係を述語で表現した組立モデルを採用し、組立可能性の検証を例示した。

[Kimura et al., 91]は接続関係などの部品間に渡る関係（パタンと呼んでいる）とローカルな詳細形状決定するアルゴリズムを内包した設計特徴を元に、機械アセンブリの形状設計を行うシステムを提案している。設計特徴間の関係は設計制約で表現している。

[Sekiguchi et al., 86] , [Imamura, 97a]はいずれも部品接続関係をはめあいと接触の２種類に整理し、述語の集合で表現したモデルを採用。前者はグラフ探索手法で機械全体の分解手順を生成し、後者はマルチエージェントの協調モデルにより、組立／分解、部品交換の作業手順を生成した。組立性評価の他、メンテナンスへの適用が可能である。

2.4.2.2 自動設計（設計エキスパートシステム）

1980 年代に米国を中心に多数の設計エキスパートシステムが開発された。Ｖベルト設計の V-Belt [Dixon & Simmons, 84]、エアーシリンダ設計の AIR-CYL[Brown, 85]、コピー機械設計の PRIDE[Mittal et al., 86]、 高層ビル設計の ALL-RISE[Sriram, 87]、エレベータ設計の VT[Marcus et al., 88]などである。これらはいずれも、コンフィギュレーション設計技術を取り入れた設計自動化システムである。PRIDE など対象構造の変更も可能なものもあるが、いずれのシステムも設計対象を意識したシステム構成なので、部品選択はあらかじめ登録されている範囲に限定され、大幅な構成変更や想定外の構成変更には対応できない、などの限界がある。

[Dixon, 86]は、汎用性のある設計エキスパートシステムのシステム構成法を提案している。ここでは**図 2.3** に示すように、設計問題を部分問題に展開し、部分問題はさらに分解していくことにより、設計問題を木構造に分解している。部分問題はローカルに問題解決を試み、解決できた場合は上位の部分問題に解を返す。解決できなかった場合は、上位に部分問題への展開の仕方の再検討を促し、新しい部問題のもとで解決を試みる。この部分問題展開の再検討依頼は問題解決がうまくいかない場合、最終的にトップレベルまで伝播する。すべての部分問題展開の候補が試され、解が求まらない場合は最終的に設計は失敗する。**図 2.4** に、Dixon が提示した部分問題解決のためのモジュール構成を示す。この仕組みは、構造変更も可能なコンフィギュレーション設計の汎用的方法を示しており、多くのシステムがこのようなアーキテクチャを取っていると見なせるが、あらかじめ想定した解空間からの探索という限界をかかえており、想定外の設計構成、設計要求への対応は困難である。

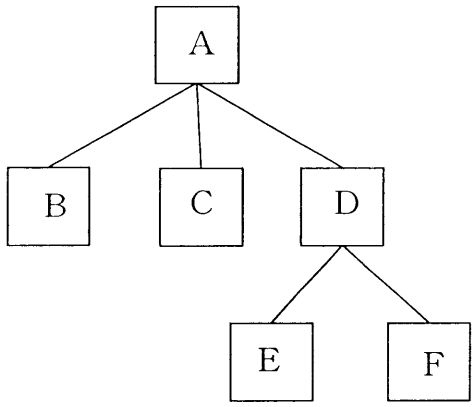


Fig. 2.3 A decomposition model of design

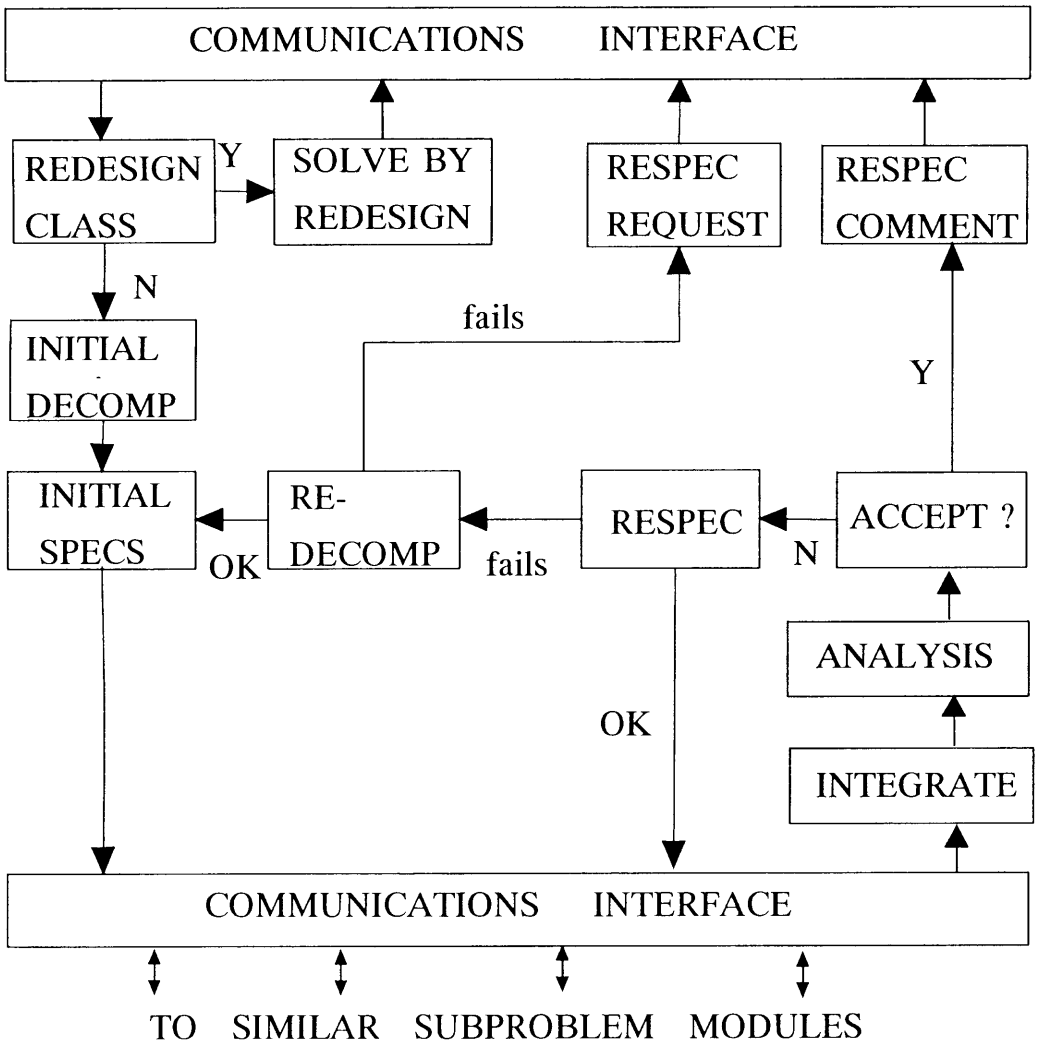


Fig.2.4 A decomposition module

2.4.3 パラメトリック／コンフィギュレーション設計

パラメトリック形状モデリング、パラメトリック設計、コンフィギュレーション設計の研究例を紹介する。ここでとりあげる多くのものが制約処理技術を採用している。

2.4.3.1 パラメトリック形状モデリング

[Anderl & Mendgen, 95]は、2次元、3次元CADへの制約適用についての現状と課題を整理した。3次元CADの多くがフィーチャー（CSGまたは掃引形状）によるモデリング機能を取り入れており、フィーチャーの形状、位置、姿勢のパラメータ指定に制約を用いるのが有効としている。また、スケッチ入力から、平行、直角、円などの制約情報を自動検出することも有効であるが、誤認識などの課題もある。

[Kimura, 87]は、設計要求を反映した制約記述により形状の生成、操作を行える可変形状 (variational geometry)の概念を導入した。制約は論理関係（述語）で記述されており、布製品形状の生成、ソリッドモデルのパラメトリック変更、シートメタルのプロセスプランニングへの適用を例示した。[Shimada et al., 89]も同様に、制約記述による形状モデリングを開発した。形状モデリングはフィーチャーベースで行われ、フィーチャー内及びフィーチャー間に制約定義ができ、制約の変更、追加などが可能である。

多くの市販ＣＡＤも形状モデルのパラメトリック定義、変更機能を有している。CATIA(http://www-6.ibm.com/jp/manufacturing/prod/catia/)、I-DEAS(http://iccon.org/)、Unigraphics(http://www.ugsjapan.com/)、ProEngineerの４大ＣＡＤをはじめとして、CADCEOUS(http://www.unisys.co.jp/CADCEUS/)、AutoCAD(http://www.autodesk.co.jp/)、Solidworks (http://www.kusco.co.jp/solidworks/swindex.htm)などのＣＡＤすべてにおいて、レベル、方法に違いはあるにしろ形状のパラメトリック定義／修正機能を有している。多くはスプレッドシート・ソフトと連動させて、何らかの設計計算の結果を反映させて形状のパラメトリック定義を行うことが可能である。現在では自動車企業などの大手企業では4大CADのいずれかを利用している場合が多く、形状モデリング機能及び解析、加工などの関連ソフトとの連携機能など総合力が重視され、ＣＡＤも集中化、デファクトスタンダード化が進んでいる。

2.4.3.2 パラメトリック設計

基本設計段階における制約適用は、パラメトリック設計システムでの適用例が多い。

[MacCallum & Duffy, 87]は、Designerと呼ぶパラメトリック設計システムを開発した。対象をパラメータ集合{x1,...,xn}とパラメータ間の関数関係 xi=fi(x1,x2,..,xi-1,xi+1,...,xn)により表現した。各パラメータには値、精度、単位、関数関係には信頼度の属性が記述できる。大型船を例題に、概略的な諸元を求める予備設計を行っている。[Higuchi & Nagasawa, 92]は、DSP (Designer's SPread sheet)と呼ぶ表形式の設計計算用ツールを開発した。これはDesigner同様、関数形式でパラメータ間の関係を記述し、生成検証法などにより設計解を探索する方法が取られている。Designer、DSPともにモジュール構造を持たないので、デ

ータ量の大きな設計対象を扱う場合のプログラムの記述性、再利用性に問題があると思われる。

[Kimura & Suzuki, 86]は、設計の要求段階から部品設計の段階に至るまでのプロダクトモデリングを、制約解決により扱う枠組みを提案した。設計要求を制約として与え、設計対象の構成をコンフィギュレーションにより求め、機能に関わる各種設計計算、部品寸法を制約充足により取り扱った。

[Ishiba et al., 95]は、制約解決によるエレベータ設計支援の専用システムを開発した。制約に重み付けを加えることにより制約実行順序を決めており、制約矛盾解消にノウハウデータを用いている。会話型で設計解を求め、結果は図面とドキュメントにより出力される。

[Araya & Mittal, 87], [Nagai et al., 89]は、過小制約設計問題（値を決定するパラメータの数に対して制約の数が少ないため、解が無数に存在し、決定的に解を求めることができない設計問題）を対象に、設計解探索のためのプランを自動生成する方法を論じている。プラン生成の内容は対象問題の構造、パラメータ依存関係、値の存在範囲などの情報に基づく、値生成器の配置、起動順序の決定などである。

2.4.3.3 コンフィギュレーション設計

2.4.2.2 で取り上げた、V-Belt [Dixon & Simmons,84], AIR-CYL[Brown, 85], PRIDE[Mittal et al.,86], ALL-RISE[Sriram, 87], VT[Marcus et al., 88]はいずれも、コンフィギュレーション設計技術を取り入れた設計システムである。コンフィギュレーション設計では、設計モデル及び設計解探索範囲を明確に規定できるので、探索アルゴリズムを自動化しやすいといえる。

2.4.4 設計言語

設計言語も多くの研究例がある。汎用性が高いことが特徴であるが、概念設計ないし基本設計が対象であり、詳細設計を対象としたものはない。

[Neville & Joskowicz, 93]は、機構の構成を構成要素間の相対的位置・姿勢の変化の範囲として表現し、挙動を位置・姿勢の遷移規則として表現する概念設計用言語を開発した。設計対象物の挙動が要求する挙動とマッチするか否かを推論できるとしている。[Lai & Wilson, 87] は、機能を自然言語（英語）に似たシンタックスで表現する機能設計用言語 FDL を開発した。設計対象の機能上の冗長、欠陥などを探しだし、改善方法をアドバイスすることができる。

[長澤,古川,荒牧, 84]は、述語論理に制約表現を導入し、さらにオブジェクト指向形式によるモデル表現をいち早く取り入れた言語 ADL を開発した。制約は制約伝播法を用いており、Ｖベルト設計などへの適用を試みている。[Yokoyama, 89]は、オブジェクト指向言語に制約を導入した知識表現言語として FREEDOM を開発した。ここでは、制約記述は数式と述語による。インスタンスにおける制約計算の結果、所属するクラスを変更する機能を持つが、制約伝播がオブジェクト間にループする場合は対応できない。フレーム言語に数式制約及び依存関係記述を導入した言語も開発されている[Harris, 86]。計算結果を記

録し、矛盾解決の手がかり、制約解決の説明などに使うことができる。

2.4.5 創発型設計

遺伝アルゴリズム(genetic algorithm)、属性文法などを用いて、生物の進化ないし生物の成長になぞらえた設計方法をとるものを創発型設計と呼ぶ。

[Brown & Hwang, 93] は、Ｖベルトドライブの構造固定型コンフィギュレーション設計に遺伝アルゴリズムを適用した。設計要求を満たすように、プーリー、ベルトをカタログから選択する。評価関数は重量としており、疑似最適解（最適解と数パーセントの差）は比較的容易に見つけることができるとしている。

構造可変なコンフィギュレーション設計への遺伝アルゴリズム適用例としては、[Bennett, 97]のアナログIC設計、[Skalak et al., 98]の配管設計などがある。前者では数１０個の素子から構成されるオペアンプ、フィルター、立方根演算、ロボットコントローラなどの回路を設計することができ、後者では太陽熱給水システム、工場における冷却水の配管設計を行うことができたとしている。いずれの場合も、ストリング列による対象構造の表現、無意味な構造の生成を防ぐオペレータの整備、設計解を評価するためのシミュレーションが重要なファクターとなっている。

[Rinderle, 91]は設計対象のパラメトリックモデルを属性文法により成長させ、目的の構造物を自動設計する方法を提案している。一例としてブーム（起重機の腕）設計に適用を試み、パラメトリック設計とコンフィギュレーション設計を融合する有力な方法として提案しているが、属性文法記述の意味づけ、対象表現能力が不明で、技術の汎用性に疑問がある。

２．５　設計研究における課題

設計研究における研究課題も、概念設計、基本設計、詳細設計に分けて議論する。

概念設計段階では、従来の設計案を模倣しない、新規性の高い概念設計案を導出することが課題である。機能設計に関しては、機能－挙動－構造表現、機能展開、構造生成に関して多数の方法が提案されてきているが、機能の表現方法、機能展開の方法、部分機能の取り方など自由度が大きく、議論が収束していない。定性物理モデルを利用した機能設計は有力な方法であるが、これも多様な機能の表現、分解、構造合成法が提案がなされている。いずれの方法も、部分機能モデル、機能－構造対応モデルのデータベースに依存しており、用意されたモデルの組み合わせの範囲で設計モデルが構築されるのであり、設計案の見落としがなくなるという利点はあるが、真に新しい設計案を導出できるのではない。データベースに蓄積するモデルの粒度をより小さくすれば、新規設計導出の可能性はより高くなるが、機能表現の粒度とのかねあいもあり難しい課題である。最近は、Triz などのように、設計アイデアデータベースを用意し、設計者のアイデア創出を支援するという方法が注目されている。ボンドグラフを使った機能設計研究もいくつか見られる。エネルギーの１入力１出力システムに対象を絞って、多様な設計解を導出した研究[Ulrich & Seering, 89]は興味深いが、１入力１出力以上に複雑なシステム設計への適用が今後の課題

である。コンフィギュレーション空間による機構定性シミュレーション[Joscowicz & Sacks, 91]は、対象の構造の数値的な値を必ずしも必要とせず、構造と挙動の因果関係を示すことが比較的容易という利点があり、厳密な解を必要としない概念設計段階では有力な支援ツールである。しかし、数値シミュレーションと同様、設計解そのものを求める能力はない。定性シミュレーションと結びつけた設計解探索の研究が期待される。

基本設計段階では、数値シミュレーション、パラメトリック／コンフィギュレーション設計が計算機支援技術の中心になる。数値シミュレーションについては、機構動力学、電子回路などの集中定数系については、精度の高いシミュレーションが可能になっているが、集中定数モデルにモデル化する過程は自由度が高く、ここにシミュレーションの精度が依存している。分布定数系のシミュレーションは、弾性域の応力解析、層流など線形近似が成り立つ場合は比較的精度の高い解析が可能であるが、塑性変形、乱流など非線形で動的変化の激しいものに対しては精度の高い解析は依然困難な課題である。解析モデル表現自体の模索が続いている。数値解析と設計解探索を関連づけて自動化する、いわゆる逆問題は、重要な課題であるが、手法が対象に依存しているものが多く、報告例も多くない。今後の課題である。またシミュレーションと CAD システムとの情報の受け渡しにも課題が残っている。

パラメトリック／コンフィギュレーション設計に関する課題は、後に詳述する。

詳細設計段階では図面 CAD、３次元 CAD が有効である。形状（図面）モデルのパラメトリック化、パラメータとスプレッドシートとの関連づけ、図面と３次元形状の連動、寸法指定の自動化、形状特徴を使ったモデリング、ベクトルデータとラスターデータの混在、レンダリング技術の発達によるリアリスティックな表現など、高度な設計支援が実現されつつある。しかし、形状情報と寸法精度、材料、加工法などの技術情報を一体的に扱う製品モデリングの研究や機能、加工と関連づけた形状特徴によるモデリングは依然研究途上である。切削、鍛造などにおける工程設計の研究は、ルールベース、事例ベースなどいろいろな手法が試みられているが、企業による個別性の問題、知識の曖昧性、形状モデルと知識ベースの関連づけなどの問題が十分に解決されたとは言い難く、依然研究途上といえる。

設計支援全般にわたる課題としては、設計システム間の情報の共有／交換の問題がある。概念設計、基本設計、詳細設計の各段階の設計の内容は大きく異なるので、全段階で有効な設計モデル、設計支援技術は存在しない。そこで、設計モデル間、設計システム間の情報連結の方法が重要な課題となる。内容が大きく異なる設計モデル間の情報連結の仕組みとしてメタモデル[Tomiyama et al., 89]が提案されたが、定量的情報を扱う設計モデル間の情報リンクを定性モデルで行うことに無理があると思われる。概念設計から詳細設計までの統合システムとして SchemeBuilder[Bracewell, et al., 95]が構築されたが、個別システムの連続性、適用汎用性を示せていない。設計システム間の情報交換技術として STEP[Kimura & Kojima, 95]が存在するが、形状情報伝達が中心であり、技術情報伝達に関して依然不十分である。

本研究の目的は、設計モデルの構造が未確定ないし、動的に変化しうる設計に対して適用できるパラメトリック／コンフィギュレーション設計技術を開発することにある。そこで、パラメトリック／コンフィギュレーション設計技術に属する従来研究を調査し、課題

```
(define-frame hoist_cable_model
 :own-slots ( (instance-of class)
  (subclass-of vt-component))
 :template-slots (
  (has-parameter-slot
       hoist_cable_diameter  hoist_cable_quantity_f
hoist_cable_ultimate_strength hoist_cable_unit_weight)
  (hoist_cable_diameter
    (slot-cardinality 1) (slot-value-type real-number))
  (hoist_cable_quantity_f
    (slot-cardinality 1) (slot-value-type integer))
  (hoist_cable_ultimate_strength
    (slot-cardinality 1) (slot-value-type integer))
  (hoist_cable_unit_weight
    (slot-cardinality 1) (slot-value-type real-number))
  (model-id  (slot-cardinality 1))) )

(define-frame hoist_cable_model_m01
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.5)
  (hoist_cable_quantity_f 3)
  (hoist_cable_ultimate_strength 14500)
  (hoist_cable_unit_weight 0.03)
  (model-id hoist_cable_model_m01)))

(define-frame hoist_cable_model_m02
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.625)
  (hoist_cable_quantity_f 3)
  (hoist_cable_ultimate_strength 23000)
  (hoist_cable_unit_weight 0.048)
  (model-id hoist_cable_model_m02)))

(define-frame hoist_cable_model_m03
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.5)
  (hoist_cable_quantity_f 4)
  (hoist_cable_ultimate_strength 14500)
  (hoist_cable_unit_weight 0.03)
  (model-id hoist_cable_model_m03)))

(define-frame hoist_cable_model_m04
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.625)
  (hoist_cable_quantity_f 4)
  (hoist_cable_ultimate_strength 23000)
  (hoist_cable_unit_weight 0.048)
  (model-id hoist_cable_model_m04)))

(define-frame hoist_cable_model_m05
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.5)
  (hoist_cable_quantity_f 5)
  (hoist_cable_ultimate_strength 14500)
  (hoist_cable_unit_weight 0.03)
  (model-id hoist_cable_model_m05)))

(define-frame hoist_cable_model_m06
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.625)
  (hoist_cable_quantity_f 5)
  (hoist_cable_ultimate_strength 23000)
  (hoist_cable_unit_weight 0.048)
  (model-id hoist_cable_model_m06)))

(define-frame hoist_cable_model_m07
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.5)
  (hoist_cable_quantity_f 6)
  (hoist_cable_ultimate_strength 14500)
  (hoist_cable_unit_weight 0.03)
  (model-id hoist_cable_model_m07)))

(define-frame hoist_cable_model_m08
 :own-slots (
  (instance-of class) (subclass-of hoist_cable_model))
 :template-slots (
    (has-parameter-slot  hoist_cable_diameter
hoist_cable_quantity_f  hoist_cable_ultimate_strength)
  (hoist_cable_diameter 0.625)
  (hoist_cable_quantity_f 6)
  (hoist_cable_ultimate_strength 23000)
  (hoist_cable_unit_weight 0.048)
  (model-id hoist_cable_model_m08)))
```

Fig. 2.5 Hoist cable model expressed by LISP language

```
class hoist_cable_model
super vt_component;
parameter
   hoist_cable_diameter real [],
   hoist_cable_quantity_f integer [],
   hoist_cable_ultimate_strength integer [],
   hoist_cable_unit_weight real [],
   model_id string [];
where
   data(hoist_cable_diameter, hoist_cable_quantity_f,
         hoist_cable_ultimate_strength,   hoist_cable_unit_weight, model_id);
local
   data(0.5,3,14500,0.03,hoist_cable_model_m01);
   data(0.625,3,23000,0.048,hoist_cable_model_m02);
   data(0.5,4,14500,0.03,hoist_cable_model_m03);
   data(0.625,4,23000,0.048,hoist_cable_model_m04);
   data(0.5,5,14500,0.03,hoist_cable_model_m05);
   data(0.625,5,23000,0.048,hoist_cable_model_m06);
   data(0.5,6,14500,0.03,hoist_cable_model_m07);
   data(0.625,6,23000,0.048,hoist_cable_model_m08);
end.
```

Fig. 2.6 Hoist cable model expressed by FDL language

を整理した。本研究では上記研究目的と同時に、これらの課題の解決を意識して進めた。

(1) パラメータ間の関係記述力が不十分

パラメータ間の関係表現を数式あるいは関数式のみに依存しているシステムが多いが [MacCallum & Duffy, 87], [Marcus et al., 88], [Higuchi & Nagasawa, 92]、それだけでは表現能力の点で不十分で、より多様な表現が求められることがしばしばある。

(2) 設計対象の記述性が低い

記述性とは、記述内容の理解のしやすさ、記述の効率（情報量／記述量）を意味することにする。多くの設計システムで設計対象の記述性に問題がある。モジュール構造を持たない記述形式をとるものは、全体の構成、内容が把握しにくい。モジュール構造を持つものでも、既存言語を使っているものは記述性が高いといえないものが多い。例えば、CLOS によるエレベータモデルの記述例（**6.3** 節参照）では、LISP 独特の表現（括弧が多い）による読みにくさと情報の重複による記述量の多さのため、記述性は高いとはいえない。**図 2.5** にエレベータモデルの中から hoist cable model（エレベータかごをつり下げるケーブルのモデル）を LISP で表現したものを例示する。情報の重複もあって、記述量が多く、読みやすいとはいえない。比較のために同じ内容を FDL で表現したものを**図 2.6** に示しておく。

(3) 設計解探索能力の問題

数式のみで、かつ、静的に（記述内容が条件によって変化することはない）設計対象のパラメータ間関係が表現されている場合は、グレブナーベース[Sawada, 95, 97]や数理計画法、数値解析などの数理的手法[Wolfram, 92]を適用して（最適）設計解を効率的に求めることができるとされている。しかし数式に加えてテキストの扱いが含まれていたり、条件により適用する制約が動的に変化しうる場合（例：**6.3** エレベータ設計）は数理的方法を適用することは困難で、このような複雑な設計問題を解けるシステムは多くない。

図 2.7 に、条件により適用する制約が変化する表現の例（エレベータ設計）を示す。

```
platform!model_id = `platform_model_m03 ==>
35 + platform_width * platform_weight_factor_x +3.228 * platform!platform_depth + 0.34
*opening_width_door + platform_weight_factor_ap * (0.226 * platform!platform_depth -
platform_weight_factor_z) = platform_weight;

platform!model_id = `platform_model_m02 ==>
35 + platform_width *platform_weight_factor_x + 2.774 * platform!platform_depth + 0.03
* platform_width * platform!platform_depth + 0.226 * opening_width_door +
platform_weight_factor_ap * 0.226 * (platform!platform_depth -platform_weight_factor_z)
= platform_weight;
```

Fig. 2.7 Constraint expressions with activation preconditions

(4) 設計対象のモジュール構造と組立

設計対象モデルをルール集合、数式集合、スプレッドシートなどを使って表現しているものはモジュール構造を持たないので[MacCallum & Duffy, 87], [Higuchi & Nagasawa, 92], [Ishiba et al., 95]、プログラム全体構造の理解、変更、再利用などが難しくなる。また、オブジェクト、フレームなどの利用により設計モデルがモジュール構造を持つものでも、モジュール間の関係が複雑かつ一様でない場合が多く、その場合モジュールの組立（関係付け）が煩わしい作業となる。

(5) 設計モデル構造の柔軟性

設計対象の構造が未確定であったり、設計途中で変更される可能性のある設計に対応することにより、設計支援の自由度、設計解探索解の範囲が格段に広がる。このためには、設計モデルを設計途中で動的に成長、変更、交換、削除することが可能である必要があるが、パラメトリック／コンフィギュレーション設計においてこのような能力をもつものは、非常に限られている。またそのようなものでも、変更の範囲をあらかじめ想定しており、構造変更の自由度や適用範囲に制限がある。

２．６　パラメトリック／コンフィギュレーション設計に有効な研究アプローチ

本研究、設計言語 FDL は、パラメトリック／コンフィギュレーション設計に属するも

のと見なしている。そこで、**2.5** の最後に整理したパラメトリック／コンフィギュレーション設計技術の課題を解決するために有効な研究アプローチとして、オブジェクト指向、制約解決、属性文法、遺伝アルゴリズム、ボンドグラフ、事例ベース推論、オブジェクトの動的構造変更能力をとりあげ、これらを概観、整理することにする。

(1) オブジェクト指向

オブジェクト指向の概念は抽象データ型、アクターモデルの考え方から生まれてきた。抽象データ型とは、リスト、ツリー、スタックなどのデータ構造の性質を抽象的に規定するという考え方である。これらのデータをプログラム言語で記述する方法は一通りではないが、データへのアクセス方法を統一すれば、データがいかに記述されているかは外部からは隠されることになり、プログラムの保守性、可読性、可搬性が向上することになる。アクターモデルとは、フォン・ノイマンモデルと対比される計算モデルである。フォン・ノイマンモデルでは１個の CPU がすべてのデータの制御を行うので、データ通信の観点から、CPU へのアクセス経路がボトルネックになるが、アクターモデルではアクターと呼ばれる「データ＋手続き」で定義されるモジュールが自分の役割を知って、独立に仕事を進め、他のアクターとはメッセージ交換により情報交換をするので、より効率的なデータフローを実現できると考えられる。

オブジェクト指向はこれらの特徴を継承し、クラス階層、has_a 階層、クラスーインスタンス関係によるプログラムモジュールの整理が可能であり、「対象の直感的なモデル化が容易」、「構造化により、複雑なプログラミングが容易」、「継承機能を利用することによりプログラムの再利用が容易」といった特徴を持つと考えられている。オブジェクト指向言語としては、Smalltalk[Goldgerg, 84]を始め、C++, Prolog++[Moss, 94], Prolog Object[Andersson et. al.,94], Common Lisp Object System [Steele Jr, 90], ESP [Uchida, 87]などがすでに開発されている。

オブジェクト指向概念を利用した設計システムの構築例はすでに無数に存在している [Brown & Chandrasekaranm 86], [Donaldson & MacCallum, 94], [Harris, 86], [Maher, 85], [Mittal et al, 86], [Nagasawa et al., 84], [Persidis & Duffy, 91], [Rigopoulos & Oppenheim, 90], [Shimada et al., 89], [Sriram, 87], [Tweed & Bijil, 90]。このように、多くの取り組みが試みられていることは、オブジェクト指向の特長が設計システム構築にも有効に働いていることの間接的な証明になっていると考えられる。

(2) 制約解決

制約解決とは制約表現の解を求めることであるが、制約表現としては陽関数、陰関数、不等式、非等式、一階述語、ルールなどが対象となり、対象により異なるアプローチで解が求められている。制約伝播法は関数表現に対して適応可能であるが、伝播がデッドロックしてしまう場合があるのが弱点である。これを回避する方法として、ブフバーガーアルゴリズムによりグレブナー基底を求め、方程式解を求める方法や、線形計画法を適用したものもある。また、制約系の矛盾の発見、矛盾を起こす方程式の指摘、不完全制約系からの代表解の導出などがグレブナー基底をベースに、実行可能である。一階述語論理に関しては、ホーン節に対象をしぼり、SLD(Linear resolution with Selection funcution for Definite

clause)導出を適用したProlog処理系が、健全性を保証しつつ解を効率よく求めることができる。最近は有限可算集合を対象に、探索手法により制約解を効率よく求める方法が中心に研究されている[Prade, 98]。このように、多様な制約解決の関連研究があるが、異なる表現の制約表現を混在させたり、条件により制約表現が変化するような対象の扱いは困難で、有効な数理的手法は提案されていない。

(3) オントロジー

オントロジーについては様々な捉え方があり必ずしも合意が得られている状態ではないが、個別に開発された領域依存知識ベースシステムの間で情報交換可能な理論体系を構築し、個別知識の有機的連携を可能にする理論とする立場をとる。「存在に関する体系的な理論」、「人工システムを構築する際のビルディングブロックとして用いられる基本概念／語彙の体系」などの定義も与えられている[Mizoguchi, 99]。

Stanford 大学のオントロジープロジェクト[http://www-ksl-svc.stanford.edu/]では、５０ほどの開発例があり、エレベータ設計(**6.3**)も取り上げられている。オントロジー言語としてLISPを使っており、数学、物理の基本語彙／理論をすべて体系的に表現している。ontolinguaと呼ぶシステム間翻訳システムにより、個別開発されたシステムが相互に情報交換できることになっている。

(4) マルチエージェントシステム

それぞれが知識を持ち、自律的に状況を判断して動作することのできる多数のエージェントが互いに情報交換し、協力して共通の問題を解決する仕組みを持つシステムである。黒板システムは、エージェント（自律的に動作できるルール）群が共通のデータベースとして黒板を使うという意味で、一種のマルチエージェントシステムといえる。故障診断、プランニング、スケジューリング、工程設計などで適用例が見られるが、設計への適用例は多く見られない。設計対象表現に力点を置かず、動的な設計知識によるシステム構築が必要なので、システム構築が難しいためと推定できる。しかし、設計対象モデルを取り込めば、コンカーレント設計の分野などで有力な方法と考えられる。

(5) 遺伝アルゴリズム

遺伝アルゴリズムは設計対象の性能・機能を評価するためのシミュレーションの自動化が可能であれば、適応可能である。そのような対象として、アナログ電子回路設計、空調の配管設計において、部品の配置、選択、パラメータの決定（コンフィギュレーション設計とパラメトリック設計の同時進行）を行い、近似的な最適設計解を求めている。特にアナログ電子回路設計への適用例[Bennett, 97]では、ベテランの設計者に迫る設計解を導出しており、印象的である。遺伝アルゴリズムによる設計は、対象が適切であればきわめて強力な設計手法であると評価できるが、シミュレーションの自動化が必須という点で、機械設計への適用は限られると思われる。また、遺伝情報へのコーディングの方法が個別の工夫に頼っており、適用例を広げるネックになっていると思われる。

(6) 事例ベース設計

事例ベースを利用した設計手法は、要求により設計対象の構造が大きく変化し、統一的設計手法の適用が困難な対象には有力な手法に思えるが、工程設計、スケジューリング、故障診断などへの適用が多く、設計への適用例は多くない。適切な設計事例の検索手段、設計変更の体系的手法、設計事例の網羅などが困難なためと推測される。既存設計（製品）、の再利用が可能など、メリットも大きいので、今後の研究の進展が期待される。

(7) オブジェクトの動的構造変更能力

オブジェクト指向言語では、プログラム動作時にクラスオブジェクトからインスタンスオブジェクトを生成したり、スロット値の変更を行うことは可能であるが、クラス／インスタンスオブジェクトの構造や、メソッド（アルゴリズム）記述を変更することは、Smalltalk, C++など、ほとんどのオブジェクト指向言語で不可能であった。オブジェクト指向言語を設計に適用した例は多いが、オブジェクトの動的構造変更能力の欠如のため、設計モデルの動的構造変更も同様に不可能となり、結果として設計の自由度に制限が加えられることになった。オブジェクトの動的構造変更能力の導入は、オブジェクト指向概念を用いた設計システムの設計自由度の向上に大きな効果があると期待できる。

図 2.8 に **2.5** に掲げたパラメトリック／コンフィギュレーション設計の課題と **2.6** で掲げた研究アプローチの対応関係を示す。

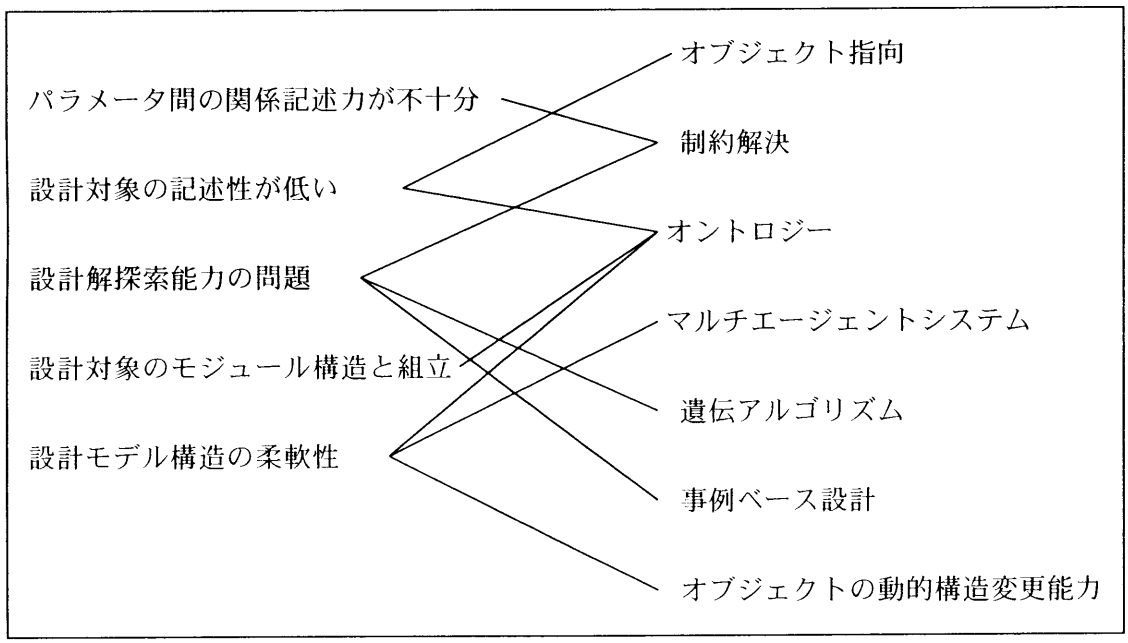


Fig. 2.8 Relations of parametric/configuration design issues and research approaches to resolove them

２．７　ＦＤＬ設計言語の位置づけ

2.5 に掲げた、パラメトリック／コンフィギュレーション設計の課題を解決することを、本研究の主要目的として設定する。**2.6** では、そのためにとりうる研究アプローチとして、(1) オブジェクト指向、(2)制約解決、(3) オントロジー、(4) マルチエージェントシステム、(5) 遺伝アルゴリズム、(6) 事例ベース設計、(7) オブジェクトの動的構造変更能力、の７項目を掲げ、整理した。このように、パラメトリック／コンフィギュレーション設計の諸課題を解決するためには多用なアプローチが考えられるが、本研究では、オブジェクト指向、制約解決、オブジェクトの動的構造変更のアプローチを採用すること
とした。

2.7.1　本研究のとった研究アプローチの理由

本研究は「新ソフトウエア構造化モデル」プロジェクトの中で実行された。巨大なソフトウエアシステムの開発とメンテナンスに関わる困難性を軽減することを目指した、次世代の言語開発がプロジェクトの目標であった。より具体的には、ソフトウエアシステムの構成、データ構造、アルゴリズムの変更を、整合性を保ちつつ容易に行うことが可能となるメカニズムを導入した計算機言語を開発することを目標とした。この目標に対して、本研究の目的とオーバーラップさせながら研究を進める必要があったので、パラメトリック／コンフィギュレーション設計能力を持つ制約・対象指向言語を開発し、そこにオブジェクトの記述内容を整合性を維持しつつ変更することが可能となる、オブジェクトの動的構造変更能力を導入する研究アプローチをとることにした。結果として、パラメトリック／コンフィギュレーション設計に関わる諸課題（**2.5**）を解決し、動的構造変更の可能な制約対象指向言語を提示することができたので、このアプローチは妥当であったと考える。

遺伝アルゴリズムは有力なアプローチではあるが、対象表現と設計評価の方法がアプリケーションに強く依存しており、汎用性の高い設計言語の開発をめざす本研究のアプローチと性格が異なるので採用しなかった。事例ベースは本アプローチと相容れることはできるが、設計事例の収集が困難で、そのような方向性は打ち出さなかった。マルチエージェントシステムは興味ある方向性で、組立分解プランニングの研究に適用を試みているが、FDL への適用は将来の課題である。

2.7.2　FDL の特徴と従来研究との比較

FDL は機械システムの基本設計支援汎用ツールを目指して開発された。旋盤の基本設計演習における経験[Inoue et al., 88]から、設計の自動化は目指さず、設計者の思考の自由度がより高い、会話型設計環境の提供を目指した。しかし、エレベータ設計（**6.3**）のように設計対象の探索空間が巨大な場合には、会話型設計による対応では無理であり、自動設計も対応可能なようにした。

FDL のもつ特徴について整理し，従来研究と比較すると次のようになる。

(1) 設計用形式言語

文法が明確に定義された形式言語により設計対象を表現する。オブジェクト指向、制約表現、関係データベース、などの情報技術を取り入れ融合したものであり、それぞれの特徴の相乗効果により、高度な設計支援能力の獲得を目指した。

言語表現形式は、設計者の立場から、設計対象の表現に適切な形式をとり、文法も明確に示している（付録 1,2)。従来も設計用言語の開発はいくつかあるが、モジュール構造をもたず形式性が弱かったり、インプリメンテーションに使った言語形式の影響を強く受けて、表現が理解しにくいなどの問題があった。FDL ではそのような問題を解決した。

(2) 制約指向言語

制約伝播法を中心に多様な制約表現、制約表現の条件付き起動などの仕組みを取り入れたことにより、大規模、非線形、動的に変化する制約表現も解決可能になった。制約問題を解く方法としては、関数形式によるもの、ブフバーガーアルゴリズム、線形計画法などの数理的手法によるものなどがあるが、条件により動的に変化する制約記述を扱えるものはほとんどなかった。長澤らのシステム[Nagasawa et al., 84]はそのようなものも扱えるが、エレベータ問題（**6.3**）でベンチマークテストを行ったところ、最初の解を求めるのに 8 時間程度かかったとされる。FDL では最初の解を求めるのに 5 分程度、すべての解(458 通り)を求めるのに 10 時間程度でできており、使った計算機環境は異なるが、FDL の解探索の効率の高さを示している。

(3) モデル間の関係付け

モジュール構造を持つ表現形式（言語、データ構造など）により対象モデルを表現する場合、対象を構成する要素モデルをアセンブリすることにより行う場合が多い。この場合、要素モデル間の関係付けは、データ要素（パラメータ）ごとに指定する場合が一般的であるが、非定型で煩わしい作業であった。本研究では構造融合操作（**5.2**）を考案することにより、アセンブリ作業を単純化、定型化し、従来より容易に行うことが可能となった。

(4) 動的構造変更機能

設計モデルを表現するオブジェクトの動的構造変更（プログラム実行系の上で、プログラムのデータ構造やアルゴリズムを変更すること）を可能とするために、構造融合、構造交換などのメタ操作、was_a、was リンクによる動的オブジェクトの継承管理方式、状況を判断し構造変更を自動的に行うための配置演算子などの新技術を導入した。従来のオブジェクト指向言語でも、CLOS など、動的構造変更を許しているものもあるが、クラスオブジェクトだけに対象を限定されたり、変更の内容も強い制限のある場合が一般的であったが、FDL ではクラス／インスタンスを問わず、すべてのオブジェクトを対象に、大きな制限を受けず、動的構造変更が可能である。また、構造変更に伴う、種々の情報矛盾も解決ないし管理する仕組みを備えている。

オブジェクトの動的構造変更機能を導入した結果、不完全な設計モデルで設計を開始し、必要に応じて設計モデルを動的に成長、変化させつつも、設計の連続性を保ちながら設計

を進めることが可能になった。従来、概念設計用の単純なモデルを使ってモデルの変更、成長を扱った研究はあるが、本研究のように基本設計段階のパラメトリック／コンフィギュレーション設計用のモデルを動的に成長、変化させているものは、属性文法、遺伝アルゴリズムを適用した研究が散見される程度である。

(5)コンフィギュレーションシステム

従来の研究の多くが個別例題の特徴に依存した探索アルゴリズム、知識表現をとっており[Mittal et al., 86], [Sriram, 87], [Marcus et al., 88]、汎用性に問題があった。既存の制約ベースの汎用システムは解探索能力が十分でなかった。遺伝アルゴリズムは対象が適切であれば、優れた能力を発揮するが、適用に様々な条件があり、機械設計への適用は進んでいない。これらに比較し、FDL は高い汎用性と高い解探索能力の両者を確保したと考える。

(6) 会話型設計と自動設計

設計システムの開発研究を手がけている研究グループの分類の仕方の一つとして、設計支援派と自動設計派があるように思われる。設計支援は、設計は意志決定の連続で行われるものであり、設計の主体はあくまで設計者であるべきであるという考え方をとっている。一方、自動設計派は人工知能研究者に多いように思われるが、設計も人間の知的活動の一つであり、知能の解明がいつかは可能であるという仮定に立つならば、設計活動も解明可能であり、人工設計（自動設計）をめざすのが研究の本筋だという考え方である。また、LSI 設計など大規模設計では、実際に自動化の要請が高いものと思われる。

このような考え方を背景に、従来は会話型設計による設計支援か、自動設計のいずれかの立場をとって、システムの研究開発が行われてきたように思われる。著者は、工作機械設計などの経験から設計支援派の考え方をとってきた。しかし、エレベータ設計（**6.3**）では、対象の規模、複雑さの故に会話型設計では対処が困難で、自動設計の必要性も認識した。以上の設計事例による評価の結果、FDL では、会話型設計、自動設計のいずれも、選択できるようにした。

図 2.9 にパラメトリック／コンフィギュレーション設計の課題と FDL がとった解決技術の対応関係を示しておく。

２．８　２章のまとめ

本章では、**2.1** で設計用語の定義を与え、**2.2** で特にパラメトリック／コンフィギュレーション設計について数理的定義を与えた。**2.3** で設計活動の内容を製品企画、概念設計、基本設計、詳細設計の４段階に分類して、その内容を整理した。これらは、本論文を構築していく上での基礎となるものである。**2.4** では設計関連研究を、設計段階、設計技術のらの課題のうち、パラメトリック／コンフィギュレーション設計に関わる課題について、その解決のために可能なアプローチを列挙した。**2.7** では本研究の位置づけを行った。FDL 観点から整理し、概観した。**2.5** では、設計研究における課題を整理した。**2.6** では、これ

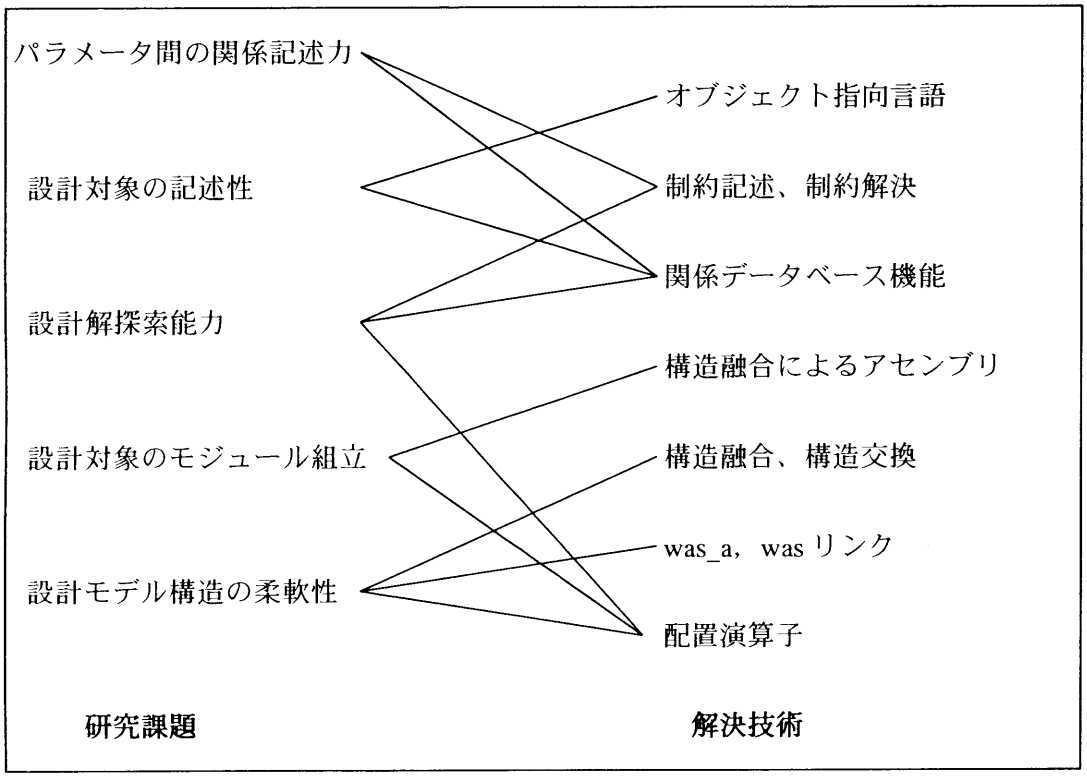


Fig.2.9 Relations of parametric/configuration design issuues and research approaches of FDL

は **2.6** で示した研究アプローチのうち、オブジェクト指向、制約指向、オブジェクトの動的構造変更機能の導入というアプローチを選択したが、**2.7.1** でその理由付けを行った。**2.7.2** では、FDL のもつ特徴ごとに、従来研究との関係、違いについて整理した。

次章では、FDL の概要を FDL-I と FDL-II の 2 期にわけて紹介する。

第３章　基本設計支援用制約対象指向言語の概要

本章では、基本設計支援用制約対象指向言語 FDL の概要を紹介する。

FDL の開発は FDL-I（1990 年～ 1992 年）、FDL-II（1993 年～ 1995 年）の２段階からなる。FDL-I は論理型対象指向言語 ESP[Uchida, 87]により開発されたが、既存の対象指向言語のメカニズムを利用しているため、オブジェクトの構造を動的に変更することができない点が問題となった。そこで、新たに FDL-II を Prolog [Clocksin & Mellish, 87]により開発し、オブジェクトの動的構造変更能力を取り入れ、基本設計支援の柔軟性を獲得した。

以下、FDL-I、FDL-II の概要を紹介する。

３．１　FDL-I

FDL-I はパラメトリック設計支援を目的として開発した制約対象指向言語である。コンパイラー型言語であり、システム記述及びオブジェクト記述に ESP を用いている。FDL-I の特徴を整理すると次のようになる。

（１）制約対象指向言語

オブジェクトのプログラム（メソッド）部分を制約により記述する対象指向言語である。部品スロット（part slot）とパラメータスロット（parameter slot）の２種類のスロットを持ち、前者は構造を持つオブジェクト（設計モデル）を、後者は integer, real, number, string の４種類のデータを格納するために用いる。制約記述も含め、読みやすく、書きやすい記述方法をとっており、設計技術者自身がプログラミングし、設計業務に使用することを目指している。

（２）３種類の制約

FDL は単一属性制約、同一性制約、多属性制約の３種類の制約を扱う。単一属性制約は、パラメータスロットのデータタイプ、存在範囲、単位、デフォルト値を指定することができる。同一性制約は二つのスロットに存在するオブジェクトが同一であることを宣言するためのものであり、一つのオブジェクトを共有（structure sharing）することにより表現される。オブジェクトごとに分散的に定義された多属性制約を接続し、大域的ネットワークを構成する役割も持つ。

多属性制約はデータの整合性の管理、設計知識、設計仕様の表現に用いられ、述語、等式、不等式を使って記述できる。制約に起動条件を付加し、制約の実行制御を行うこともできる。制約解法として制約伝播法[Sussman & Steel, 80]を用いており、平方根、三角関数、対数、値発生器などを利用できる。

（３）関係データベース

あらかじめモデルに記述した関係データの検索機能を提供する。Ａ型とＢ型の２種類がある。Ａ型は多属性制約と同型式で記述された検索式を用い、制約ネットワークに完全に組み込まれ、検索結果は制約ネットワークに返される。Ｂ型では、検索式は関係属性（データ項目）とモデルのパラメータの関係を表現した制約式として定義され、自動的ないし会話的に検索が行われる。検索結果は専用ウインドウに表示され、選択されたデータはすべて設計解候補とみなされる。検索結果は制約ネットワークに返されない。

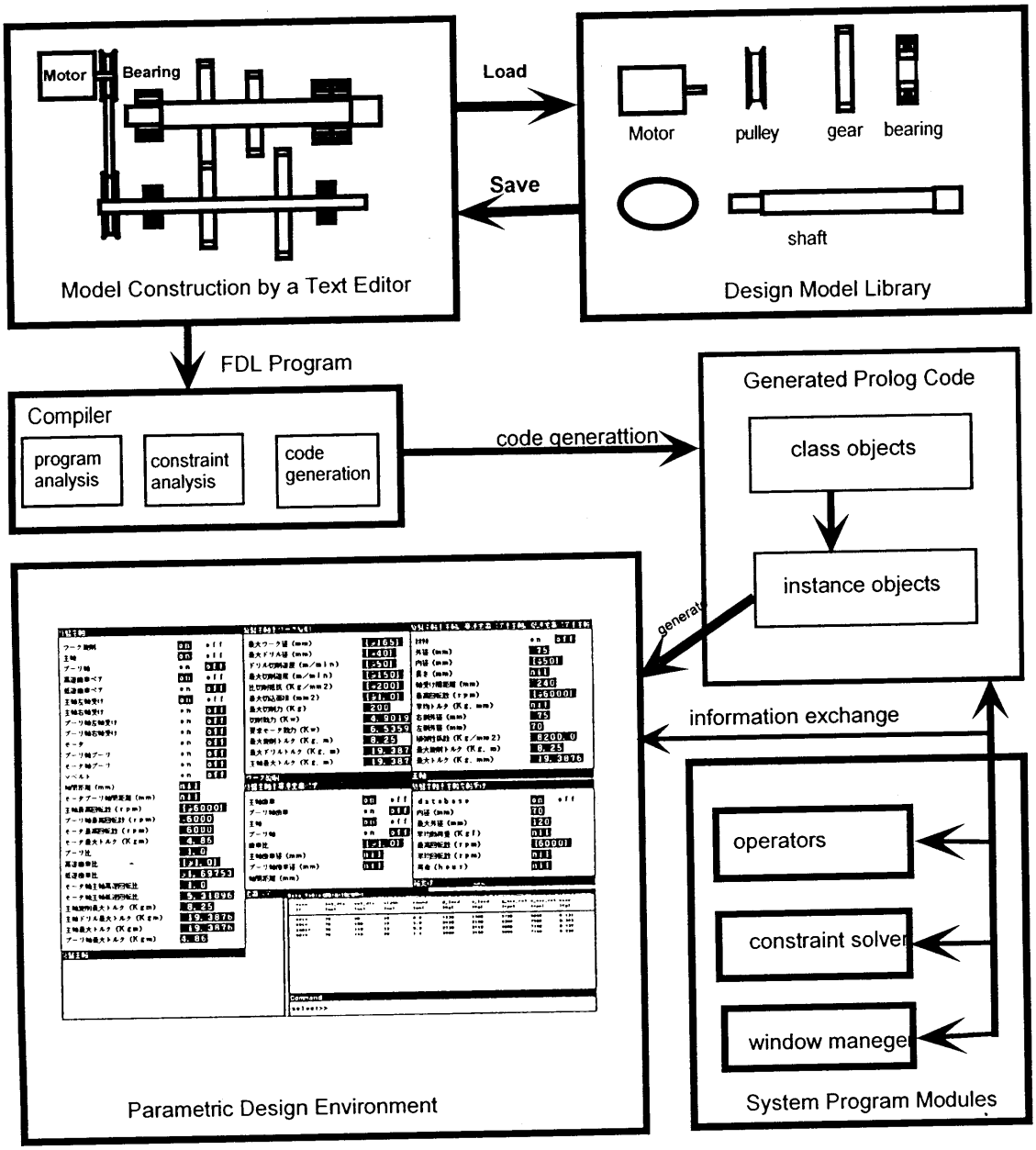


Fig. 3.1 Architecture of the FDL-I based design system

（4）構造化スプレッドシート型パラメトリック設計支援環境

オブジェクトごとに、その部品スロットとパラメータスロットの構成を反映したウインドウが、インタフェイスとして生成され、任意の場所に表示することができる。ウインドウのスロット間に定義された制約は、データ入力によって起動され、値の自動決定／修正が行われる。ウインドウ・スロットには入力専用、出力専用、入出力用の３タイプがあり、これらは制約定義により自動的に決まる。また、[] でくくって入力した値は定数とみなされ、制約起動によっても変更されない。

FDL-I による設計システム・アーキテクチャーを**図 3.1** に示す。部品モデルは FDL-I により記述するが、設計ライブラリに用意されたモデルを利用してもよい。部品モデルはモデルエディター(model editor)により組み上げられ、設計対象全体を表すアセンブリモデルとなる。FDL-I ではモデルエディターとしての特別のツールは用意されず、一般のテキストエディターにより部品モデルを組み合わせたアセンブリモデルを定義する。アセンブリモデルをコンパイルするとパラメトリック設計支援環境(parametric design environment)が生成される。右下にあるのが B 型関係データベース用ウインドウ、その外は構造化スプレッドシート型のパラメトリック設計用ウインドウである。モデリング管理部(modeling manager)は、モデル記述のシンタックスチェック、単一属性制約及び同一性制約の無矛盾チェック、多属性制約における因果関係の無矛盾チェックを行う。制約解決器(constraint solver)は制約伝播法によるものを使っているが、FDL と独立のモジュールになっているので、他の方式による制約解決器を利用することも可能である。

FDL-I では以下のような４段階の手順を踏んで設計を進める。

１）対象モデリング

対象モデルを定義する段階である。(1)サブモデルの準備、(2)サブモデルのアセンブリ、(3)整合性管理用制約の記述、(4)設計知識の記述の４段階からなる。

サブモデルの準備は、(1)ライブラリにあるものをそのまま利用する、(2)継承機能を使って記述を付加する、(3)テンプレートとして利用し一部手直しする、(4)新たに書く、のいずれかの方法によって行う。サブモデルのアセンブリの段階では、同一性制約と多属性制約によりモデル間の関係を記述する。モデル単位で記述された多属性制約はアセンブリにより互いに結合され、大域的ネットワークを構成する。

２）モデル・コンパイル

FDL をコンパイルし、ESP に変換する段階である。シンタックス・チェックと同一性制約におけるデータタイプと単位の一致をチェックする。

３）仕様入力

プログラムを実行し、設計用のウインドウが作成された段階で、要求仕様を値としてウインドウの適切なスロットに入力する。これらの値は、[] で囲って入力することにより制約解法の中で定数として扱われる。

４）会話型設計

マルチウインドウを介して、会話型の設計を進める。パラメータ入力は@付スロットを優先的に行い、^付スロットは入力禁止である。入力された値は、まず単一属性制約によ

るチェックを受け、次に多属性制約に引き渡される。多属性制約はサブモデルをまたがって、複数のモデル間の関係を制御し、同時に多数の値を変更する。

FDL-I の評価のため、旋盤設計を例題に実験を行った。旋盤は**図 3.2** の右下に示すように、モータ、ベルトドライブ、中間軸、主軸、歯車ペアなどから構成されている。旋盤主軸モデルのデータは階層構造を持っており（**図 6.2** 参照）、部品 29 個、パラメータ 256 個、同一性制約 101 個、多属性制約 26 個、A型データベース 16 個（主軸材料、中間軸材料、モータ、vベルト、プーリ、キーの選択に使用）、B型データベース 4 個（軸受の選択に使用）から構成されている。要求仕様として与える値の数 11 個、ユーザが決める値の数 2 個で、残りのパラメータ値はすべて制約解決により自動的に決まる。

各パラメータの初期値は nil である。まず、最大ワーク径、最大ドリル径、最大ドリル切削速度、最大切削速度、比切削抵抗、最大切込面積、主軸材料、主軸最大回転数、主軸平均回転数、主軸内径、軸受要求寿命、の値を要求仕様として[]付きで入力する。これらは@付スロットであり、システム側が入力を求めていることが明示されている。次に、やはり入力が求められているプーリ比を 1.0 と仮に決めて入力する。すると歯車歯数の存在可能範囲が定まるので値生成器が自動的に起動され、生成検証法により連立不等式を満たす値を求めるが、すべて失敗する。そこでプーリ比を 2.0 に変更し、再び生成器を起動すると今度は連立不等式を満たす値が見つかる。さらに、歯車の数をユーザ自身が入力して別解を見つけることもできる。**図 3.3** 左下には、こうして得られた設計結果例の一部が示されている。

FDL-I はコンパイラー言語であり、モデルの動的構造変更（言語実行中の構造変更）を行うことは不可能であった。従って、あらかじめ設計対象の構造が確定し、パラメータの変更のみで処理できる、いわゆるパラメトリック設計に対してのみ適応可能であった。モデルの構造変更を行うには、アセンブリモデルをテキストエディターで修正し再コンパイルする必要があり、設計の連続性が保てず問題であった。そこで、オブジェクトの動的構造変更を可能とする FDL-II を開発することにした。

3．2　FDL-II

オブジェクトの動的構造変更はほとんどの対象指向言語で不可能である。一部の対象指向言語は動的構造変更機能を持っているが、それらもインスタンス単位の動的変更ができないため、動的な設計オブジェクトを表現するためには不充分であった。そこで、Prolog 上に一般的な対象指向言語を構築し、その上に制約解決、関係データ検索機能、及び、FDL-II の特徴である動的構造変更機能をインプリメントした。なお、FDL-II はインプリメンテーションに用いた Sicstus Prolog [Andersson et al., 94]の特徴を受け継いで、コンパイラー処理とインタープリター処理の混在する言語となっている。

FDL-II には、動的構造変更の能力を実現するために次のような機能を取り入れた。

（1）スロット、制約などの追加及び削除を行うメタ操作

オブジェクトのパラメータ／部品スロット、単一属性制約、他属性制約、ローカル述語の追加及び削除の操作を行うことができる。この際、データの整合性の維持が考慮される。

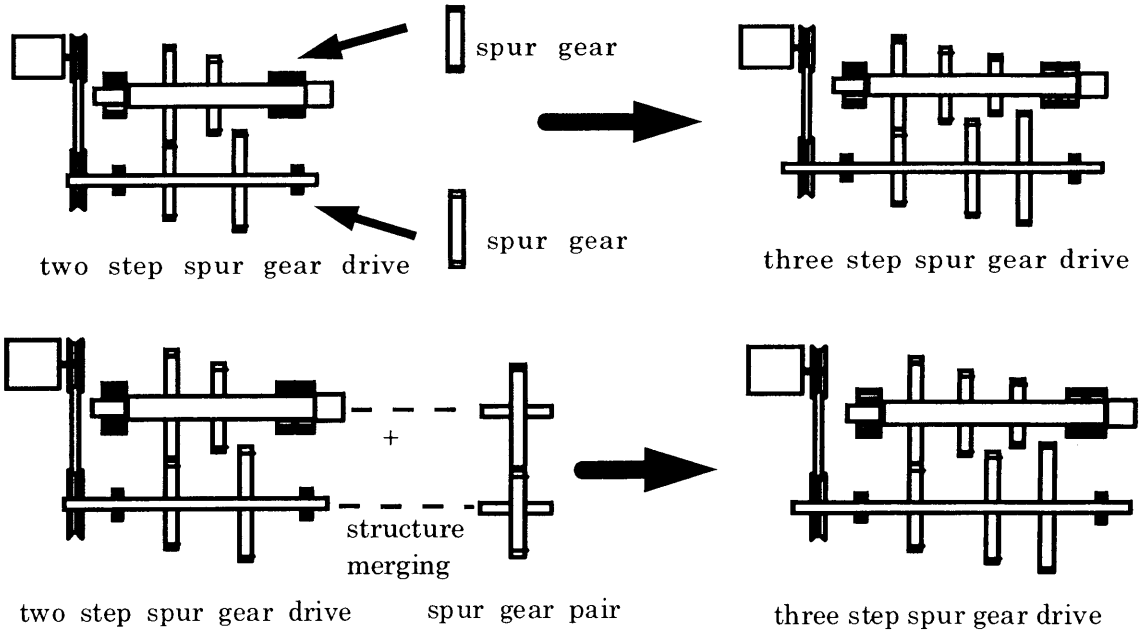


Fig. 3.2 Assembling two gears in two ways

（2）構造融合

二つのインスタンスオブジェクトを融合する操作である。対象の二つのオブジェクトを消去し、その代わりに二つのオブジェクトのデータ構造並びに制約記述の和集合に相当するオブジェクトを生成する。また、二つのオブジェクトが持っていた他のオブジェクトとの関係（has_a 関係）も、新たに生成されたオブジェクトが引き継ぐ。

構造融合操作は、煩雑かつ不定形な設計モデルの組立作業を単純化、定型化する上で大変有効である。例えば**図 3.2**（上）に示すように、２段変速旋盤の設計モデルを３段変速旋盤の設計モデルに変更するためには、二つの平歯車を２段変速旋盤に組み付け（組み立て）る必要がある。この組み付け作業とは、従来手法では２つの平歯車と２段変速旋盤の２本のシャフトとの間に次のような関係式を記述することに相当する。

・駆動軸、駆動歯車間に最大回転数、回転動力、最大トルク、直径の等値関係を設定する。
・従動軸、従動歯車間に上記と同様の等値関係を設定する。
・駆動歯車、従動歯車それぞれのモジュール、歯幅、周速間に等値関係を設定する。
・駆動歯車と従動歯車の直径の和、及び、軸間距離の２倍の値を等しくする。
・歯車減速比、駆動側、従動側の最大回転数及び最大トルクの間の関係を記述する。

このような組み付け作業の内容は対象により変化するので、組み付け作業は複雑かつ不定形な作業といえる。一方、構造融合を用いた組み付け作業は、**図 3.2**（下）に示すように、歯車ペアユニットと２段変速旋盤の駆動軸、従動軸、軸間距離同士を構造融合させるのみで完了する。なぜならば、歯車ペアユニット内には上記の関係式がモデル内の関係として予め記述してあり、歯車ペアユニットと２段変速旋盤の間で上記の三つの構造融合操作を行うと、関係式が自動的に歯車ペアユニットの歯車と２段変速旋盤の従動軸、駆動軸、軸

間距離の関係に変化するからである。構造融合操作はプログラム作成時、プログラム実行時、いずれの場合も適用可能である。

（３）構造交換

オブジェクト X をオブジェクト Y と交換する操作。この場合、オブジェクト X と他のオブジェクトとの関係(has_a 関係)はオブジェクト Y が引き継ぐ。シャフト、ベアリングなどの部品モデルを異なるタイプのモデルと交換する場合に使用する。例えば、主軸モデルと中間軸モデルの交換、アンギュラー玉軸受けモデルと転がり軸受けモデルの交換などに用いられる。

（４）配置演算子

設計モデルの組立、構成変更などの作業を自動的に行うためのルールである。

＜起動条件＞　==〉　＜構造変更操作列＞；

の形式により定義され、起動条件が満たされると、構造変更操作列部に定義されているスロット／制約の追加、構造融合、構造交換などの操作列を実行して、オブジェクトを目的の構造に自動的に変更する。あらかじめ可能性のあるモデルの構造変更操作を配置演算子として準備しておくことにより、想定内のモデル構造変更であれば、自動的に変更作業を行うことができる。

（５）動的オブジェクトの管理機構

オブジェクトを動的に変更すると、クラスーサブクラス間ないしクラスーインスタンス間の継承関係が壊されてしまうという問題が生じる。クラスオブジェクトに関しては、CLOS、KL-I などの言語が動的構造変更を許しているが、継承関係の整合性を保つために、変更対象のクラスを継承するすべてのオブジェクトを同時に変更する仕組みをとっており、オブジェクトを個別的に変更することができない。インスタンスオブジェクトに関しては、動的変更を許している言語は存在しない。このような事情で、既存言語では動的に構造が変化していく設計モデルを表現するには不充分であった。この問題を回避するために、FDL-II には was_a リンク及び was リンクを導入し、オブジェクトの動的変更時の継承関係の管理を行った。was_a リンクは構造変更の操作を受けたインスタンスの継承関係を管理するためのものである。スロットの追加などのメタ操作を受けたインスタンスは親クラスとの関係を is_a リンクから was_a リンクに変更し、was_a リンクには元の親クラスと動的構造変更の履歴を記録する。変更されたインスタンスは was_a リンクを介して元の親クラスを参照するが、履歴を参照することにより、継承できる情報と、継承対象から除外する情報を区別する。was リンクは親クラスと動的変更を受けた子クラスとの関係を管理するためのもので、変更履歴の記録や継承情報の選択の仕組みは同様である。

なお、本件に関しては、「オブジェクト指向言語における動的オブジェクトの管理方式」という名称で、日米の特許を取得済みである。

国内特許　特許番号２５８０５３６　登録日　平成８年１１月２１日

米国特許　特許番号５５６００１４　登録日　平成８年　９月２４日

以上のような機能を導入することにより、FDL-II はモデルの動的構造変更が可能となった。**図 3.3** に FDL-II によるシステムアーキテクチャを示す。メタオペレーションが追加され、モデルエディタにはグラフィックエディタが用意され、モデルの動的構造変更も

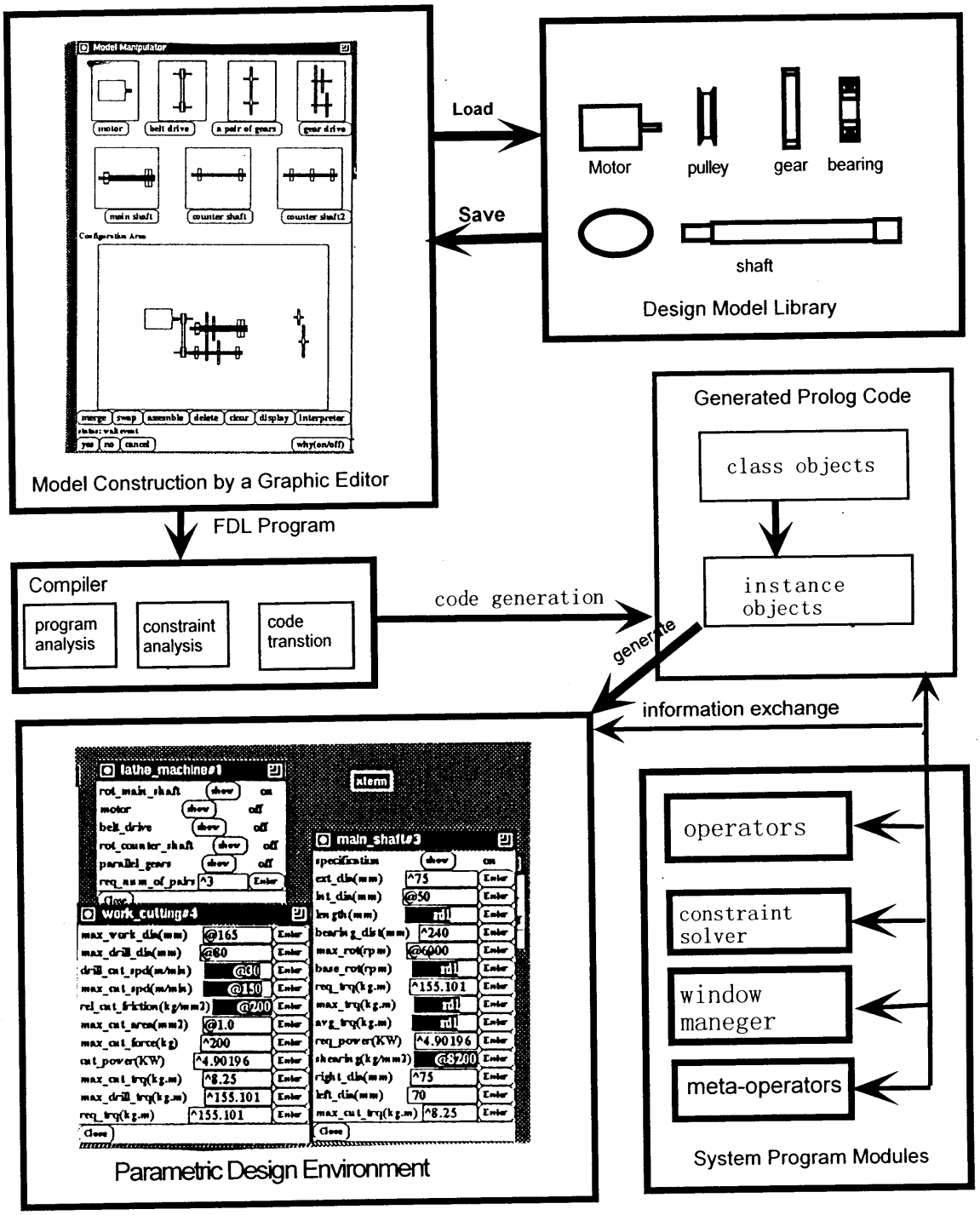


Fig. 3.3 Architecture of the FDL-II based design system

グラフィカルに行うことができる。

FDL-II でも、言語評価のために旋盤設計を試みた。FDL-I で使った例題では、旋盤の構造を最初から２段変速として固定的に扱ったが、実際は、主軸の出力が最大トルク出力時以外一定であることが求められため、要求仕様に応じて構成（歯車変速段数）が変化する。しかも、設計がある程度進展しないと構成は決定できない。そこで、構成可変な旋盤モデルによる設計を FDL-II の例題として取り上げる。

ここでは、旋盤設計は主軸、ベルトドライブ、モータのみから構成される不完全なモデルからスタートする。要求仕様を順次入力し、必要な変速段数が決まった段階で、必要に応じて歯車ペア等を追加してモデルを構成する。変速機構の構成変更は、構造融合操作等を使ってマニュアルで行う方法と、配置演算子により自動的に行う方法がある。**図 3.4** に旋盤モデルの FDL による記述例を示す。where 以下に制約、配置演算子が記述されている。(1)は、モータ軸とベルトドライブの駆動軸の構造融合記述、(2)はモータの要求パワーを、(3)は必要となる変速段数を計算する制約記述である。(4)～(6)は変速段数１～３段の構成に自動変更するための配置演算子である。なお、@、^は、それぞれ入力、出力パラメータを表すための前置詞である。

図 3.5 に FDL が提供する設計環境を示す。左下のウィンドウ"Model Manipulator"は、モデル構築用のライブラリ、モデルの２次元表示領域、種々の指示を行うためのボタン群から構成されている。配置演算子(6)が起動されて、３段変速機構を持つ旋盤モデルが自動的に構築されている様子を見ることができる。なお、"why(on/off)"ボタンは、個々のパラ

```
class lathe_machine
part
 rot_main_shaft:= #rot_main_shaft,
 motor:= #motor,  belt_drive:= #belt_drive,
 main_shaft <== rot_main_shaft!main_shaft;
parameter
 req_num_of_pairs integer [];
where
motor!flw_shaft==belt_drive!drv_shaft;                                  %(1)
@main_shaft!req_power/0.75 = ^motor!req_power;                          %(2)
log((@main_shaft!max_rot) * (@main_shaft!req_trq)/
(@motor!flw_shaft!max_trq* @motor!flw_shaft!max_rot))=X,
 log((@motor!flw_shaft!max_rot/ @motor!flw_shaft!base_rot))= Y,
 floor((X/Y+0.1)) + 1 = ^req_num_of_pairs;                              %(3)
@req_num_of_pairs= 1     ==> :merge(main_shaft,belt_drive!flw_shaft);   %(4)
@req_num_of_pairs >= 2, @req_num_of_pairs =< 3  ==>
 :add_part(self,rot_counter_shaft,#rot_counter_shaft),
 :add_part(self,parallel_gears,#parallel_gears),
 :merge(belt_drive!flw_shaft,rot_counter_shaft!counter_shaft),
 :merge(rot_counter_shaft!counter_shaft, parallel_gears!drv_shaft),
 :merge(main_shaft,parallel_gears!flw_shaft);                           %(5)
@req_num_of_pairs=3  ==> :assemble(parallel_gears, gear_pair2,#gear_pair); %(6)
end.
```

Fig.3.4 A lathe description with configuration operators

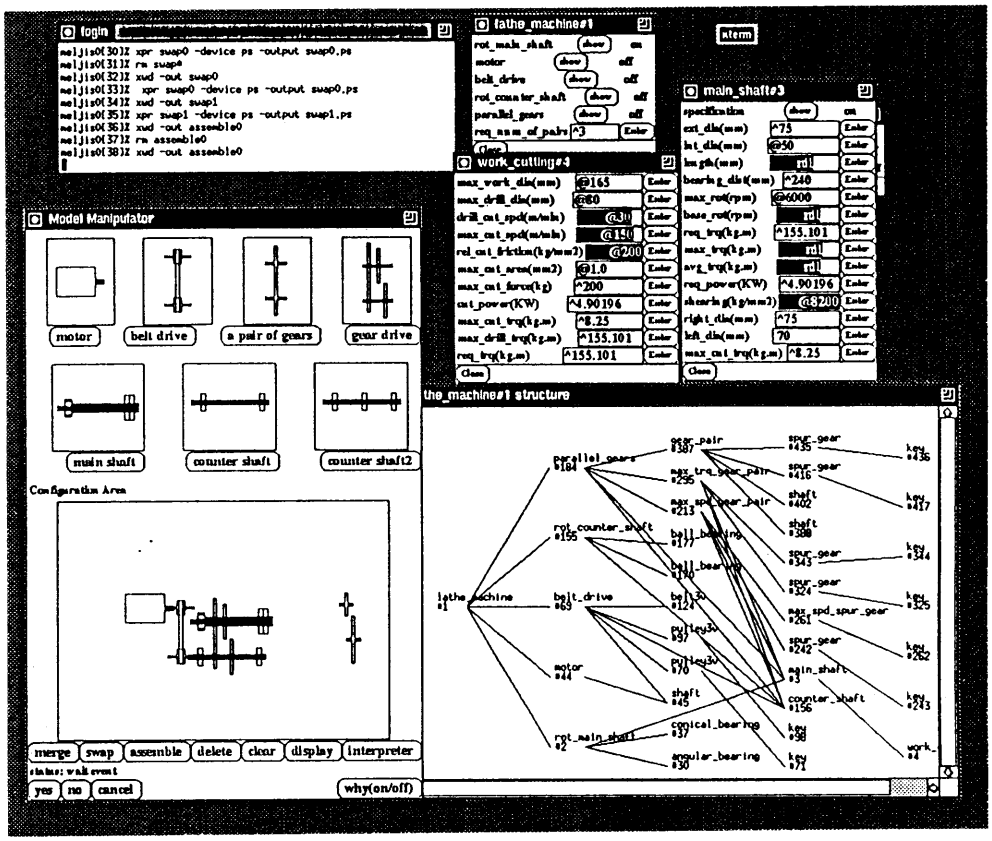


Fig. 3.5　A design environment created by FDL-II

メータの値を求めるのに使われた制約式を表示するためのものである。右下のウインドウはモデルのデータ構成を示すためのもので、モデルが構造変更する度に再描画される。右上のウインドウは、制約起動、モデル生成などのイベントがある度に、その詳細情報を表示する。

FDL-II では、次のようなことが可能となった。

（1）設計モデルの構造が未確定でも設計を始めることができる

概念設計段階で、モデルの構造を確定できるとは限らない。そのような場合でも、仮に決めた設計モデルでパラメトリック設計をスタートさせ、設計の進展に従ってモデルの構造を変更することができる。

（2）パラメトリック設計と組合せ設計の同時進行

パラメトリック設計と組合せ設計を任意の順序、任意の回数行うことができる。モデルエディターで組合せ設計（モデルのアセンブリ）を行った後、構造化スプレッシートによりパラメトリック設計を行うが、そのモデルの変更が必要であることが判明した場合、モデルエディタでモデルの変更を行うことができる。この過程を任意の回数繰り返すことができるので、より広い範囲の設計解探索を行える。

（3）設計モデルのライブラリ蓄積

動的構造変更により新たに作られた設計モデル（クラス、インスタンスオブジェクト）をライブラリに蓄積する事ができる。従って、設計作業量が増えるにつれ設計モデルライ

ブラリの蓄積を増やすことができる。

３．３　３章のまとめ

３章では、FDL の開発を FDL-I と FDL-II の２期に分け、それぞれの技術的概要について設計例題（旋盤設計）を交えて紹介した。FDL-I は制約対象指向言語であり、数式、不等式、述語、ルールなど多様な制約を使うことができ、高度な設計解探索能力を持つことが特徴である。FDL-II は FDL-I の技術的特徴を継承しつつ、動的構造変更機能を導入することにより、設計支援の自由度を向上させた点が特徴となる。

以下、第４章及び第５章で FDL-II の技術的内容についてより詳細に述べる。第４章は基本機能に関する部分で、FDL-I でほぼ実現されていたものである。第５章は動的構造変更機能に関わる部分である。第６章では、旋盤設計（構造固定と構造可変）、エレベータ設計を設計例題としてとりあげ、FDL の設計能力を評価する。第７章は、本論文全体の結論である。

第４章　制約対象指向言語 FDL の基本機能

本章では FDL-II の基本機能ついてより詳細に述べる。FDL-II には対象指向言語、論理型言語、制約指向言語、関係データ検索機能、スプレッドシート型インタフェイスなどの特徴がある。これらはパラメトリック設計支援を行うために導入された機能であり、おおむね FDL-I(第３章参照) ですでに実現されていた機能に相当する。

本章では、最初に FDL の言語仕様を示し、FDL の基本機能について詳述した後、FDL による設計手順、設計実験例を紹介し、問題点を指摘する。

４．１　FDL のクラス定義の概要

本節では、FDL におけるクラス定義の仕方について述べる。

FDL は **class**、**super**、**part**、**parameter**、**database**、**where**、**method**、**local**、**end** の９つのキーワード(オペレータ)を持っており、FDL の定義はこれらのキーワード以下に記述される。クラス定義は **class** で始まり **end** で終了し、この二つ以外のキーワードは省略可能である。

９つのキーワード以下には、次の内容が記述される。

class　クラス名を記述する。クラス名はクラスを識別するための名称であり、FDL 内でユニークでなければならない。

super　継承するクラスを１個以上指定する。FDL は多重継承機能を提供するので、任意の個数の継承クラスを指定できる。

part　任意の個数の部品スロットを定義することができる。

部品スロットは次のような書式で定義する。

〈part slot〉 := #〈part class〉

部品スロットが定義されているクラスからインスタンスを生成すると、そこにも部品スロット 〈part slot〉が作られ、クラス〈part class〉から生成されたインスタンスが格納される。

parameter　ここには任意の個数のパラメータスロットを定義することができる。パラメータスロットの定義は、スロット名、パラメータタイプ、単一属性制約からなる。定義の詳細については単一属性制約の節（**4.4**）を参照。

database　B 型関係データベースを定義する。定義は関係属性と検索式定義からなる。詳

細は **4.6** の B 型関係データベースの項を参照。

where 同一性制約及び多属性制約を定義する。定義の詳細についてはそれぞれの項を参照。

method メソッドの定義を行う。

local ローカル述語の定義を行う。

end クラス定義の終了を意味するキーワード。省略することはできない。

図 4.1 に FDL の書式を示す。

```
class 〈class name〉 has
  super 〈super class names〉 ;
  part 〈part slot definition〉 ;
  parameter 〈parameter slot definition〉 ;
  database 〈item definition〉; 〈retrieval formula〉;
  where 〈constraint definition〉;
  method 〈method definition〉;
  local 〈local method definition〉;
end.
```

Fig. 4.1 FDL format

FDL は設計技術者自身がプログラミングし、設計業務に活用できることを目標に、理解しやすく、書きやすい書式を採用するにした。

４．２　対象指向言語としての機能

4.2.1 データとその操作手続きのモジュール化

対象指向言語ではオブジェクトを単位としてプログラム記述を行う。オブジェクトは抽象データ型とアクターモデルの概念（**2.6**(1)）を引き継いでおり、データとその操作手続きをひとまとめにして記述される。オブジェクト内ではデータの格納場所を任意の個数定義することができ、これはスロット(slot)と呼ばれる。プログラム全体としての仕事はオブジェクト間のメッセージ交換(message passing)により進められる。メッセージを受信することにより起動される手続きはメソッド(method)と呼ばれる。オブジェクトへの仕事の依頼はメソッドを介してのみ行うことができ、オブジェクト内へのデータの直接のアクセスは禁止される。

以上述べた事は、対象指向言語一般の特徴であるが、FDL は設計言語、制約指向言語（**2.6**(2)）でもあり、また、実装言語（Prolog）の影響を受けて、論理型言語でもある。そのため、次のような技術的特徴が付け加わる。

（１）データ格納用のスロットには部品スロットとパラメータスロットの２種類がある。

前者は構造を持つオブジェクト、後者は数値、ストリングなどのパラメータ格納用である。
（2） メソッド記述を Prolog 述語により行う。メソッドは、他のメソッド、ローカル述語、システム述語を参照することができる。
（3） メソッドとは別にローカル述語を Prolog 述語を使って定義できる。メソッドはオブジェクトの内外から参照でき、クラス間で継承されるのに対し、ローカル述語はオブジェクト内でのみ参照でき、クラス間継承を行わない。
（4）メソッドと同時に同一性制約（**4.4.2**）、多属性制約（**4.4.3**）を記述することができ、制約伝播によるオブジェクト間通信を行える。
（5）すべてのオブジェクトから、システム述語を参照できる。システム述語は Prolog が提供する組み込み述語で書換えることはできない。

4.2.2 継承関係による階層構造(taxonomy)

クラスを新たに定義する場合、その内容がすでに定義されているクラスと重複している場合がある。この場合定義済みのクラスの内容を継承することにより、定義の重複を避けることができる。この継承関係は is_a 関係または a_kind_of 関係と呼ばれる。1個のクラスのみ継承を行うことを、単一継承(single inheritance)と呼び、単一継承により構成されるクラス階層は必ず木構造になる。一方、複数の定義済みクラスから継承することを多重継承(multiple inheritance)と呼ぶ。

```
class shaft
part
material:= #material;
parameter
    ext_dia integer 'mm',
    int_dia integer 'mm',
    length integer 'mm',
    bearing_dist integer 'mm',
    max_rot integer 'rpm',
    shearing real 'kg/mm2',
end.

class main_shaft
super shaft;
parameter
  max_cut_trq real 'kg.m';
where
  shearing == material!shearing;
  %tortion equation -- to calculate the diameter
  @int_dia**4.0+2.334E10*max_cut_trq/ @shearing = X,
  sqrt(sqrt(X))/5.0 = Y,
  (floating_point_to_integer(Y)+1)*5 = ^ext_dia;
end.
```

Fig.4.2　An example of inheritance

FDL は、キーワード super 以下に複数のクラス名を指定することにより多重継承が可能である。多重継承により、あらかじめ種々の性質をもつクラスをいくつか用意しておき、これらを組み合わせて新しいクラスを定義することができる。この場合、組み合わせのしかたによりクラス定義のバリエーションが容易に行える。また、同一のクラスが複数の系列の taxonomy に属する整理の仕方も可能である。

一方、多重継承によりスロット、制約、メソッドを重複定義する可能性がある。同名のスロット定義が多重継承された場合は、super 内で先に指定してあるクラスのスロット定義を優先し、重複定義はしない。メソッド、制約については重複定義を許している。メソッドが重複して定義されている場合のメソッドの選択方法は、prolog 同様のバックトラック付き縦型探索によっている。制約については、そのまま重複して適用している。

図 4.2 に FDL で記述した継承の例を示す。クラス shaft は部品とパラメータを定義しているが、関係式の記述がない。クラス main_shaft は、クラス shaft のデータ構造をそのまま継承し、さらに、パラメータ max_cut_trq とねじり剛性に関する式を追加することにより、データ定義を行っている。

4.2.3　部分と全体の関係による階層(partnomy)

クラスの階層は、全体と部分の関係によっても構成することができる。例えば、旋盤はモータ、ベルト伝動部、歯車変速機、主軸から構成され、それぞれの部位がまたそれぞれの部品から構成されるといったような階層構造で表現される。このようなデータの階層構造は普遍的な表現形態である。オブジェクト指向言語では、オブジェクト内にスロットと呼ぶデータの格納場所を設けており、ここに他のオブジェクトを格納することを許すことによりデータの階層構造を可能としている。

FDL では、部品スロット(part　slot)とパラメータスロット(parameter　slot)の 2 種類のスロットを用意している。部品スロットは構造を持つ部品、部品アセンブリ、材料など、任意のサブモデル（オブジェクト）を置くためのものである。

パラメータスロットは構造を持たない値を置くためのものである。値もオブジェクトとして表現し、'real'、'integer'、'number'、'string'、'list'の 5 種類のデータタイプがある。単位と単一属性制約(**4.4.1**)も併せて指定でき、初期値は指定があればデフォルト値、そうでなければ nil である。

4.2.4　クラス－インスタンス

オブジェクトにはクラスとインスタンスという型がある。クラスは対象の一般的、抽象的概念を表現するのに対し、インスタンスはその具体例を表現するのに用いられ、クラスから必要に応じて任意の個数生成することができる。

FDL-I ではクラスそのものの内容を表現するために、クラススロット、クラスメソッド、クラス制約を記述することができるようになっていたが、これらは実際に必要となる場面がなかった。そこで、FDL-II ではクラス固有の定義をすべてなくし、クラスはインスタンスを生成するための情報のみを持つことにした。

インスタンスは次のメッセージをクラスに送ることにより、生成される。

:create(#Class,Instance)　　(4.1)

ここでClassにはクラス名、Instanceは変数である。このメッセージをクラスClassが受け取ると、インスタンスを生成し、その名称をInstanceにユニファイして返す。インスタンスの名称はClass#Integerの形式で表現され、Integerにはインスタンス番号と呼ぶインスタンスごとに固有の整数が自動的に割り当てられる。インスタンスの参照は、Class#Integerまたは#Integerの形式で行う。

4.2.5 メソッドとブロードキャスティング

4.2.5.1 クラス class

クラス名 class を持つクラスオブジェクトがシステムとして用意されており、全てのクラスオブジェクトに暗黙のうちに継承されている。クラス class はすべてのクラスオブジェクト、インスタンスオブジェクトが持つべきメソッドを定義しており、ユーザは自由に利用することができる。

4.2.5.2 メソッドの記述形式

メソッドは次の形式で定義する。

:method_name(Object,arg1,...,argn):- goal1, goal2,..., goalm;　　(4.2)

ここで、

method_name： メソッド名
Object： メソッドを実行するオブジェクト名
arg1,...,argn： 引数
goal1,...,goalm： ゴール列。ローカル述語、組み込み述語、メソッドのいずれかを記述する。

FDLのメソッドはProlog述語とほとんど同じ形式をとっているが、次の点が異なる。

（1） 述語名の前に':'を置く。':'に続く述語はメソッド名とみなされ、ローカル述語やシステム述語と区別される。
（2） ヘッドの第1引数はオブジェクトを受け取るための変数として使用する。
（3） ヘッドの第2引数及びそれ以降の引数はメソッド呼び出し側との間でデータのやりとりを行うためのパラメータとして使用する。
（4） メソッドにはクラスメソッドとインスタンスメソッドの2種類がある。

4.2.5.3 クラスメソッド

クラスメソッドは、クラスオブジェクトが提供するメソッドで、クラス class により次のようなメソッドが提供されている。

:create(#Class,Instance) (4.3)
:show(#Class) (4.4)
:show_all(#Class) (4.5)

:create/2 (述語名:create、引数の数2の意味、以下同様) (4.3)は、Classから Instanceを生成するメソッドである。ここでインスタンスは Class#Integerの形式で表現される。:show/1 (4.4)は Classをプログラム記述と同じ形式で表示し、:show_all/1 (4.5)は Classを継承情報も含めて表示する。例えば**図 4.2** に示すクラス main_shaft を:show/1 により表示した場合は、**図 4.3** に示すように、プログラムと同形式となるが、:show_all/1 で表示した場合は**図 4.4** に示すように、継承情報も含めて全ての情報が表示される。

クラスメソッドはクラス class が提供するもののみ利用可能で、ユーザが定義することはできない。

```
FDL| ?-:show(#main_shaft).

class main_shaft
super shaft;
parameter
   max_cut_trq real 'kg.m';
where
   shearing==material!shearing;
   @int_dia**4.0+23340000000.0*max_cut_trq/ @shearing=_1065,
   sqrt(sqrt(_1065))/5.0=_1035,
   (floating_point_to_integer(_1035)+1)*5= ^ext_dia;
end.

:show(#main_shaft)
yes

FDL| ?-
```

Fig. 4.3 Class main_shaft displayed without inheritance

```
FDL| ?-:show_all(#main_shaft).

     class main_shaft
     super shaft;
     part
       material:= #material;
     parameter
       ext_dia integer 'mm',
       int_dia integer 'mm',
       length integer 'mm',
       bearing_dist integer 'mm',
       max_rot integer 'rpm',
       shearing real 'kg/mm2',
       max_cut_trq real 'kg.m';
     where
       shearing == material!shearing;
       @int_dia**4.0+23340000000.0*max_cut_trq/ @shearing=_2755,
       sqrt(sqrt(_2755))/5.0=_2725,
       (floating_point_to_integer(_2725)+1)*5= ^ext_dia;
     end.


     :show_all(#main_shaft)
     yes
     FDL| ?-
```

Fig. 4.4 Class main_shaft displayed with inherited information

4.2.5.4 インスタンスメソッド

インスタンスメソッドはキーワード'method'以降に定義することができる。クラス class が提供するインスタンスメソッドとして次のようなものがある。

```
:set(Instance,Slot,Value)                    (4.6)
:get(Instance,Slot,Value)                    (4.7)
:show(Instance)                              (4.8)
:show_all(Instance)                          (4.9)
:show_structure(Instance)                    (4.10)
```

:set/3 (4.6)は Instance の Slot の値を読みだし、Value にユニファイする。
:get/3 (4.7)は Instance の Slot に値 Value を代入する。
:show/1 (4.8)は Instance の部品スロット、パラメータスロット及びそれらの値を表示する。スロットは継承したものも含む。
:show_all/1 (4.9)はスロット、同一性制約、多属性制約を表示する。
:show_structure/1 (4.10)は:show_all/1 を部品スロットにセットされているオブジェクトに対して再帰的に適用し、インスタンスの部品階層構造全体を表示する。

4.2.5.5 ブロードキャスティング

クラス FDL は放送局の役割を持ち、全てのオブジェクトに対して同時にメッセージを発信することができる。これはブロードキャスティングと呼ばれ、クラス fdl に依頼することにより全オブジェクトに関係する操作を行うことができる。

以下のブロードキャスティグを用意している。

:undo(#fdl)	(4.11)
:undo(#fdl,Num)	(4.12)
:initialize(#fdl)	(4.13)
:history(#fdl)	(4.14)
:show_class_names(#fdl)	(4.15)

:undo/2 (4.12)はデータ入力及びそれに伴って行われる制約またはメソッド起動によるデータ変更を Num ステップ分、アンドゥーする(操作前の状態に戻す)。

:undo/1 (4.11)は:undo/2 において Num=1 としたものと同等のものである。

:initialize/1 (4.13)はメタオペレーションを除くすべてのデータ入力操作を行う以前の状態に戻す。:history/1 (4.14)は過去のデータ入力やメタ操作(5 章参照)の履歴を表示する。:show_class_names/1 (4.15)はインタープリター上にロードされている全てのクラスオブジェクト名を表示する。ブロードキャスティングはクラス fdl が提供するもののみ利用可能で、ユーザが定義することはできない。

4．3 論理型言語としての特徴

FDL は論理型言語としての性格も持ち、メソッドとローカル述語は述語論理のホーン節により表現される。Prolog とほぼ同じ書式を用いるが、述語の終了記号に'.'の代わりに';'を用いる。また、メソッドは頭部に前置子':'を付ける。

メソッド、ローカル述語のいずれも Prolog が提供する述語(Prolog の組み込み述語とユーザが定義した述語)を利用でき、システム述語と呼ぶ。前置詞'sys:'を付けて、ローカル述語と区別する。

論理型言語では、アトム、論理変数、述語名の表現に文字列が使われるが、FDL では、英文字、日本語文字、数字等を文字列の要素に用いることができる。アトム及び述語名は

英小文字または日本語文字から始まる文字列で表現する。論理変数は英大文字または'_'で始まる文字列である。

４．４　制約言語としての FDL

FDL の制約には、単一属性制約、同一性制約、多属性制約の３種類があり、データの整合性管理や設計知識表現に用いられる。単一属性制約は受動的制約であり、データ整合性検証の役割を持つ。同一性制約、多属性制約はオブジェクト間を伝播し、メッセージ交換に代わってオブジェクト間の情報伝達の役割を果たすことができる。

以下、それぞれの制約について詳述する。

4.4.1　単一属性制約

単一属性制約（single attribute constraint）は、パラメータの存在範囲、単位デフォルト値などを規定する制約である。受動的制約であり、データ整合性の検証の役目を持つ。パラメータスロット定義部において、次のような書式で記述する。

〈parameter name〉〈type〉〈single attribute constraint〉

単一属性制約は**表 4.1** に示す要素のリストとして表現される。(1)は単位の指定、(2)はデフォルト値の指定、(3)～(6)は値の存在範囲の指定、(7)は存在を許す値の数え上げ集合の指定、(8)は val1 で割った剰余が Val2 となる値であることを示す。

Table 4.1　Elements for single attribute constraints

(1)　unit (U)	(2)　default (Value)
(3)　〉Value	(4)　〈 Value
(5)　〉= Value	(6)　=〈 Value
(7)　member (ValueList)	(8)　mod (Val1) = Val2

例えば、

rotation integer [unit (rpm), member ([3000,4000,5000])]

は、パラメータ'rotation'の単位は rpm、値は 3000、4000、5000 のいずれかの整数であることを表現している。同様に、

diameter integer [unit (mm), default (60), 〉=20, mod (5) =0]

は、パラメータ'diameter'は、単位は mm、20 以上の 5 で割り切れる整数でデフォルト値が 60

であることを表現している。

4.4.2 同一性制約

同一性制約（identity constraint）は**図 4.1** の'where'以下に定義される。この制約は二つのスロットに存在するオブジェクトが同一であることを宣言するためのものであり、二つのスロットは一つのオブジェクトを共有（structure sharing）することにより表現される。同一性制約の書式は次の通りである。

[<slot name>{!<slot name>} == <slot name>{!<slot name>}]

ここで { } は、0回以上任意の回数繰り返しても良い事を意味し、!は'a_part_of'関係を表す。

同一性制約は二つ以上任意の個数のスロット間で連鎖的に設定可能である。二つ以上のパラメータスロット間で設定する場合、それらすべてのスロットのデータタイプと単位が一致している必要があり、これはコンパイル時に検証される。

モデルをモジュール構造として表現する上で同一性制約は大変有用である。**図 4.5(a)** に示すような制約関係で関連づけられたパラメータは一見モジュール群に分解できないように見えるが、**図 4.5(b)** のように同一性制約を利用することにより分解することが可能である。逆に見ると、あらかじめ用意されたモジュール群を同一性制約で接続することにより、全体モデルを定義し、大域的制約ネットワークを構成することができる。このようにして、モデルのアセンブリ定義にも用いられる。

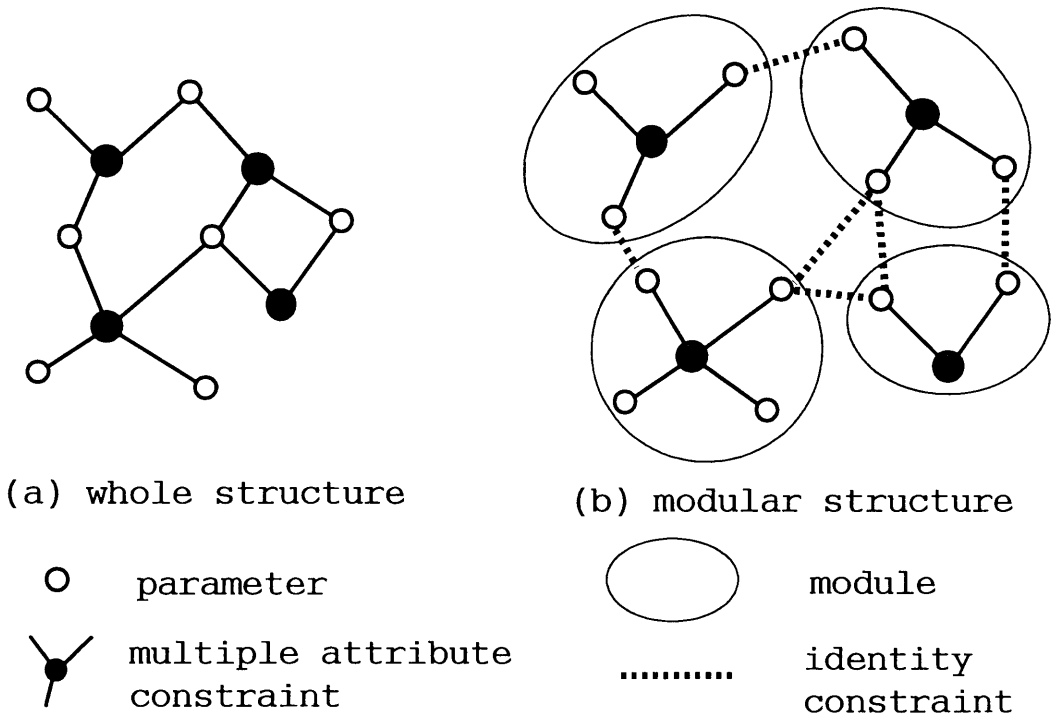


Fig. 4.5 Modularization of a data structure by introducing identity constraints

4.4.3 多属性制約

多属性制約（multiple attribute constraint）は同一性制約と同様、**図 4.1** の'where'以下に定義される。２個以上のパラメータ間の制約関係を記述するためのものであり、データの整合性管理、設計知識の表現等に用いられる。

多属性制約は前提条件部と制約本体から構成され、

[<precondition> ==>] <constraint body>;

のように記述される。前提条件部(precondition)は制約起動のための前提条件を記述する部分である。前提条件部（[]で囲った部分）は省略することも可能で、その場合、制約はそれを構成するパラメータに変化があった場合無条件で起動される。前提条件部、制約本体いずれも、Prolog 述語、等式、不等式を用いた表現ができる。このうち等式と不等式は制約伝播法[Sussman & Steel, 80]を用いた制約解決器により解かれ、関数として平方根、三角関数、対数、ｎ乗計算、剰余等を扱うことができる。解決器は:eval(#freeze,X)で呼び出すことができ、X に数式を入力すると、数式中の変数に結果を返す。解が不定の場合は変数をそのまま返す。不能の場合は呼び出しが失敗し、パラメータ値は元の値に戻される。

多属性制約は Prolog 述語、等式、不等式を同時に扱うことができるので、多くの応用が可能である。例えば、(1)制約と関係データを組合わせた関係データ検索（**4.6** 参照）、(2)述語によるニュートン法などのアルゴリズムの記述、(3)数値とストリングの混合演算、などが可能である。

多属性制約は、次の表記要素を使って表現する。

アトム　パラメータスロット名。パラメータスロットの値を参照する変数として扱われる。同一性制約と同様’アトム!...!アトム’の形式により、部品モデルのパラメータスロットを参照することも可能。

**`アトム**　固有名詞。前置演算子`がアトムに付加された場合、アトムはパラメータスロット名ではなく、固有名詞とみなされる。

論理変数　英大文字または'_'で始まる文字列。制約表現内の一時変数として使用。

+, - , *, /　加減乗除を表す演算子

>, >=, <, =<　不等式を表す演算子

sqrt(X)　平方根を求める関数

ceiling(X)　X より大きく X に最も近い整数を求める関数

floor(X)　X より小さく X に最も近い整数を求める関数

integer(X)　X と 0 の間で最も X に近い整数を求める関数

sin(X),cos(X),tan(X)　三角関数

asin(X),acos(X),atan(X)　逆三角関数

****(X,Y)**　X の Y 乗を求める関数

log(X)　X の自然対数

log(Base,X) Base を底とする X の対数
bigger(X,Y) X、Y のうち大きい方を返す関数
smaller(X,Y) X、Y のうち小さい方を返す関数
mod(X,Y) integer(X) を integer(Y) で除算した剰余を求める関数
= 左辺または右辺のいずれかが式として表現されていれば、左辺と右辺の等値関係を表す。一方がパラメータスロット名、他方がパラメータスロット名または値である場合、ユニフィケーションを表す。
@, ^ スロット名につけられる前置演算子。
述語 ローカル述語またはシステム述語を利用できる。システム述語の場合は述語の前に前置演算子'sys:'をつけて区別する。

前置演算子なしスロット名は入出力双方が可能な入出力変数、@付スロット名は入力変数、^付スロット名は出力変数として扱われる。制約は入力変数または入出力変数に値が入力された場合起動され、解を求めることができた場合はその結果が入出力変数並びに出力変数に返される。

@及び^はデータ入力順序の指針のために、ウインドウ上のスロットにも表示される。@付スロットは入力のみのスロットなので、制約解決の実行のため優先的に入力を行うとよい。また、^付スロットは入力禁止である。同一性制約により共通の値を共有するスロット群の内一つでも@または^の対象になっている場合は、そのすべてのスロットで@または^を表示する。同一の変数に@と^が二重に指定された場合はユーザの立場からは^の指定が優先され、ユーザからの入力は禁止、スロット上には^が表示される。例えば、

```
比切削抵抗 * @最大切り込み面積 = ^最大切削力;
@最大切削力 * @最大切削速度 = ^最大切磋駆動力;
```

の例では、変数 "最大切削力" に@と^が二重に指定されており、ユーザからの入力は禁止されスロット上には^が表示される。この^が優先される仕組みは、大規模な制約ネットワークにおいて入力変数の数を減少させる上で有効である。**6.3** エレベータ設計の例では、この仕組みにより入力変数の数を大幅に減少させることができ、解探索の計算時間を短くするのに効果的であったと思われる。

ウインドウから [] で囲って値を入力すると、これは制約内で定数として扱われる。また () で囲って入力すると、[] なしの状態へ戻すことができる。[] で囲った値の入力と@付スロットへの入力は、いずれも制約内で定数として扱われる点で同じであるが、[] は固定的定数（設計仕様など）であることを明示するためのものであり、@付きスロットの値は暫定的な値と見なされる。また、[] 指定はウインドウ上から変更できるが、@指定は変更できない。

多属性制約が、左辺が多項式、右辺がパラメータスロット名を表すアトムのみで構成される等式として表現されている場合、左辺を入力、右辺を出力とする関数として扱う。

多属性制約の一例として歯車ペアの物理的関係を記述した例を示す。

```
歯車１!外径+歯車２!外径= @軸間距離*2;
```

この例では、軸間距離の値は定数として、歯車１の外径及び歯車２の外径は入出力変数として扱われる。従って、軸間距離の値がすでに決まっていれば、歯車１の外径または歯車２の外径の一方の値が与えられると、他方の値を自動的に計算する。軸間距離の値を変更した場合は、歯車１、歯車２の外径の値は不定となる。

次に、関数型制約の例を示す。

```
car_weight + control_cable_load_average + 0.4 * platform!car_capacity =
counterweight_weight_desired;
```

この例は、付録５にあるエレベータ設計モデルからとったもので、左辺の３変数を入力、右辺の１変数を出力として扱う関数型制約である。

次に、前提条件付き制約の例を示す。

```
platform!model_id = `platform_model_m01 ==>
sys:member(sling!model_id,[`sling_model_m01, `sling_model_m02]),
compensation_cable_load_car_side_car_top =< 600;
```

この例も付録５からとった、前提条件付き制約である。==>の左辺が成立している場合、右辺も成立しなければならない。すなわち、platform!model_id の値が platform_model_m01 であった場合、sling!model_id が sling_model_m01, sling_model_m02 のいずれかの値をとり、compensation_cable_load_car_side_car_top の値が 600 以下でなければならない。

各モデルで記述された多属性制約は、同一性制約により関係付けられており、一つが起動されると他の制約に伝播していく。１回の制約伝播で同一の制約が再び呼び出されたとき、それを止めるようになっているので、伝播がループに陥ることはない。

4.4.4 コマンドファイル

設計制約は、要求仕様をパラメータの値として与えて解くが、入力すべきパラメータ値の数が多い場合は、作業量が大きくなる。プログラムのデバッグ作業等では、要求仕様の入力作業を繰り返しやり直す必要が生じ煩わしい。このような繰り返し行う可能性のある作業はファイルに登録し、一括して実行できるようにすることが望ましい。FDL では、ファイルに実行コマンド列を登録しておき、そのファイルを command_file(ファイル名)という述語で呼び出すと、コマンド列を自動的に実行できるようになっている。要求仕様の入力は:set(Object, ParameterSlot, Value)を多数実行することにより実行する。

4.4.5 生成検証法

設計制約は要求仕様が与えられても、解を決定的に求めることができない場合が多い。これは、設計が過小制約問題で、解が二つ以上存在する場合があるからである。過小制約問題に対しては、線形計画法、ダイナミックプログラミングなどの OR 解法があるが、これらは対象問題が方程式により静的に記述された問題でないと適用が難しい。FDL の制約は、方程式以外に述語による表現を扱い、また前提条件付き制約記述があるので制約は条件により動的に変化するので、OR 解法の適用は困難である。

FDL は過小制約の解法として、非線形ないし動的に変化する制約にも適用可能な生成検証法を採用している。これは、未決定のパラメータ群に順次仮定の値を発生させては入力し、制約を満たすことができる仮定値の組合せを探し求める方法である。どのパラメータが未決定入力パラメータになるかは要求仕様に依存し、ケースにより異なるので、設計モデル内に生成検証法のアルゴリズムをあらかじめ登録しておく方法は適切ではない。そこで、設計の状況に合うようにユーザー自ら定義した生成検証法アルゴリズムを、任意の時点、任意の回数、登録実行できるようにした。

生成検証法のアルゴリズムは次の述語により定義する。

config(クラス名,ID,[生成器のリスト])

ここで、クラス名は生成検証法を適用する対象クラス、ID はこのアルゴリズムを識別するためのものである。生成器は次の 3 種類が利用できる。

part(部品スロット,述語制約),
member(パラメータ,候補値のリスト)
generator(パラメータ,[最小値,最大値],ステップ幅)

part(部品スロット,述語制約)は、部品スロットに格納されているオブジェクトの述語制約を満たす値を自動生成するためのものである。例えば、**図 4.6** に示すクラスオブジェクトのインスタンスがスロット crosshead に格納されており、その model_id の値を順次生成する場合は次のように記述する。

part(crosshead,data(_,model_id))

この生成器は、ローカル述語から model_id と crosshead_height の値を読み込み、制約ネットワークにその値を伝播させる。もし、制約で矛盾が検出されたら、次のローカル述語を読み込み、以降同じことを繰り返す。すべてのローカル述語による値の生成が制約に受け入れられなかったら、この生成器の実行そのものが失敗する。

member(パラメータ,候補値リスト)は、候補値リストから候補値を順番に選んで、パラメータに入力する生成器である。

generator(パラメータ,[最小値,最大値],ステップ幅)は、最小値から始まって、ステップ

幅分だけ順次増やした値を、最大値を越えない範囲で生成し、パラメータに入力する生成器である。

```
class crosshead_model
super vt_component;
parameter
  crosshead_height real [],
  model_id string [];
where
  data(^crosshead_height, @model_id);
local
  data(10.0,crosshead_model_m01);
  data(13.5,crosshead_model_m02);
  data(8.0,crosshead_model_m03);
  data(8.125,crosshead_model_m04);
  data(8.25,crosshead_model_m05);
end.
```

Fig. 4.6 An example of predicate constraint

次に、生成検証法の記述例を示す。

```
config(elevator,c1,
        [part(crosshead,data(_,model_id)),
         member(counterweight_plate_depth,[7.0,9.0,11.0,12.0]),
         generator(counterweight_frame_height,[90.0,170.0],10.0)])
```

この例では、part/2 で 5 種類、member/2 で 4 種類、generator/3 で 9 種類の値が生成され、計 5*4*9=180 通りの値の組合せが生成される。この生成検証法では、それぞれの生成器がリストの順に値を生成し、そのたびに制約計算による生成値の検証を行う。すべての生成器が起動されていない段階でも、それまでに生成された値の組合せが制約を満たせないことがわかると、以降の生成器の起動は省略され、バックトラックして、次の値を生成する。すなわち、この生成検証法は Prolog と同様のバックトラック付き縦型優先探索法をとり、無駄な探索は回避できるようになっている。

生成検証法はアルゴリズムをファイルに記述し、load_config(ファイル名)を入力し、続いて:config(インスタンス名,ID)を入力すると、実行される。

4.4.6 制約解決支援のための why 機能

FDL は制約解決に問題が生じた場合に、値を決めた根拠となる制約を表示する why 機能を提供している。この機能を利用して、問題を起こす制約を探し出すことができる。

why 機能は次のメソッドで起動する。

:why (Instance, Slot) (4.16)

また、why 機能を起動するためのグラフィックインタフェイスも用意している (**図 2.4**)。

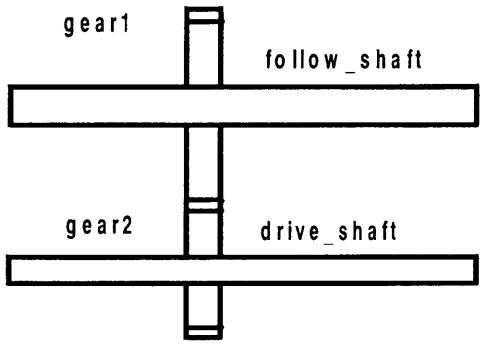


Fig. 4.7 Gear pair

```
class gear_pair has
instance
part
  gear1:= #spur_gear, gear2:= #spur_gear,
  drive_shaft:= #shaft, follow_shaft:= #shaft;
parameter
  gear_ratio real [=< 4.0, >= 0.25],
  gear1_dia integer [unit("mm"),>=50],
  gear2_dia integer [unit("mm"),>=50],
  shaft_distance integer [unit("mm"),>50];
where
  %%% identity constraints
  gear1_dia==gear1!ext_dia, gear2_dia==gear2!ext_dia,
  gear1!module==gear2!module, gear1!width==gear2!width,
  gear1!int_dia==follow_shaft!ext_dia,
  gear2!int_dia==drive_shaft!ext_dia;
  %%% mutliple attribute constraints
  gear1_dia+gear2_dia = @shaft_distance*2;
  gear1_dia/gear2_dia = @gear_ratio;
end.
```

Fig. 4.8 Gear pair assembly model description

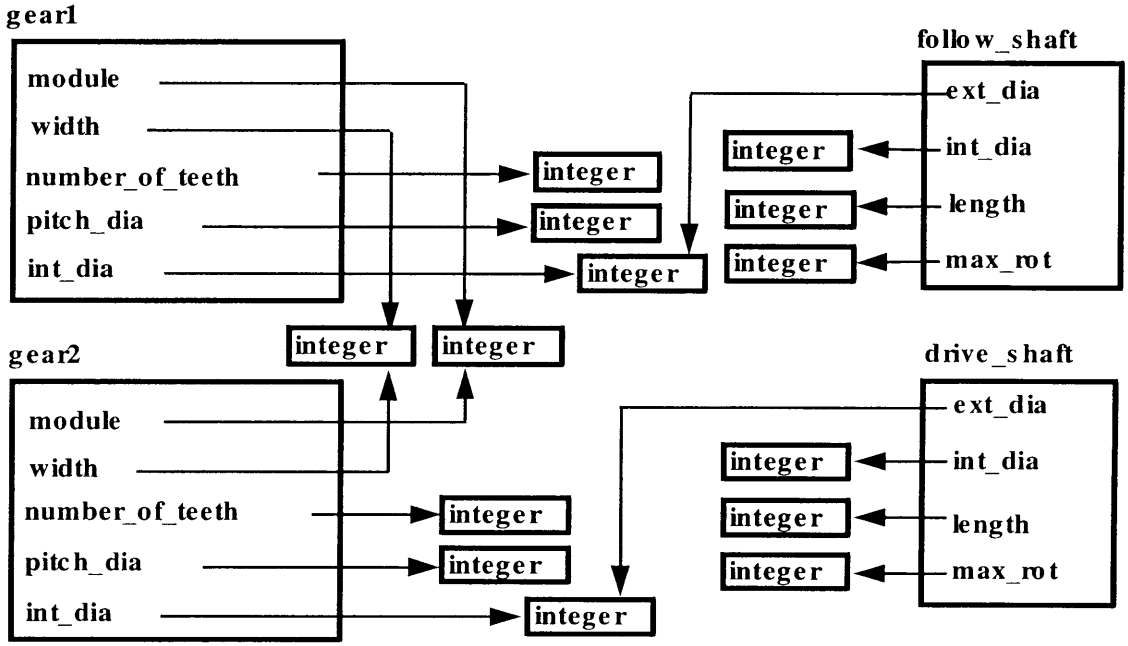


Fig. 4.9 Gear pair model data structure

４．５　モデルのアセンブリ

FDL ではサブモデルを組み合わせてアセンブリモデルを定義することができ、アセンブリモデルもサブモデルとして再帰的に利用できる。アセンブリモデルを構成するサブモデル間の関係は同一性制約と多属性制約により表現される。同一性制約は主にモデル間の物理的接続関係を表現し、多属性制約は物理的ルールやアセンブリの条件などを表現する。パラメータはアセンブリすることにより必要となる新たな属性を表現するために導入される。

アセンブリモデルの例として、**図 4.7** に示すギア・ペアをとりあげる。このアセンブリモデルの FDL による記述は**図 4.8** のとおりである。これは、２個のギアと２本のシャフトから構成されており、４個のパラメータ、６個の同一性制約及び２個の多属性制約を新たに定義することによりアセンブリ関係を表現している。**図 4.9** にアセンブリモデルのデータ構造を示す。同一性制約が構造共有により表現される様子が示されている。

４．６　関係データベース機能

FDL は物理定数、標準、カタログなどの表形式のデータを扱うために関係データベースの検索機能を提供している。FDL のデータベースにはＡ型、Ｂ型の二種類がある。

Ａ型関係データベースは、関係データをローカル述語により表現し、多属性制約を用いて関係データとパラメータスロットの関係を検索式として記述するものである。データ検

索用の制約式への値入力をトリガーとして自動的に検索を行うことができる。A型関係データベースは、多属性制約のネットワークに完全に組み込まれている。すなわち、制約伝播によりデータ検索用の制約式が起動され、検索結果は他の制約に伝播する。ユーザインタフェイス用ウインドウは一般のモデルと同じ形式のものを使う。

キーのJIS規格データをA型関係データベースで表現した例を**図4.10**に示す。この例では軸径(shaft_dia)の値を与えると、それに適したキーをローカル述語列から検索できるようになっている。

A型関係データベースは生成検証法（generate and test）におけるデータ生成器として利用することもできる。付録2のclass max_spd_spur_gearの中に、moduleの値を生成検証法により求めている例がある。

```
class key
parameter
      shaft_dia integer 'mm',
      width       integer 'mm',
      height      integer 'mm',
      min_length   integer 'mm',
      max_length integer 'mm';
where                %to select a key
      data(X,Y,^width,^height,^min_length,^max_length),
      shaft_dia >= X,shaft_dia < Y;
local
      data(13,20,5,5,10,56);
      data(20,30,7,7,14,90);
      data(30,40,10,8,18,112);
      data(40,50,12,8,22.4,140);
      data(50,60,15,10,28,160);
      data(60,70,18,12,35.5,200);
                  :
                  :
end.
```

Fig. 4.10　A type relational database（material data）

B型関係データベースは、専用ウィンドウを使い、コマンドを使って会話的に検索を行うものである。この場合も要求を表現したパラメータと選択対象属性との関係を表現した検索式をあらかじめ登録しておき、自動検索を行うことができる。検索結果は専用ウインドウに表示され、選択されたデータはすべて設計解候補とみなすことができる。検索結果は制約ネットワークに返されない。

B型関係データベースの定義は次の形式で行う。

database
〈関係データ項目定義列〉;
〈関係データ検索式〉;

この中で、〈関係データ項目定義列〉は次の形式で定義する。

〈項目名〉〈データタイプ〉〈単位〉,
〈項目名〉〈データタイプ〉〈単位〉,
:
〈項目名〉〈データタイプ〉〈単位〉;

〈項目名〉は、それぞれが関係データベース内でユニークであればよく、パラメータスロ

```
class bearing
parameter
        int_dia              integer 'mm' ,
        max_ext_dia          integer 'mm',
        mean_d_load          integer 'kgf',
        max_rot              integer 'rpm',
        avg_rot              integer 'rpm',
        life_time            integer 'hour';
database
        name                   string,
        int_dia              integer 'mm',
        ext_dia              integer 'mm',
        width                integer 'mm',
        round                real    'mm',
        d_load               integer 'kg',
        s_load               integer 'kg',
        g_max_rot            integer 'rpm',
        o_max_rot            integer 'rpm',
        mass                 real    'kg';
%   retrieval formula
        int_dia = d!int_dia,
        max_ext_dia >= d!ext_dia,
        max_rot =< d!g_max_rot,
        life_time =< (d!d_load/mean_d_load)**3.1666/avg_rot;
local
        data(['6800',10,19,5,0.5,175,75,38000,45000,0.0056]);
        data(['6900',10,22,6,0.5,275,117,34000,40000,0.0096]);
        data(['6000',10,26,8,0.5,465,196,30000,36000,0.0190]);
        data(['6200',10,30,9,1.0,520,229,24000,30000,0.0320]);
        data(['6300',10,35,11,1.0,825,365,22000,26000,0.0530]);
                                :
                                :
end.
```

Fig.4.11　B type relational database（bearing）

ット名、部品スロット名と重複してもよい。〈データタイプ〉はinteger、real、number、stringのいずれかを指定する。〈単位〉は任意のものを指定できる。

〈関係データ検索式〉は関係データ項目とパラメータスロットの値の関係を等式、不等式を使って記述するものである。パラメータスロットと区別するため、関係データ項目名には前置詞'd!'を付ける。関係データ検索式は、、パラメータの値が十分に与えられなくても、式の中で参照されているパラメータの値が与えられるたびに検索を実行する。検索の結果は、専用のウインドウに表示される。

軸受データをＢ型関係データベースで表現した例を**図 4.11** に示す。

４．７　インタプリタ

FDL はインタプリタ言語であり、現在は Prolog［Andersson et al., 94］をその記述言語及びオブジェクト・コードとして使用している。

インタプリタは、FDL を実行コードに変換するトランスレータとオブジェクトコードの実行を支えるシステムプログラムから構成される。トランスレータは文法解析部、制約方程式解析部とコード生成部から構成される。文法解析部は構文解析に基づく文法エラー検出を行い、コード生成部に引き渡すデータ（プログラムの項目ごとに分類されたリスト）を作成する。構文解析はオペレータのタイプ及び優先順位（operator precedence）を利用しており、付録１に示す FDL の記述文法に従ってプログラムを解読する。オペレータ優先順位表は付録２に示す。制約式の構造解析に関しては、特別な処理が必要となるので、別のモジュール（equation analysis）で行われ、生成されたデータは同様にコード生成部に引き渡される。

コード生成部は、文法解析部から受け取ったデータをもとに Prolog コードを実行コードとして生成する。実行コードでは、一つのクラス／インスタンスオブジェクトが Prolog 述語の集合により表現されているが、ユーザからはその内容は直接は見えない。実行コードのより詳細な内容については、付録３の FDL の内部表現の項を参照されたい。

実行プログラムはシステムプログラムと関係づけられて、解釈、実行される。システムプログラムはウィンドウ管理プログラム（window manager）、関係データベース管理プログラム（relational database manager）、制約解決器（constraint solver）、オペレータプロセッサー（operator processor）、メタオペレータインタプリタ（meta-operator interpreter）から構成される。ウインドウ管理プログラムは、ユーザインタフェイス（**4.8** 構造化スプレッドシート型インタフェイス、**5.7** 動的構造変更のためのグラフィックインタフェイス）の生成管理を受け持つ。関係データベース管理プログラムはＢ型関係データベースの検索アルゴリズムを受け持つ。なお、Ａ型関係データベースは、一般の制約記述とともに制約解決器を介して処理される。制約解決器は制約記述のうち、方程式、不等式について処理する部分である。述語、ルール記述は Prolog 処理系そのものにより処理される。オペレータプロセッサーは **4.2.5** で述べたメソッド、ブロードキャスティングの処理を受け持ち、メタオペレータプロセッサは５章で導入される各種メタオペレータ（構造融合、構造交換、スロット／制約の追加削除など）の処理を受け持つ。 インタプリタの構成を図 **4.12** に示す。

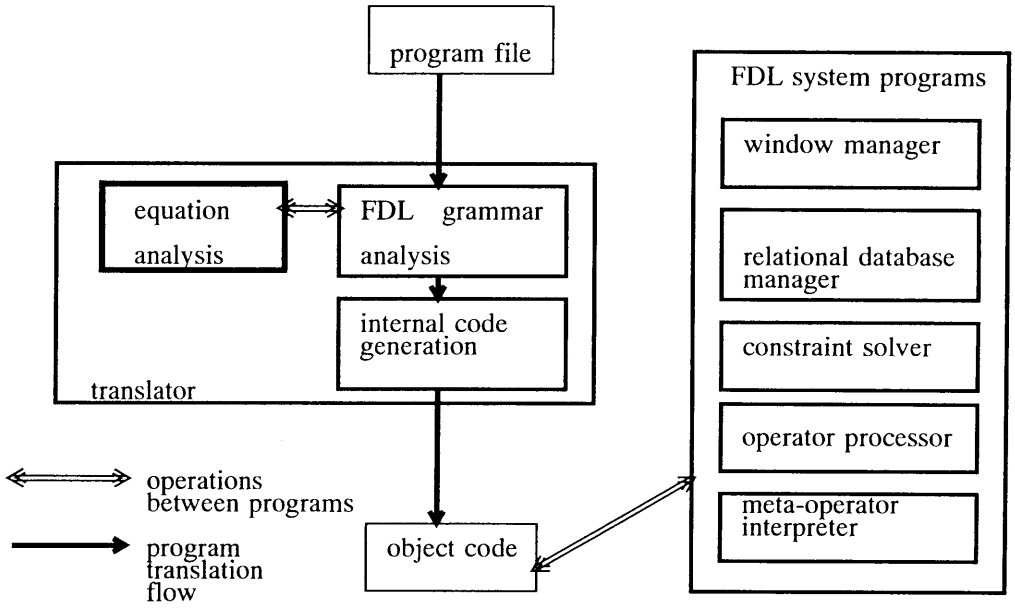


Fig. 4.12 FDL program interpretation

４．８　構造化スプレッドシート型インタフェイス

オブジェクトごとに、その部品スロットとパラメータスロットの構成を反映したウインドウが、インタフェイスとして生成され、任意の場所に表示することができる。ウインドウのスロット間に定義された制約は、データ入力によって起動され、値の自動決定／修正が行われる。ウインドウ・スロットには入力専用、出力専用、入出力用の３タイプがあり、入力専用スロットは@、出力専用スロットは＾のマークが表示される。これらのタイプは制約定義により自動的に決まる。また、[] でくくって入力した値は定数とみなされ、制約起動によっても変更されない。**図 4.13** に、生成されたインタフェイスの例を示す。部品スロットは ON-OFF 表示になっており、ON を選択するとその部品を表すウインドウが現れ、任意の位置に配置できる。パラメータスロットはウインドウ上でウインドウ・スロットとして表示される。

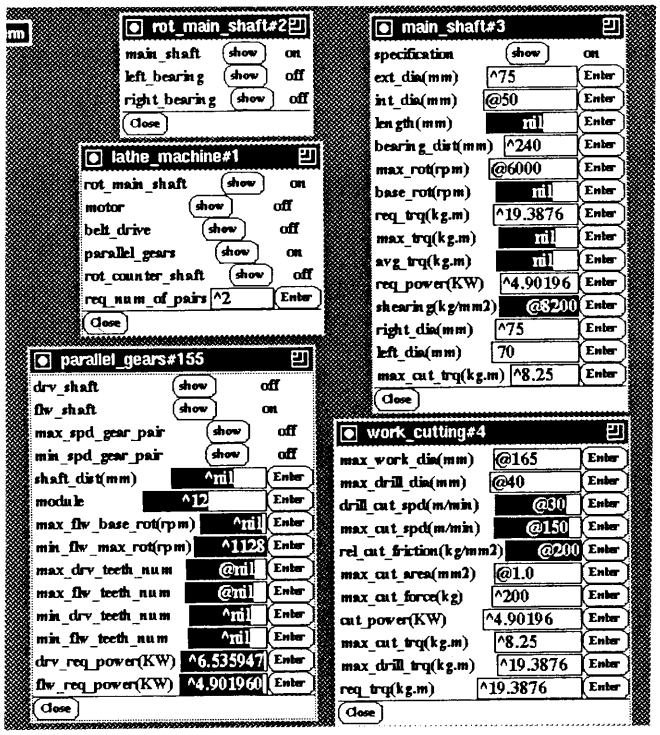


Fig.4.13　An example of structured spread sheet interface

４．９　FDL による設計手順

FDL では、以下に示すように、(1)対象モデリング、(2)モデルコンパイル、(3)仕様入力、(4)会話型設計の手順を踏んで設計を進めるのが一般的である。

（１）対象モデリング

対象モデルを定義する段階である。(I)サブモデルの準備、(II)サブモデルのアセンブリ、(III)整合性管理用制約の記述、(IV)設計知識の記述の段階からなる。

サブモデルの準備は、(a)ライブラリにあるものをそのまま利用する、(b)継承機能を使って記述を付加する、(c)テンプレートとして利用し一部手直しする、(d)新たに書く、のいずれかの方法によって行う。サブモデルのアセンブリの段階では、パラメータ間の同一性制約と多属性制約によりモデル間の関係を記述する。モデル単位で記述された多属性制約はアセンブリにより互いに結合され、大域的ネットワークを構成する。

（２）モデルコンパイル

FDL でかかれたモデルをインタプリターにロードし、Prolog に変換する段階である。シンタックス・チェックと同一性制約におけるデータタイプと単位の一致をチェックする。

（３）仕様入力

プログラムを実行し、設計用のウインドウが作成された段階で、要求仕様を値としてウインドウの適切なスロットに入力する。これらの値は、[] で囲って入力することにより制約解法の中で定数として扱われる。

（４）会話型設計

マルチウインドウを介して、会話型の設計を進める。パラメータ入力は@付スロットを優先的に行い、＾付スロットは入力禁止である。入力された値は、まず単一属性制約によるチェックを受け、次に多属性制約に引き渡される。多属性制約はサブモデルをまたがって、複数のモデル間の関係を制御し、同時に多数の値を変更する。

なお、この設計手順は FDL の基本機能（４章）のみを使用し、動的構造変更機能（５章）を利用しない場合のものである。動的構造変更機能を利用すれば、設計手順の自由度は高くなり、必ずしもこの手順に従わなくても良くなる。

４．１０　簡単なモデリングの例

ここでは設計例題として片持ち梁の応力ひずみ解析をとりあげ、FDL により設計モデリングおよび計算の実行の様子を示す。より複雑で大規模な設計例題は６章でとりあげるので、そちらを参照されたい。

4.10.1　等分布荷重を受ける片持ちばりのモデル記述

図 4.14 に示す長さｌ(mm)、断面２次モーメントＩ、縦弾性係数Ｅの片持ちばり全体に等分布荷重ｗ(kfg/m)が作用しているとき、最大せん断力（kgf)、最大曲げモーメント(kgf･mm)ならびに先端からｘ(mm)の位置のせん断力、曲げモーメント、たわみを求める。片持ちばりの材料としては、鋼鉄、鋳鉄、アルミ、銅、黄銅のいずれかを選択できるようにする。計算式は次の通りである。

最大せん断力　Ｆ max ＝　ｗ･ｌ
位置ｘにおけるせん断力　Ｆｘ＝　-ｗ･ｘ
最大曲げモーメント　Ｍ max ＝　-ｗ･ｌ^2／２
位置ｘにおける曲げモーメント　Ｍｘ＝ -ｗ･ｘ^2／２
位置ｘにおけるたわみ　ｖ＝ ｗ･ｘ^4／24EI

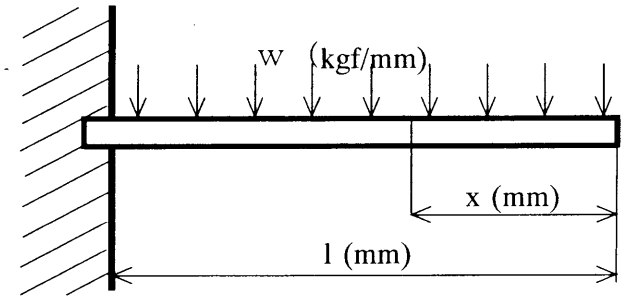


Fig. 4.14　　Cantilever beam

この例題は一つのオブジェクトですべてを記述してもよいが、オブジェクトの再利用性を考慮し、片持ちばりと材料特性に関する二つのモデルに分けて記述することにする。その場合のオブジェクト構成は**図 4.15** のようになり、片持ちばりモデルの部品（part 部）として材料モデルを持つ構造をとる。材料モデルは種々の材料の特性と検索式を記述する。

図 4.15 のオブジェクト構成を FDL で記述すると**図 4.16** のようになる。片持ちばりのモデルは part 定義、parameter 定義、制約定義で構成され、制約として片持ちばりの応力やひずみに関する式を関数型制約(**4.4.3**)形式で記述してある。材料モデルはＡ型データベース（**4.6**）形式で記述している。鉄鋼、鋳鉄、アルミ、銅、黄銅の材料特性をローカル述語により記述し、材料名を指定すると材料特性値を各パラメータに返す。材料モデルのパラメータ elasticity と片持ちばりモデルのパラメータ elasticity は、同一性制約により常に同一の値を持つようになっている。FDL 記述で用いたモデル名、パラメータ名と、**図 4.15**

で用いた名称の対応を表 **4.1** に示す。

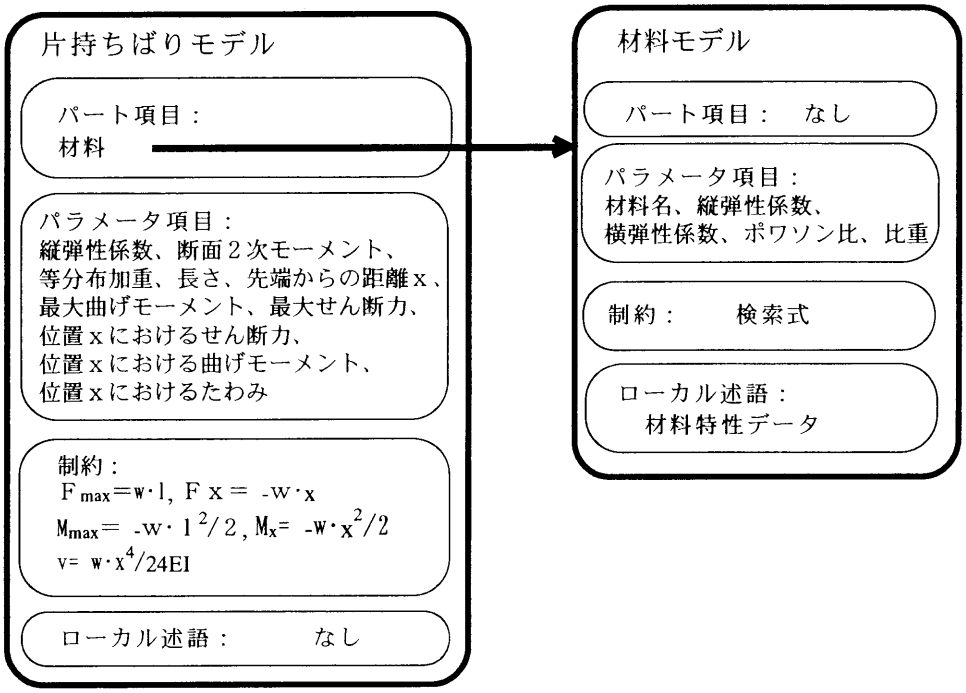


Fig. 4.15 Object structure of the cantilever beam model

Table 4.1 Meaning of parameter names in the cantilever beam model

FDL モデル上の名称	意味
material	材料モデル
name	材料名
elasticity	縦弾性係数
shearing	横弾性係数
poisson	ポワソン比
spc_gravity ;	比重
beam_model	片持ちばりモデル
elasticity	縦弾性係数
inertia	断面２次モーメント
uniform_load	等分布加重
length	長さ
position_x	先端からの距離ｘ
max_bending_moment	最大曲げモーメント
max_shearing_force	最大せん断力
shearing_force_at_x	位置ｘにおけるせん断力
bending_moment_at_x	位置ｘにおける曲げモーメント
deflection_at_x	位置ｘにおけるたわみ

```
class material
parameter
      name string [],
      elasticity real 'kgf/mm2',
      shearing real 'kgf/mm2',
      poisson real [],
      spc_gravity real [];
where
data(@name, ^elasticity, ^shearing, ^poisson, ^spc_gravity);
local
      data(steel,1000,8200,0.3,7.9);
      data(iron,21500,3300,0.28,7.9);
      data(alminum,7200,2760,0.34,2.7);
      data(cupper,12500,4700,0.34,8.9);
      data(brass,6300,2400,0.34,8.0);
end.

class beam_model
part
material:= #material;
parameter
    elasticity real 'kgf/mm2',
    inertia real 'mm4',
    uniform_load real 'kgf/mm',
    length real 'mm',
    position_x integer 'mm',
    max_bending_moment real 'kgf·mm',
    max_shearing_force real 'kgf',
    shearing_force_at_x real 'kfg',
    bending_moment_at_x  real 'kgf·mm',
    deflection_at_x real 'mm';
where
      material!elasticity == elasticity;
      uniform_load*length = max_shearing_force;
      -uniform_load*length**2 / 2 = max_bending_moment;
      -uniform_load* position_x = shearing_force_at_x;
      -uniform_load*position_x**2 / 2 = bending_moment_at_x;
      uniform_load*position_x**4 / (24*elasticity*inertia) =deflection_at_x;
end.
```

Fig. 4.16 Cantilever beam model expressed by FDL

4.10.2 FDLによる実行

ここでは前節で作成したFDLプログラム（**図 4.16**）を実際に実行してみる。**図 4.17** にプログラム実行のためのコマンド入力とシステム側の応答を示す。ユーザのコマンド入力はアンダーラインを入れてある。イタリック文字は説明のためのコメント文である。

:display(#1)コマンドで起動された構造化スプレッドシートインタフェイスを**図 4.18** に示す。beam_model と material の両方のウィンドウが作成され、値の入力を受け付け、計算結果を表示する。**図 4.17** において、:display(#1)コマンド以下に表示されている制約起動のメッセージは、構造化スプレッドシートインタフェイスへの値入力によるものである。

```
meljis0[1]% prolog                                    % Prolog の起動
SICStus 2.1 #9: Tue Dec 27 16:26:03 JST 1994

| ?- [fdl2conf].                                      % FDL システムのロード
{consulting /export2/home/imamura/FDL-II/fdl2conf.pl...}
{consulting /export2/home/imamura/FDL-II/fdl2cmp.pl...}
                :
                :
** Welcome to FDL-II design environment **

FDL| ?-['katamoti.fdl'].                              %片持ちばりモデルのロード
Loading katamoti.fdl..
Translating class material                            % class meterial を作成
Translating class beam_model                          % class beam_model を作成

FDL| ?-:create(#beam_model,B).                % beam_model のインスタンスを生成するコマンド

material#2                                    %部品となる materail インスタンスも同時に自動生成
parameter
  name                      str#3    @nil
  elasticity              real#4    ^nil  [kgf/mm2]
  shearing                 real#5    ^nil  [kgf/mm2]
  poisson                  real#6    ^nil
  spc_gravity              real#7    ^nil
end

beam_model#1
part
  material   material#2
parameter
  elasticity              real#4    ^nil  [kgf/mm2]
  inertia                  real#9    @nil  [mm4]
  uniform_load             real#10   @nil  [kgf/mm]
  length                   real#11   @nil  [mm]
  position_x               int#12    @nil  [mm]
  max_bending_moment         real#13   ^nil  [kgf.mm]
  max_shearing_force       real#14   ^nil  [kgf]
  shearing_force_at_x      real#15   ^nil  [kgf]
  bending_moment_at_x        real#16   ^nil  [kgf.mm]
  deflection_at_x          real#17   ^nil  [mm]
end
```

Fig.4.17（1）Command inputs and system messages

```
:create(#beam_model,beam_model#1)                    % beam_model インスタンスの生成完了
yes

FDL| ?-:display(#1).                                 %構造化スプレッドシートインタフェイスの立ち上げ

Constraint 1 of material#2, Input slot: name invoked..   % material#2 の制約 1 を起動
data(@name,^elasticity,^shearing,^poisson,^spc_gravity);  %起動された制約

Constraint 1 of beam_model#1, Input slot: length invoked..  % beam_model#1 の制約 1 を起動(length 入力
uniform_load*length=max_shearing_force;                     %                    による)
1.0*200=200.0;                                              %制約の計算結果

Constraint 2 of beam_model#1, Input slot: length invoked..   % beam_model#1 の制約 2 を起動 (制約 1
-1*uniform_load*length**2/2=max_bending_moment;              %                からの伝播による)
-1*1.0*200**2/2= -20000.0;                                   %制約の計算結果

Constraint 3 of beam_model#1, Input slot: position_x invoked..  % beam_model#1 の制約 3 を起動(position_x
-1*uniform_load*position_x=shearing_force_at_x;                 %                    入力による)
-1*1.0*100= -100.0;                                            %制約の計算結果

Constraint 4 of beam_model#1, Input slot: position_x invoked..  % beam_model#1 の制約 4 を起動 (position_x
-1*uniform_load*position_x**2/2=bending_moment_at_x;            %                    入力による)
-1*1.0*100**2/2= -5000.0;                                      %制約の計算結果

Constraint 5 of beam_model#1, Input slot: position_x invoked..   % beam_model#1 の制約 4 を起動
uniform_load*position_x**4/(24*elasticity*inertia)=deflection_at_x;  %    (position_x 入力による)
1.0*100**4/(24*21000.0*500.0)=0.3968253968253968;                %制約の計算結果


beam_model#1
part
  material  material#2
parameter   .
  elasticity                real#4   ^21000.0  [kgf/mm2]
  inertia                   real#9   @500.0  [mm4]
  uniform_load              real#10  @1.0  [kgf/mm]
  length                    real#11  @200.0  [mm]
  position_x                int#12   @100  [mm]
  max_bending_moment        real#13  ^-20000.0  [kgf.mm]
  max_shearing_force        real#14  ^200.0  [kgf]
  shearing_force_at_x       real#15  ^-100.0  [kgf]
  bending_moment_at_x       real#16  ^-5000.0  [kgf.mm]
  deflection_at_x           real#17  ^0.3968253968253968  [mm]
end

:display(#1)
yes

FDL| ?-
```

Fig.4.17 (2) Command Inputs and System Messages

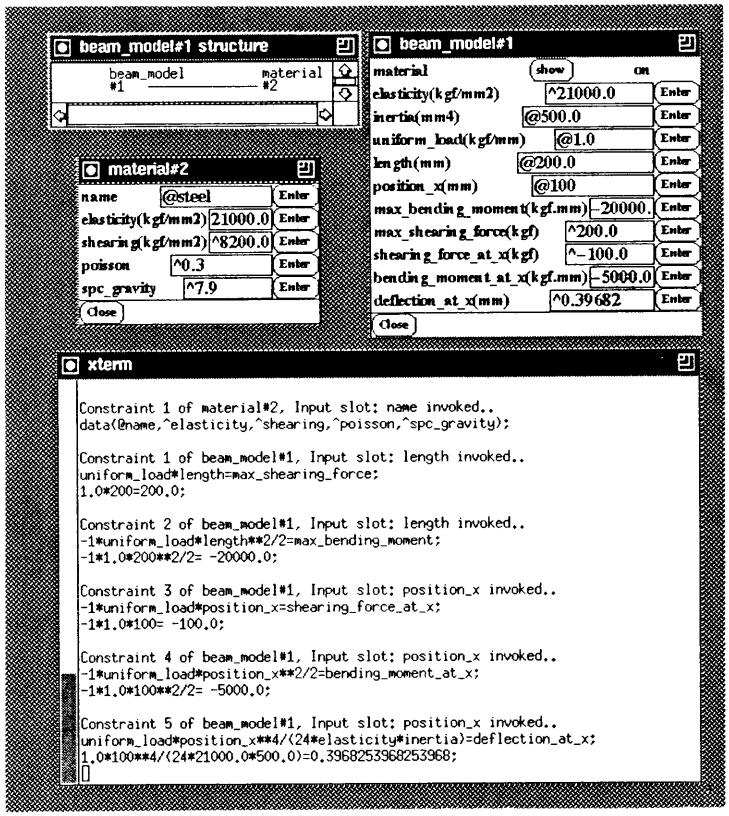


Fig. 4.18 FDL user interface created for the cantilever beam model analysis

４．１１ FDLの問題点

本章で詳述した FDL-II の基本機能はパラメトリック設計支援のために開発されたもので、FDL-I において、ほぼ実現されていたものである。この基本機能により、かなり複雑なパラメトリック設計も可能になったが、一方次のような問題点も明らかになった。

(1) モデルが固い

旋盤設計（第６章参照）のような定型的設計でも、使用によりモデルの構造を変更する必要が生ずるが、あらかじめ予測することは困難で、設計の行き詰まりにより知ることができる。従って、設計途上で動的にモデルのデータ構造、アルゴリズムが変更可能であることが必要となるが、そのようなことに対応できない。これはパラメトリック設計という手法に共通した問題である。

(2) モデルの結合手続きが複雑

現在のモデルの組立関係は、多数の同一性制約で記述しており、モデルの結合の手続きが複雑である。また、同一性制約の記述の仕方も組み付けの対象となるモデルに依存しており、個別的である。形式化、単純化された組立操作を導入することが望ましい。

これらの問題を解決するためには、設計途中でもモデルのデータ構造や制約関係をを変更するための動的構造変更機能を導入する必要がある。 また、単純化されたモデル組立のためのメタオペレーションも導入する必要がある。これらの機能を次章で導入する。

４．１２　４章のまとめ

本章では、FDL の動的構造変更機能（５章）以外の基本機能について詳述した。**4.1** では FDL によるクラス定義の方法、**4.2** ではオブジェクト指向言語としての特徴、**4.3** では論理型言語としての特徴を述べた。**4.4** は FDL の重要機能である制約解決に関する部分で、単一属性制約、同一性制約、多属性制約、生成検証法などについて詳述した。**4.5** ではモデルのアセンブリ記述、**4.6** では関係データベース機能について述べた。**4.7** は FDL のインタプリタの内容に関するもので、プログラムの構成図も示した。**4.8** ではパラメータ入力のためのスプレッドシート型インタフェイス、**4.9** では FDL による設計の手順を述べ、**4.10** では片持ちばりの応力ひずみ解析を例題に FDL の使用方法を紹介した。最後に、**4.11** で FDL の基本機能による設計支援の限界について述べ、５章への導入とした。

第５章　動的構造変更機能の導入

対象指向言語において、オブジェクトを動的に（言語実行系上で）構造変更することはできないか(C++, Smalltalk など)、クラスオブジェクトのみ可能であった(CLOS, ESP など)。後者の場合でも、オブジェクト間の継承関係を維持するため変更の影響が継承関係を介して他のオブジェクトに伝播するので、オブジェクトを個別に変更することができなかった。

FDL-II では、動的オブジェクトの管理機構である was_a リンク並びに was リンクを導入することにより、オブジェクトを個別に動的構造変更することを可能にした。また設計モデルの構造変更に適した、構造融合、構造交換などのメタ操作(オブジェクトのメカニズムそのものを変更する操作なのでこのように呼ぶ)を整備し、さらに、状況を判断して自ら動的構造変更を行うための配置演算子を導入した。

以下、FDL-II に導入した構造融合、構造交換などのメタ操作、was_a リンク、was リンクによる継承関係管理、配置演算子等について述べる。

５．１　メタ操作の導入

本節では、オブジェクトのデータ構造、アルゴリズムを変更するためのメタ操作を導入する。**図 5.1** に本節で導入されるメタ操作の主なものを示す。(1)は構造融合、(2)は構造交換、(3)は浮遊クラスの生成、(4)は浮遊クラスの一般クラスへの変更、(5)は部品スロットの追加、(6)はパラメータスロットの追加、(7)はスロットの削除、(8),(9)は多属性

(1)　:merge(Object1, Object2)
(2)　:swap(Object1,Object2)
(3)　:create_floating_class(Object,FloatingClass)
(4)　:create_settled_class(FloatingClass,Class)
(5)　:add_part(Object,Name,SlotContents,Position)
(6)　:add_parameter(Object,Name,Type,Condition,Position)
(7)　:delete_slot(Object,Name)
(8)　:add_scons(Object,Parameter,Costraint)
(9)　:renwe_scons(Object,Parameter,ConsList))
(10)　:add_constraint(Object,Constraint)
(11)　:delete_constraint(Object,Position)
(12)　:add_local(Object,Predicate)
(13)　:delete_local(Object,Position)

Object は浮遊クラスまたはインスタンス.
Class は，従来型クラス.

Fig. 5.1 Meta-operations

制約の追加と削除、(10),(11)はローカル述語の追加と削除のメタ操作である。それぞれの詳細については、**5.1.1** 以降に述べる。浮遊クラス（**5.5.2** 参照）はクラス名に前置詞"#"を付けて表現する。クラス名は一般のクラス、浮遊クラスを問わず、FDL インタープリター上でユニークでなければならない。浮遊インスタンス（**5.5.1** 参照）はインスタンス番号に前置詞"#"を付けて表現する。一般型インスタンスは"クラス名#インスタンス番号"の形式でも表現できるが、浮遊インスタンスはいずれのクラスにも属さないのでこのような表現は許していない。

なお、本章のメタ操作の説明に用いられる論理変数"Object"は、インスタンスまたは浮遊クラスを表すことにする。演算子":"はオペレータすべてに付くもので、一般述語と区別するためのものである。

5.1.1 部品スロットの追加

部品スロットの追加は次のいずれかのメタ操作により行う。

:add_part(Object,Slot,Part,Position) (5.1)

:add_part(Object,Slot,Part) (5.2)

(5.1)の操作では、部品スロット Slot を Object に追加する。Part は Slot に格納するオブジェクトを指定するもので、既存のインスタンスが指定された場合はそれをそのまま Slot に格納し、クラスが指定された場合は、そのインスタンスを生成し Slot に格納する。Position は Slot を置く場所を指定するためのもので、最初に置く場合は first、最後に置く場合は last と指定し、それ以外の位置は順番を表す整数で指定する。(5.2)のように Postion は省略可能で、その場合 Position=last と同等である。

5.1.2 パラメータスロットの追加

パラメータスロットの追加は次のいずれかのメタ操作によって行う。

:add_parameter(Object,Slot,Type,Condition,Position) (5.3)

:add_parameter(Object,Slot,Type,Condition) (5.4)

パラメータスロット Slot を Object に追加する。Type はパラメータのタイプ、Condition は単一属性制約、Position は Slot の位置を指定する。(5.4)のように Position は省略可能で、その場合 Position=last と同等である。

5.1.3 スロットの削除

部品スロット及びパラメータスロットの削除は、両者とも次のメタ操作によって行う。

:delete_slot(Object,Slot) (5.5)

Object の部品スロットまたはパラメータスロット Slot を削除する。スロットに格納されているオブジェクトは他のスロットと共有されている場合はそのまま残すが、他のスロットから参照されていない場合は、同時に削除される。これは、partnomy 階層構造におけるトップレベルのオブジェクトを除き、どこからも参照されないオブジェクトは削除する

という考え方に基づく。

図 5.2 に:delete_slot/2 の適用例を示す。:delete_slot(#1,B)を実行すると、オブジェクト#1 のスロット B は削除されるが、その格納物である#4 はオブジェクト#2 のスロット A によって参照されているので、削除されずに残る。さらに:delete_slot(#2,A)を実行すると、オブジェクト#2 のスロット A は消去され、オブジェクト#4 もどこからも参照されなくなるので削除される。

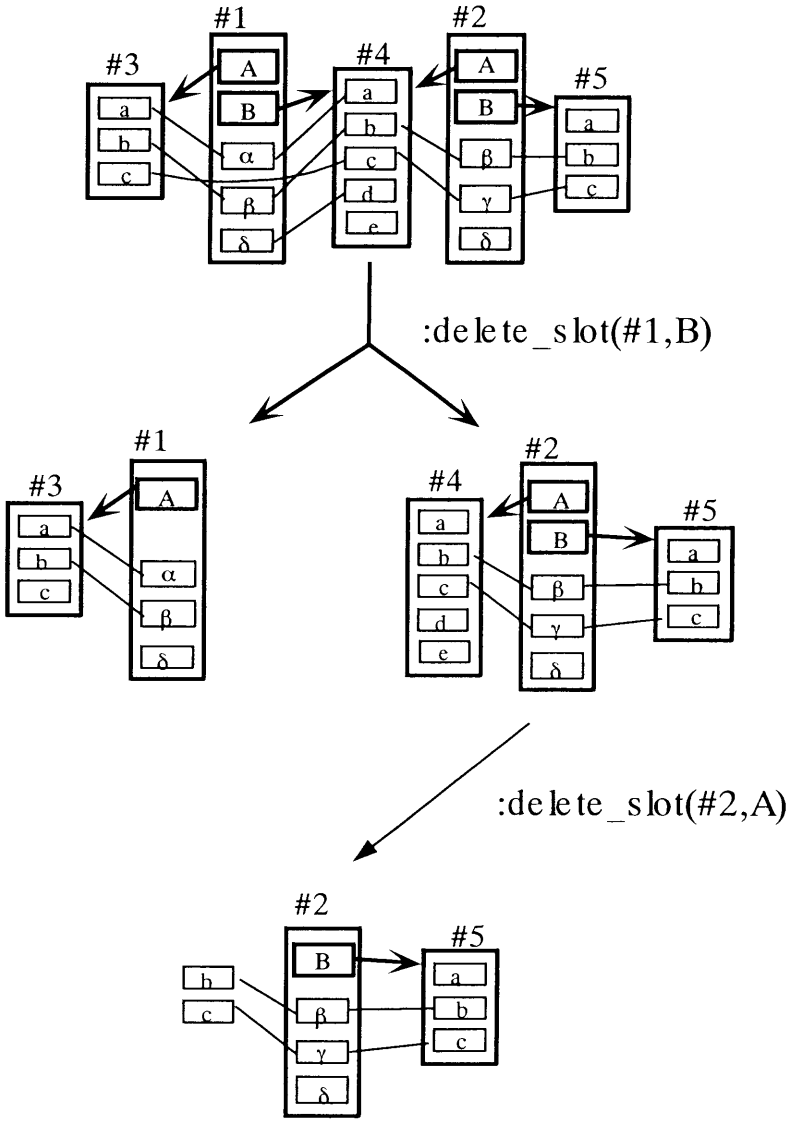


Fig. 5.2 Slot deletion by a meta-operator

5.1.4 単一属性制約の追加／更新

パラメータスロットに記述される単一属性制約の追加及び更新は次のメタ操作によって行う。

:add_scons (Object,Parameter,Constraint) (5.6)
:renew_scons (Object,Parameter,ConsList) (5.7)

(5.6)は Parameter に単一属性制約 Cons を追加するものである。追加する単一属性制約 Constraint がすでに存在する単一属性制約と矛盾する（制約の解がなくなる）場合、この操作は拒否される。(5.7)は parameter の単一属性制約を、ConsList に更新するものである。ConsList は単一属性制約のリストである。ConsLis はそれ自身矛盾しない制約集合であることが要求され、そうでない場合は、この操作は拒否される。

5.1.5 多属性制約の追加／削除

多属性制約の追加及び削除は、次のメタオペレータにより行う。

:add_constraint (Object,Constraint) (5.8)
:delete_constraint (Object, Position) (5.9)

(5.8)は多属性制約 Constraint を Object に追加する。(5.9)は Position の位置にある多属性制約を削除する。多属性制約の追加(5.8)により制約間に矛盾が生じる（過制約状態になる）可能性があるが、制約間矛盾のシステム検出はできないので、多属性制約の追加に伴う無矛盾性の管理はユーザの責任になる。

5.1.6 ローカル述語の追加／削除

ローカル述語の追加及び削除は次のメタ操作により行う。

:add_local (Object, Predicate) (5.10)
:delete_local (Object, Position) (5.11)

(5.10)はローカル述語 Predicate を Object に追加する。(5.11)は Position の位置にあるローカル述語を削除する。

5．2 構造融合操作

構造融合操作は、機械部品モデルのアセンブリ作業を単純化、定型化することを目的に導入された。

SPICE のような電子回路シミュレータでは、部品の接続作業は単純化、定型化されている。それは取り扱う部品の種類（トランジスタ、ダイオード、抵抗、コンデンサ、インダ

クタンスなど）が限られている上、接続作業はキルヒホッフの法則（接続点は等電圧、流入電流の総和０）の適用という定形作業に集約できるからである。一方機械部品のパラメトリックモデルのアセンブリ作業では、機械部品は多種多様であり、また接続作業は部品間に新たな制約関係を導入する作業となり、その内容は複雑かつ不定形である。

構造融合操作とは、二つのオブジェクトの再帰的な融合という内容を持つ操作である。この操作により、設計モデルのアセンブリ作業を単純化、定型化させることができる。

Objec1 と Object2 の構造融合は、次のメタオペレータにより実行する。

:merge(Obejct1, Object2) (5.12)

5.2.1 構造融合の定義

構造融合(structure merging)は、二つのインスタンス間のデータ構造並びに制約記述の和集合に相当するインスタンスを生成する操作である。構造融合は次のように定義される。

オブジェクトAとBが構造融合し、Cが生成される場合、

$$C = A + B \tag{5.13}$$

と記述することにする。=は等価の意味である。このとき、C のパラメータ・スロット、部品スロット、制約の集合はそれぞれ次のように定義される。

$$C.parameter\_slot = A.parameter\_slot \cup B.parameter\_slot \tag{5.14}$$

$$C.part\_slot = A.part\_slot \cup B.part\_slot \tag{5.15}$$

$$C.constraint = A.constraint \cup B.constraint \tag{5.16}$$

但し、X.Y はオブジェクト X の Y 集合を意味し、∪は和演算を意味する。また、スロットの和演算において、A と B に同一スロット a がある場合 C のスロット a の値は、A のスロット a の値を引き継ぐことにする。これは A のスロット a に繋げられているオブジェクト A!a を残し、B!a を捨てるとことを意味する。また、C の所有者集合（structure sharing のため、所有者が２つ以上存在し得る）は次のように定義される。

$$owner(C) = owner(A) \cup owner(B) \tag{5.17}$$

ここで、owner(X)はオブジェクト X の所有者（has_a と逆向きのリンクで表現される）集合である。

オブジェクトCのスロットXへの格納物及びその所有者集合は次のようになる。

$$C!X = A!X + B!X \tag{5.18}$$

$$owner(C!X) = owner(A!X) \cup owner(B!X) - \{A,B\} \tag{5.19}$$

ここで、X!Y はオブジェクト X の持つスロット Y に格納されているオブジェクトを表す。(5.18)式は、構造融合により作られたオブジェクトの構成要素も構造融合により作られていることを意味し、構造融合が再帰的な操作であることがわかる。すなわち、オブジェクト C のスロットに構造を持つオブジェクト X が格納されている場合、そのオブジェクトに対して(5.14)~(5.19)式のアルゴリズムが再帰的に適用される。すなわち、構造融合操作は対象となるオブジェクトがもつ partnomy 階層のすべての階層で上から下に向かって再帰的に適用される。(5.19)式はオブジェクト間の has_a 関係の変化を表現している。最後の項で、C!X の所有者集合から A、B が除かれているのは、これらのオブジェクトが構造融合により消滅するからである。

A!X、B!X がパラメータオブジェクトの場合、次の二つの条件を満たす場合構造融合が行われる。

(1) A!X、B!X が同じ type、同じ単位を持つ。
(2) A!X、B!X それぞれの単一属性制約の和集合が C!X の単一属性制約となるが、これが解を持つこと。

例えば、A!X、B!X の単一属性制約がそれぞれ[>3]と[<7]の場合、制約記述の和集合[>3,<7]（制約を満たす実数空間としては積集合）は解を持つので構造融合できるが、[<7]と[>9]の場合、和集合は解を持たないので、構造融合操作は拒否される。(5.17)式において A!X と B!X の値が異なる場合は、A!X の値が nil（値なし）の場合を除いて A!X の値が C!X の値として採用される。C!X の値が A!X または B!X と異なる場合、構造融合を契機として制約伝播が自動的に起こり、制約解の整合性を回復する。

5.2.2 構造融合操作の意義

構造融合操作は、煩雑かつ不定形な設計モデルの組立作業を単純化、定型化する上で大変有効である。例えば**図 5.3** に示すように、２段変速旋盤の設計モデルを３段変速旋盤の設計モデルに変更するためには、二つの平歯車を２段変速旋盤に組み付け（組み立て）る必要がある。この組み付け作業とは、従来手法では２つの平歯車と２段変速旋盤の２本のシャフトとの間に次のような関係式を記述することに相当する。

(1) 駆動軸、駆動歯車それぞれの最大回転数、回転動力、最大トルク、直径（外径と内径）に等値関係を設定する。
(2) 従動軸、従動歯車間に上記と同様の等値関係を設定する。
(3) 駆動歯車、従動歯車それぞれのモジュール、歯幅、周速間に等値関係を設定する。
(4) 駆動歯車と従動歯車の直径の和、及び、軸間距離の２倍の値を等しくする。
(5) 歯車による減速比（歯数の比）と駆動側、従動側の最大回転数及び最大トルクの間の関係を記述する。

このような組み付け作業の内容は対象により変化するので、組み付け作業は複雑かつ不定

形な作業といえる。一方、構造融合を用いた組み付け作業は、**図 5.4** に示すように、歯車ペアユニットと２段変速旋盤の駆動軸、従動軸、軸間距離同士を構造融合させるのみで完了する。なぜならば、歯車ペアユニット内には上記の関係式がモデル内の関係として予め記述してあり、歯車ペアユニットと２段変速旋盤の間で上記の３つの構造融合操作を行うと、関係式が自動的に歯車ペアユニットの歯車と２段変速旋盤の従動軸、駆動軸、軸間距離の関係に変化するからである。

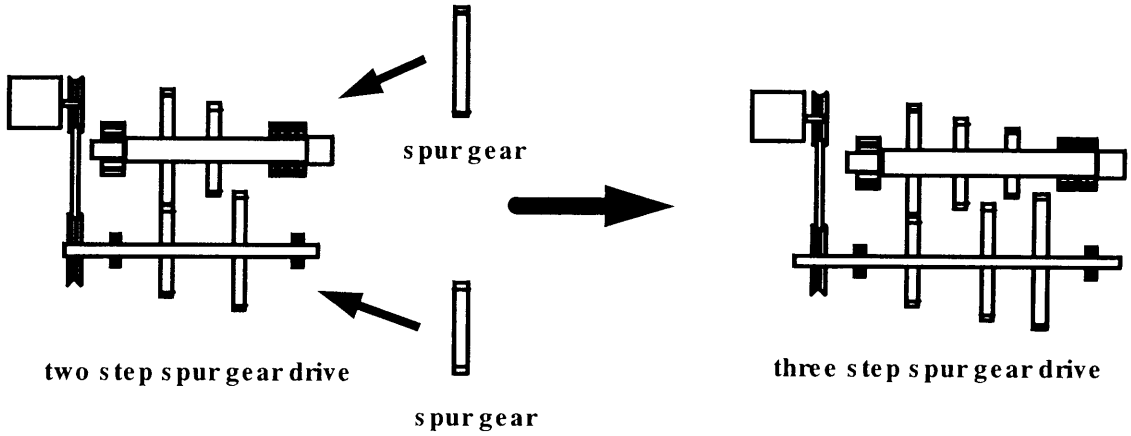


Fig. 5.3 Assembly of spur gears

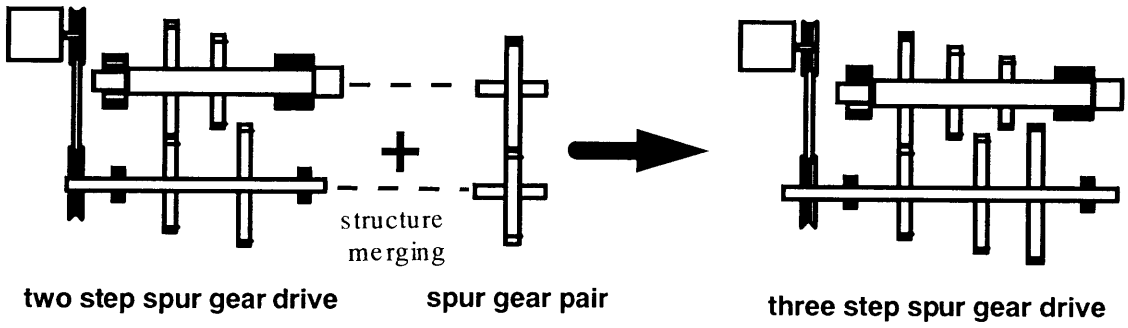


Fig. 5.4 Assembly of spur gears by structure merging

図 5.5 に２段変速旋盤に歯車ペアを組み付ける前のデータ構造を、**図 5.6** に構造融合操作により組み付けを行った後のデータ構造を示す。**図 5.5** における shaft#2 と shaft#4、shaft#3 と shaft#5 同士を構造融合した結果が**図 5.6** である。いずれの図においても、ハッチングしてある部分が歯車ペアのデータに相当する。**図 5.6** において、ハッチング部の内外を結ぶ等値関係が、構造融合によって自動的に設定された同一性制約に相当する。

構造融合操作はプログラム作成時、プログラム実行時、いずれの場合も適用可能である。プログラム内では==オペレータを使って同一性制約を指定すると、その対象となる二つのオブジェクト間に構造融合がインスタンス生成時に行われる。プログラム実行中は、メタオペレータ:merge(Object1,Object2)により構造融合操作を実行できる。

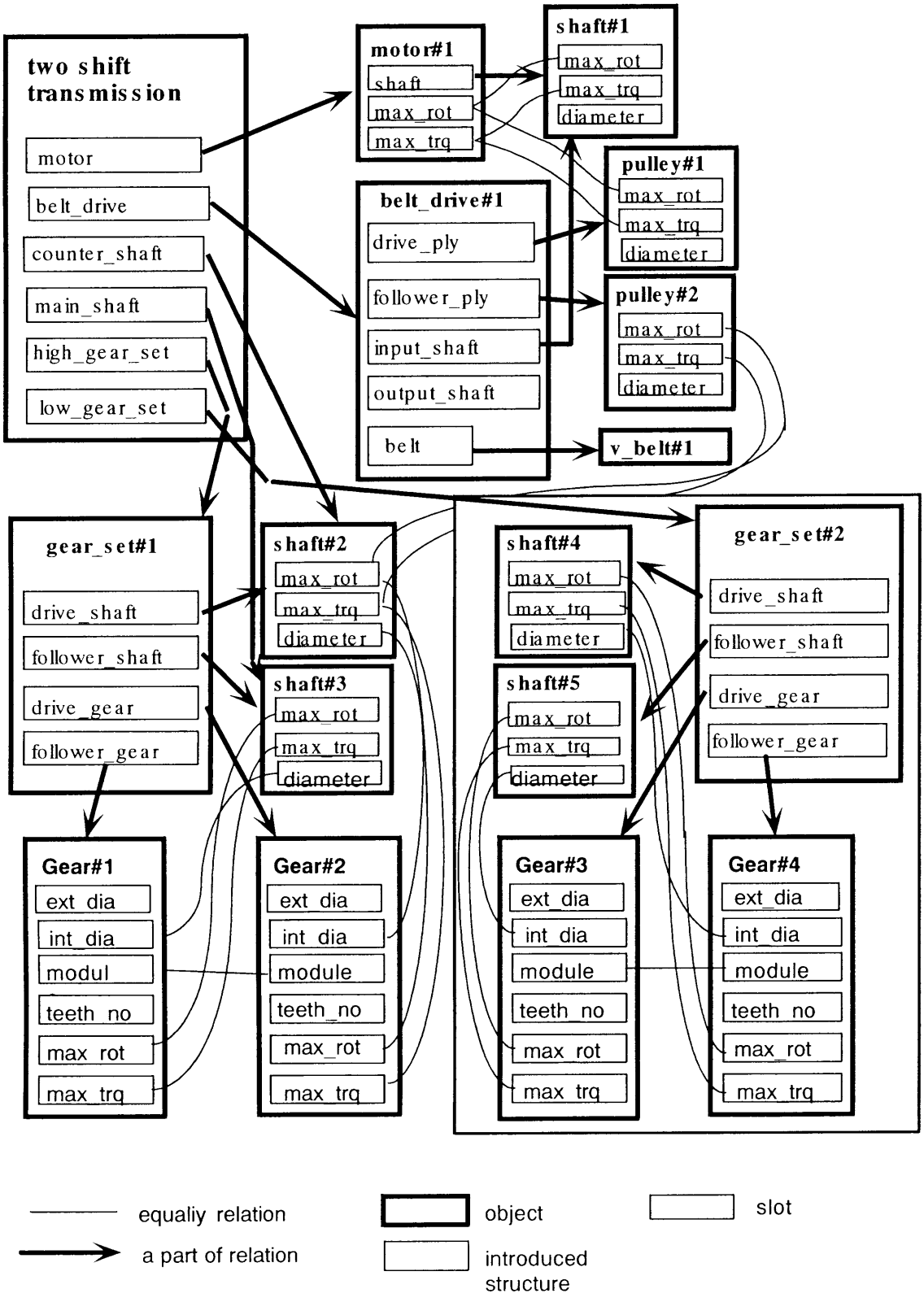


Fig. 5.5 Models to be assembled by structure merging

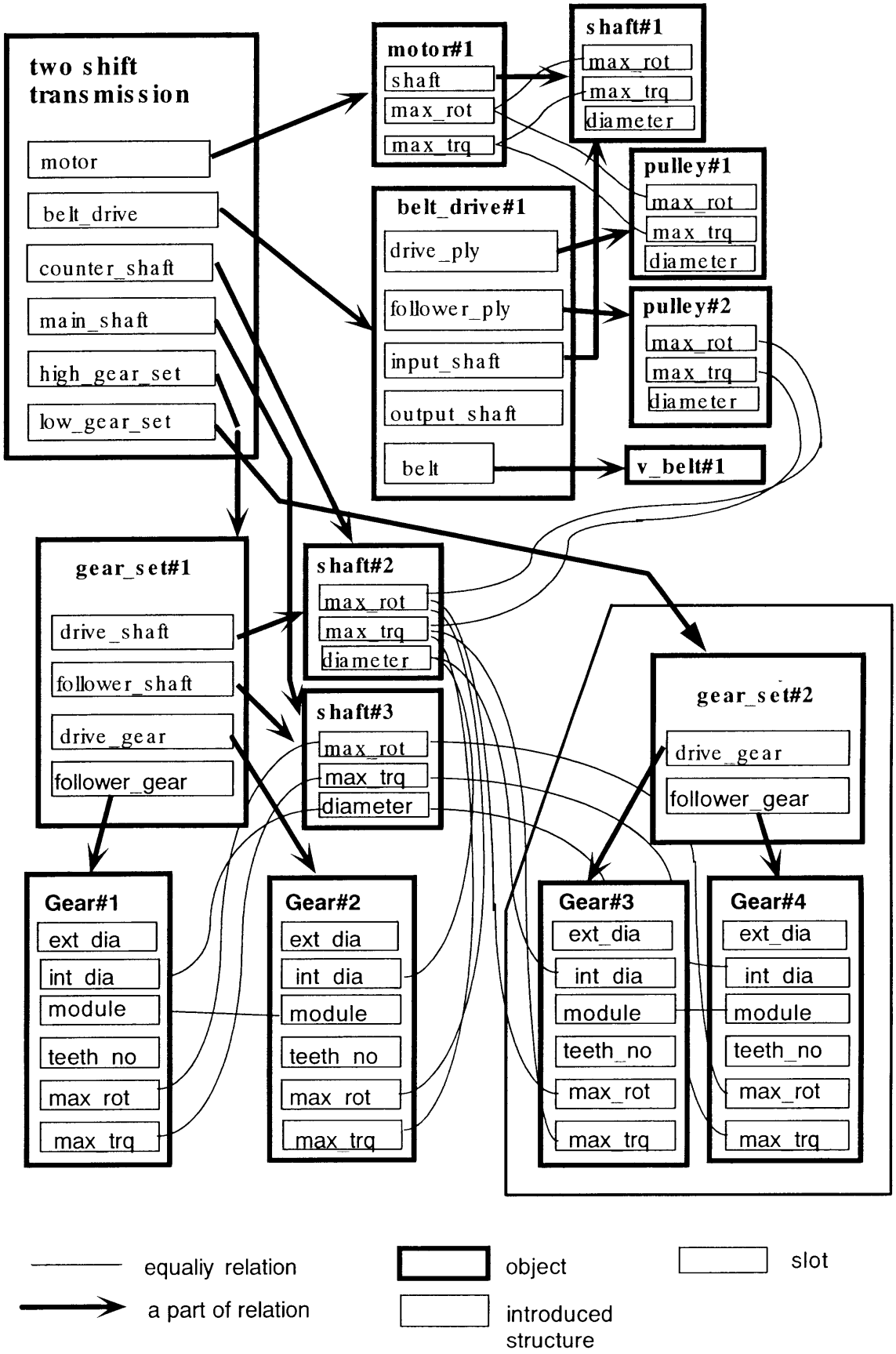


Fig. 5.6 Models assembled by structure merging

５．３　構造交換操作

構造交換（strucutre　swapping）操作は:swap(Inst1,Inst2)により実行され、一方のインスタンス(Inst1)を他方のインスタンス(Inst2)に置き換え、それに伴い Inst1 が持つ同一性制約、has_a 関係を Inst2 に引き継ぐ作用を持つ。次のように定義される。

オブジェクトＡがＢと構造交換されてＣが生成される場合、

$$C = A \leftarrow B \tag{5.18}$$

と記述することにする。この場合、

$$C.parameter\_slot = B.parameter\_slot \tag{5.19}$$
$$C.part\_slot = B.part\_slot \tag{5.20}$$
$$C.constraint = B.constraint \tag{5.21}$$

また、

$$^{\forall} X \in A.S \cap B.S \tag{5.22}$$
但し、$S \in \{parameter\_slot, part\_slot\}$

について、.

$$C!X = A!X \leftarrow B!X \tag{5.23}$$
$$owners(C!X) = owners(B!X) - \{B\} + \{A\} \tag{5.24}$$

である。(5.23)式は A!X と B!X についても構造交換が再帰的に適用されることを示している。(5.24)式は A!X と B!X が入れ替わるためには B!X の所有者集合の中で B を A で置き換えたモノが新しい所有者集合になることを示す。また、

$$^{\forall} Y \in B.S \cap \sim A.S \tag{5.25}$$

については(～ X は X の補集合を表す)、

$$C!Y = B!Y \tag{5.26}$$
$$owners(C!Y) = owners(B!Y) \tag{5.27}$$

となる。

構造交換操作は軸受けモデルを異なるタイプのものに交換するなど、設計モデルを構成する部品モデルを別の部品モデルに交換する場合に利用できる。例えば、**図 5.7** に示すよ

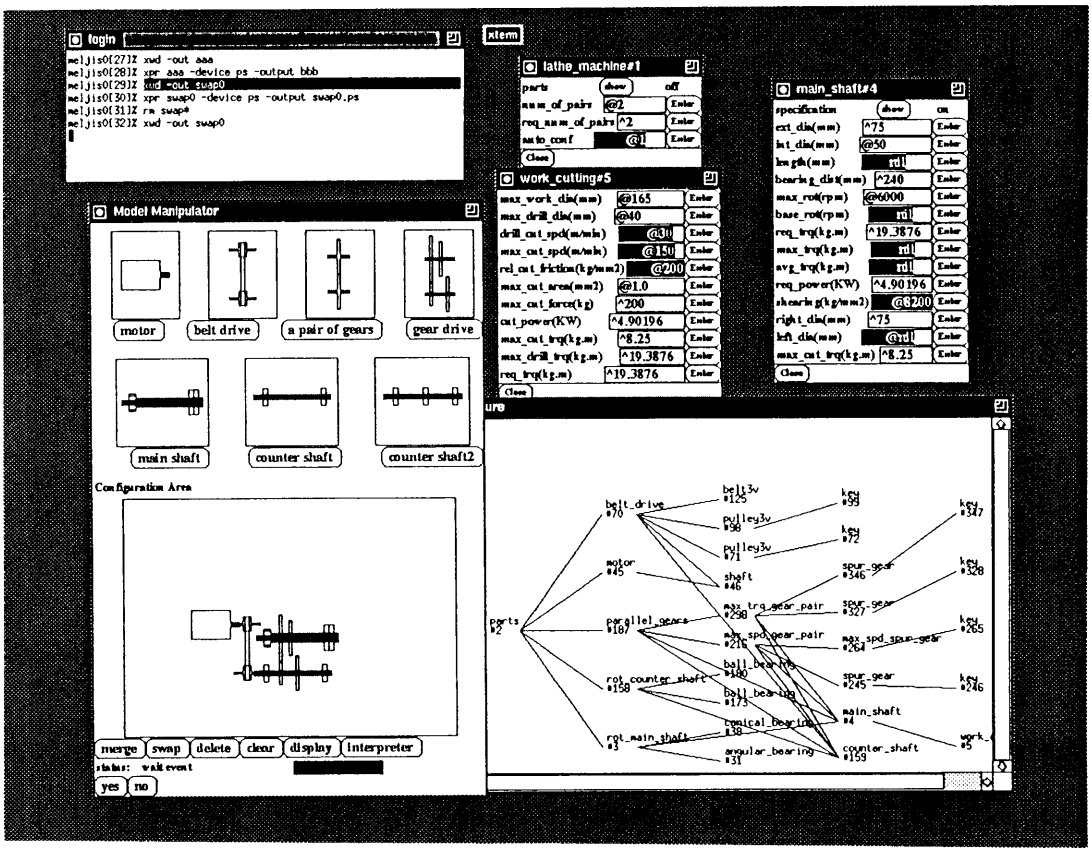


Fig. 5.7 Two shift gear transmission

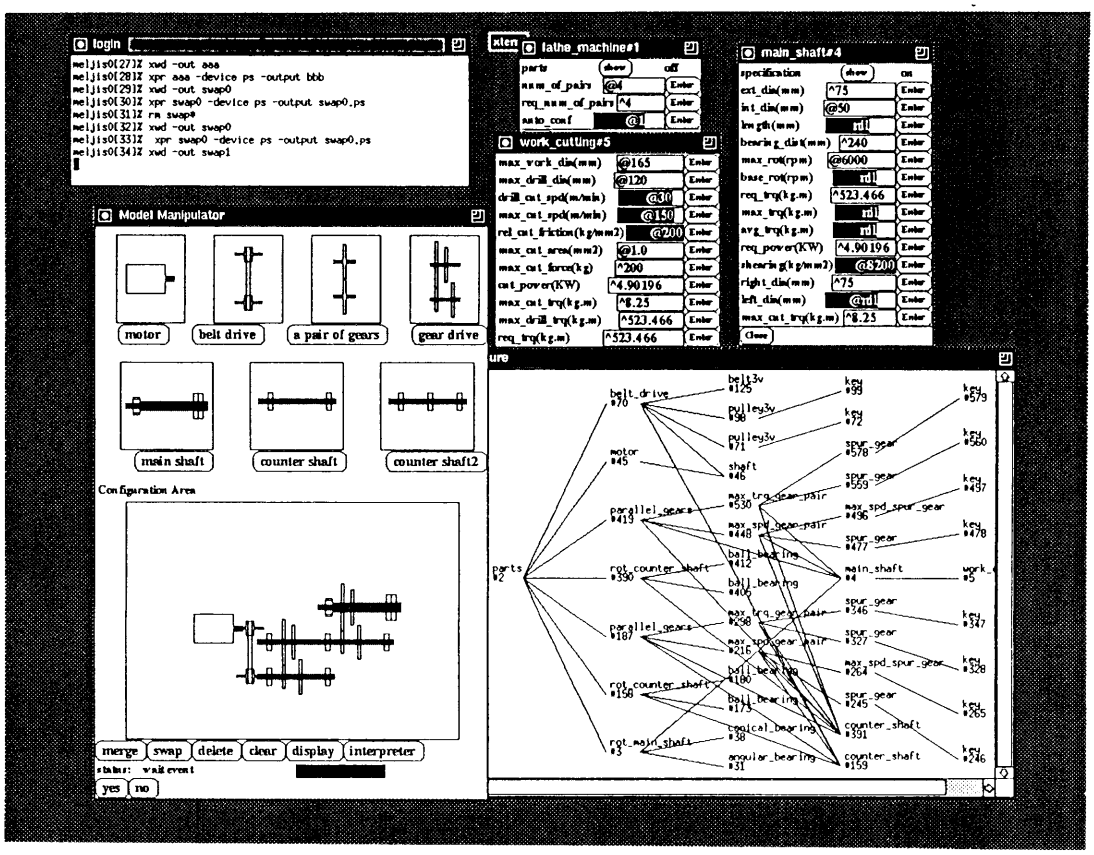


Fig. 5.8 Four shift gear transmission

うな２段変速の歯車伝達機構を、**図 5.8** に示す４段変速の歯車伝達機構に構造変更する場合、主軸と第２中間軸（**図 5.8** で軸受けが３つあるもの）の構造交換が必要となる。この構造変更操作では、次の操作列が実行される。

```
:add_part(parts,rot_counter_shaft_2,#rot_counter_shaft,4)
:add_part(parts,parallel_gears_2,#parallel_gears,5)
:swap(parts!rot_main_shaft,parts!rot_counter_shaft_2)
:merge(parts!rot_counter_shaft_2!shaft,parts!parallel_gears_2!drv_shaft)
:merge(parts!rot_main_shaft!shaft,parts!parallel_gears_2!flw_shaft)          (5.28)
```

５．４　アセンブル操作

部品の組み付け操作はスロットの追加と一連の構造融合操作により行われるが、これらの作業を一回の操作で行えるようにしたのがアセンブル操作である。次のオペレータを呼び出すことにより実行される。

```
:assemble(Object,Slot,#Class)                                                (5.29)
```

これは Object に部品スロット Slot を作成し、そこに Class から生成したインスタンスを格納する。その上で、Object と生成したインスタンスの部品スロット及びパラメータスロットで同じ名称を持つものすべてについて、格納物同士を構造融合するものである。

アセンブル操作の適用例として、２段変速の歯車伝達機構に歯車ペアをアセンブルして３段変速に変更した例を**図 5.9**、**図 5.10** に示す。この作業を行うには、アセンブル操作を使わないと次のオペレータ列が必要であった。

```
:add_part(Object!parallel_gears,gear_pair2,#gear_pair),
:merge(Object!parallel_gears!drv_shaft,gear_pair2!drv_shaft),
:merge(Object!parallel_gears!flw_shaft,gear_pair2!flw_shaft),
:merge(Object!parallel_gears!shaft_dist,gear_pair2!shaft_dist);              (5.30)
```

これは、スロット gear_pair2 を生成し、そこにクラス gear_pair のインスタンスを格納し、parallel_gears と gear_pair2 の駆動軸(drv_shaft)、従動軸(flw_shaft)、軸間距離(shaft_dist)同士を構造融合するものである。

この一連のオペレーションは次に示すアセンブル操作のみで置き換え可能である。

```
:assemble(Object!parallel_gears,gear_pair2,#gear_pair)                       (5.31)
```

平歯車ペアとカサバ歯車ペアのアセンブルなど、許してはいけない操作があるが、配置演算子（**5.6**）における起動条件と同じ仕組みでチェックすることは可能である。

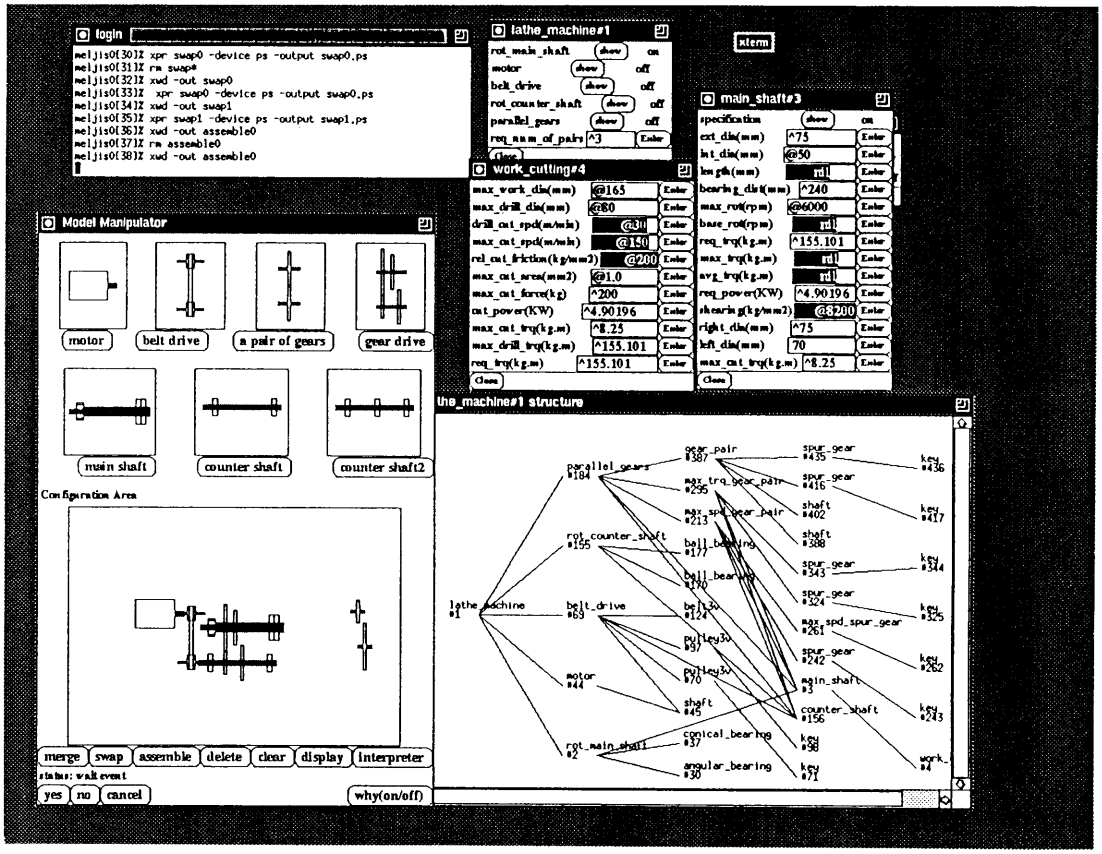


Fig.5.9 Model structuring by assemble operation (1)

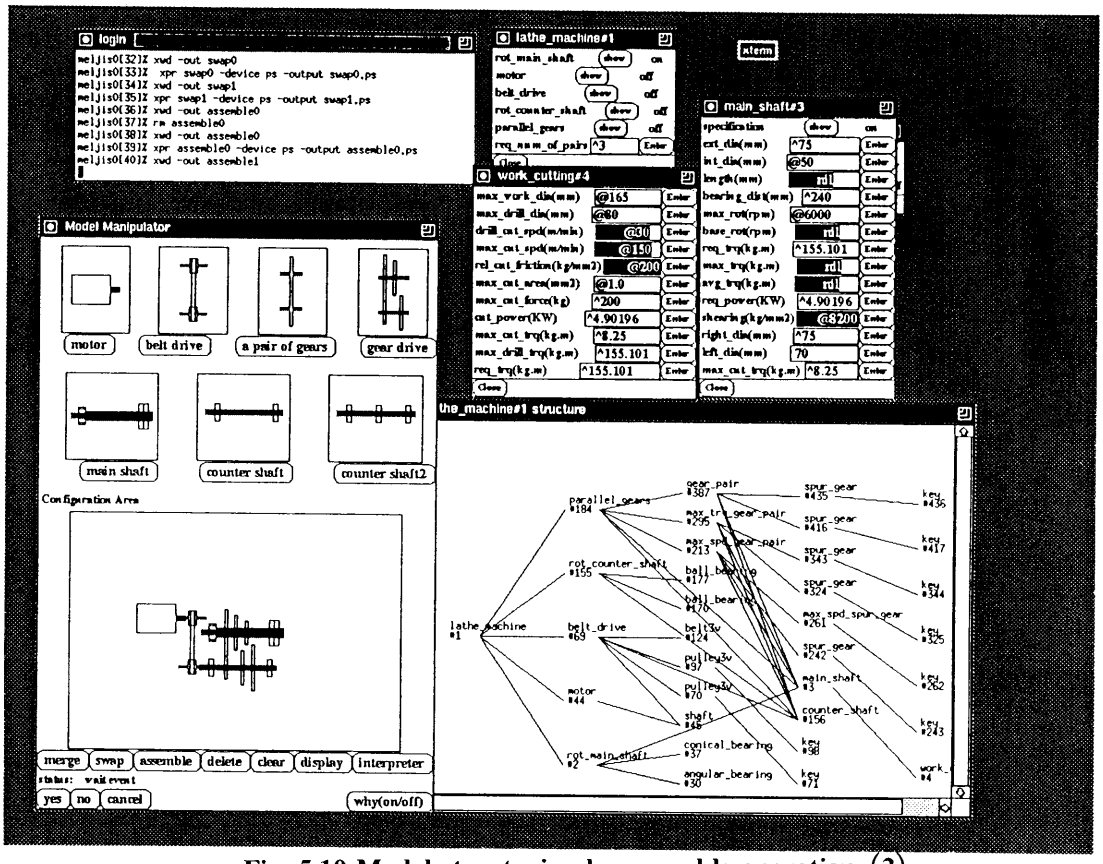


Fig. 5.10 Model structuring by assemble operation (2)

5．5　動的オブジェクトの管理機構

従来、Smalltalk80[Goldberg, 84], C++, Objective C, ESP[Uchida,87], Prolog++[Moss,94]など、ほとんどの対象指向言語は、オブジェクトの動的構造変更（言語実行系上で動作中の言語の構造を変更すること）を許していなかった。CLOS(Common Lisp Object System)[Steele Jr, 90]，Loom，KLIはクラスオブジェクトの構造変更を許しており、is_a関係の整合性を保つために、変更の対象となるクラスを継承するすべてのサブクラス及びインスタンスはその影響を受けて自動的に変更する仕組みをとっている。例えば**図 5.11** のクラス a の method1 を変更すると、クラス a を継承するクラス b、クラス c、インスタンス b#1、インスタンス b#2、インスタンス c#3 の method1 が連動して変更される。一方クラス a を継承するいずれのオブジェクトにおいても、method1 を独立に変更することはできない。なぜならば、親クラスとの継承関係(is_a 関係)の整合性が壊されてしまうからである。この方式は多数のオブジェクトを整合性を保ちながら同時に変更できる点で優れているが、オブジェクトを個別に変更できないので、設計モデルを扱うには不十分である。

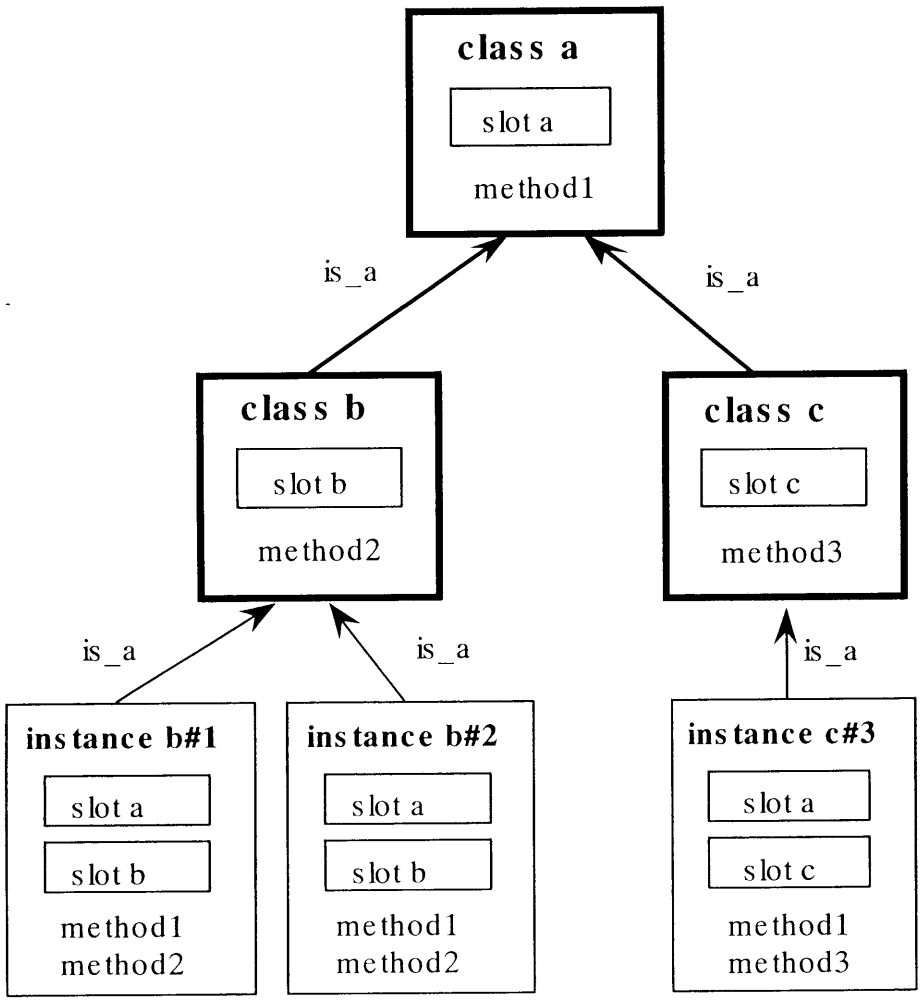


Fig. 5.11　is_a link

設計ではモデルの構造を次々と変更していくことは常にあることであり、しかも変更途中のモデルは多くがアドホックな存在である。従って、設計モデルを表現するオブジェクトは、次のような要求を満たすことが望まれる。

(1) 他のオブジェクトに影響を及ぼすことなく、単独でデータ構造やメソッド(FDL では制約)を動的かつ無制限に変更できる必要がある。
(2) 上記の要求は、クラス、インスタンス双方で満たされる事が望ましい。
(3) 動的変更を受けたオブジェクトは必要に応じてオリジナルの継承階層に組み入れ、アドホックな存在から安定的な存在へ変更することができる。

以上のような要求を満たすために、FDL では新たに was_a リンク及び was リンクを導入した。前者はインスタンス、後者はクラスの個別的動的変更を可能にするためのものである。本節では、これらの新しい概念を紹介する。

5.5.1 was_a リンク

was_a リンクはインスタンスの個別的動的変更を可能にするための、継承関係表現方法である。is_a リンクがクラス間あるいはクラス－インスタンス間の単純な継承関係を表現するのに対し、was_a リンクはクラスと動的変更を受けたインスタンス間の継承関係を表現する。was_a リンクにはインスタンスに対する変更操作の履歴リストが記述されており、単純な継承関係との違いがわかるようになっている。

例えば、**図 5.12** のようにクラス x とそのインスタンス x#1 があると、その間には is_a 関係が存在する。ここで、x#1 はクラス x に所属し、1 番目に生成されたインスタンスであることを意味する。この状態において、メタオペレータ (:add_part/4) によりインスタンス x#1 のデータ構造を変更すると、x と x#1 の間の is_a 関係が成立しなくなる。そこで、インスタンス x#1 は x に所属しなくなるので名称#1 に変更され、さらに was_a リンクを#1 と X の間に張り、そこに#1 に加えられたメタオペレーションのリストを記す。これは#1 が受けた構造変更の履歴を表すためのものである。#1 は自分のクラスを持たず、他のオブジェクトと独立して存在しているので、浮遊インスタンスと呼ぶ。

was_a リンクは、選択的継承を実現する。 FDL-II では、同じクラスを親に持つインスタンス間で共通な情報、すなわち、メソッド、多属性制約、ローカル述語については、インスタンスは is_a リンクを介して、親クラスにある記述を参照、利用する。一方、スロットの内容についてはインスタンスごとに個別の情報を持つので、インスタンスごとに情報を保有し、親クラスを参照しない。浮遊インスタンスもメソッド、多属性制約、ローカル述語に関しては、was_a リンクの構造変更履歴リストの内容に応じて選択的にクラスの記述を参照することができる。すなわち、構造変更履歴リストに delete_method、delete_constraint、delete_local などの情報消去の記述がある場合、該当する情報はマスクし、それら以外の情報を継承する。一方、add_method、add_constraint、add_local などの情報追加記述、add_part、add_parameter、delete_slot などのスロットの追加／削除記述については浮遊インスタンスで局所的に変更処理が行われるので、継承関係に影響しない。

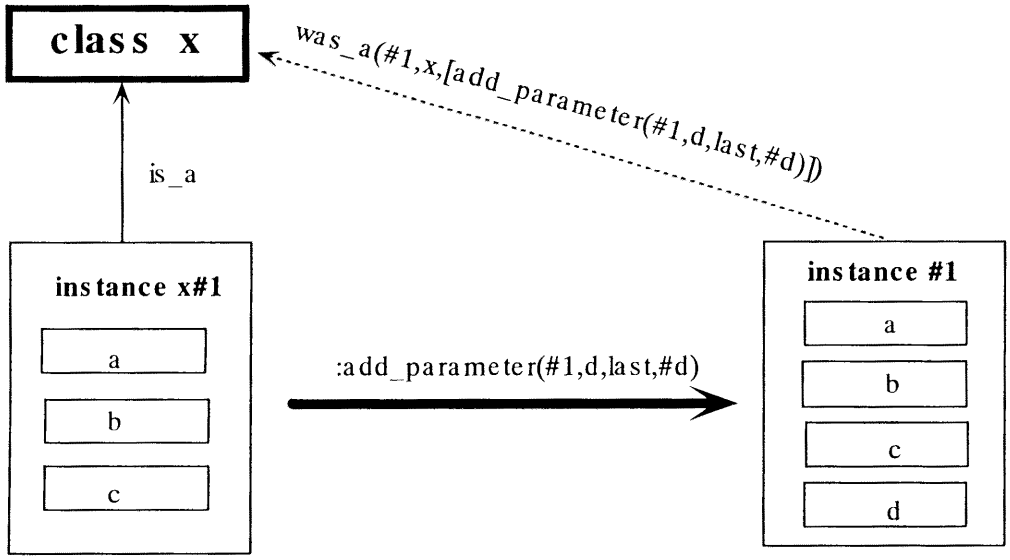


Fig. 5.12 was_a link

5.5.2 was リンク

was_a リンクには、元々所属していたクラスと変更の履歴が記述してあるので、これを利用して浮遊インスタンスのクラスを生成することができる。例えば**図 5.13** に示すように、**図 5.12** で変更操作を受けたインスタンス#1 からメタオペレータ:create_floating_class(#1,B)により、#1 のクラス y を生成することができる。クラス y は最上位クラス class のみを継承し、その他のクラスとは独立して存在するので浮遊クラスと呼ぶ。また浮遊クラス y と#1 の元のクラス x とは was リンクにより関係づけられ、そこには was_a リンクにあった構造変更履歴リストがコピーされる。浮遊インスタンス#1 は y#1 と名称を変え、浮遊クラス y と is_a リンクにより継承関係を持ち、一般のインスタンスと同じ扱いを受けるようになる。

浮遊クラスの概念は、他のクラスへ影響を及ぼすことなく、クラスを個別に変更することを可能にするために導入された。サブクラスへの影響を及ぼすことなくクラスを変更するには、create_floating_class/2 により対象クラスのコピーである浮遊クラスを作り、元のクラスと was リンクを張る。この時点では、was リンクの構造変更履歴リストは空リストである。浮遊クラスからは、一般のクラスと同様任意の数のインスタンスを生成することができる。浮遊クラスとそのインスタンスは is_a リンクによってのみ関係づけられ、浮遊クラスのインスタンスは個別的動的構造変更を許していない。一方、浮遊クラスの動的

浮遊クラスを一般のクラスへ変更する操作（:create_settled_class/2）も用意する。これは、クラス定義を is_a 継承を利用した定義に変更する操作になり、ここでも was リンクの情報が利用される。構造変更は自由であり、その結果は浮遊クラスのインスタンスに伝播する。

5.5.3 米国特許との比較

FDL ではオブジェクト間の継承関係の was_a リンク、was リンクは「オブジェクト指向言語における動的オブジェクトの管理方式」という名称の米国特許 5560024 を取得してい

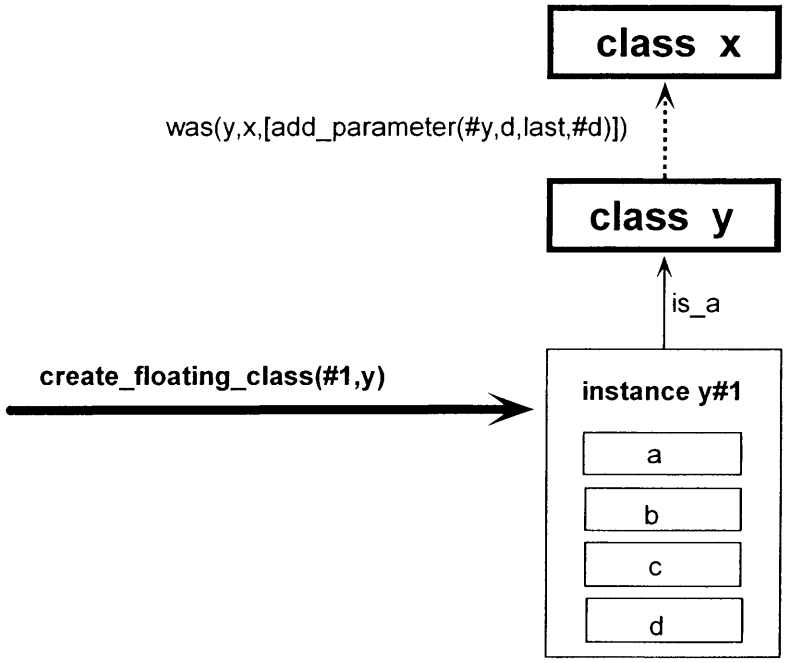


Fig. 5.13 was link

る。その特許申請の際、米国 Patent abd Trademark Office から８件の類似特許が送付され、それらとの違いを説明する必要があった。その際の答弁書は手元に残ってないが、ここでは、それらの類似特許との比較を再度行い、違いを明らかにする。

(1) 米国特許 5418964 "SYSTEM AND METHOD FOR PARENT CLASS SHASOWING IN A STATISCALLY LINKED OBJECT HIERARCHY" (May23,1995)

クラスを動的に変化させることができ、親クラスとの継承関係を動的に管理する。CLOS と類似の方法とみなせる。

FDL ではクラスの動的変更に際しては、親クラスとの関係は was リンクにより参照関係のみを持ち (継承関係はなくす)、必要に応じて継承関係を復活する戦略をとっており、継承関係の変化を動的に管理する上記の方法に比べ、コンピュータへの負担が小さいという利点がある。また、この特許ではインスタンスの動的変更は考慮外である。

(2) 米国特許 5315709 "METHOD AND APPARATUS FOR TRANSFORMING OBJECTS IN DATA MODELS" (May24,1994)

source design object と target design object がそれぞれ異なる空間にあり、異なる方式で記述されて (例えば、関係データモデルとエンティティデータモデル) いても、お互いに関係づけられ、それぞれの変化がお互いに影響しあうことができる。オブジェクト表現の変換を動的に、かつ効率よく行うもの。類似特許といえるかもしれないが、継承関係の管理に関するものではない。

(3)米国特許 5339430 "SYSTEM FOR DYNAMIC RUN-TIME BINDING OF SOFTWARE MODULES IN A COMPUTER SYSTEM" (Aug.16,1994)

電話通信におけるスイッチングソフトウエアの修正、グレードアップはプログラムを止めることなく行う必要がある。この特許では動作を中断することなくプログラムを修正するために、新たにローカルクラスオブジェクトとして導入し、動作中のプログラムとダイナミックなリンクを行うとしている。プログラムの動作を中断すること無くモジュールのバージョン変更を行うことを目的としており、オブジェクト間の継承関係の管理を目的としたものではないので、FDL の特許とは異なる。

(4)米国特許 5339438 "VERSION INDEPENDENCE FOR OBJECT ORIENTED PROGRAMS" (Aug.16,1994)

クラスオブジェクトの変更があっても、クライアントオブジェクト（インスタンス）をの実行バイナリをコンパイルし直すことなく使う仕組みを、オブジェクト間の関係を表現するテーブル操作　を工夫することにより可能にした。C++などを対象にした技術であるが、CLOS などではすでに達成済みの技術と考える。

(5)米国特許 5,386,568 "APPARATUS AND METHOD FOR LINKING SOFTWARE MODULES" (Jan. 31, 1995)

元々独立に開発された任意のオブジェクト間のメッセージ通信を可能にするため、通信仲介用の INPUT、OUTPUT オブジェクトを導入した。本発明は継承関係の問題とは無関係である。

(6)米国特許 5,379,431 "BOOT FRAMEWORK ARCHITECTURE FOR DYNAMIC STAGED INITIAL PROGRAM LOAD" (Jan. 3, 1995)

コンピュータシステムと周辺機器のインタフェイスとなる I/O 処理部分は従来型の言語で作られ、拡張性に乏しいことが問題であった。本発明ではオブジェクト指向言語により I/O 処理プログラムを作り、ＯＳのロード時に割り当てを決める仕組みを持たせ、I/O 処理の拡張を可能にした。 本発明も継承関係の問題とは無関係である。

(7)米国特許 5,418,966 "UPDATING REPLICATED OBJECTS IN A PLURALITY OF MEMORY PARTITIONS" (May 23, 1995)

分散処理システムではデータオブジェクトのコピーを多数分散して持っているが、データオブジェクトを変更する場合は多数のコピーを同時に変更する必要があり、その際コンピュータ処理に重い負荷がかかり処理速度が低下する。この発明では書き込みのロックをコピーオブジェクトに順番に課していくことにより、処理効率を上げることができた。本発明も継承関係の問題とは無関係である。

(8)米国特許 5,261,098 "METHOD AND APPATUS FOR DERIVING OBJECT TYPE AND OBTAINING OBJECT TYPE ATTRIBUTE VALES" (Nov. 9, 1993)

オブジェクトのタイプと属性を得るための方法に関する特許。オブジェクトのタイプと

属性をデータベースに格納し、データベース管理システムによりオブジェクトのタイプと属性を求めることが可能である。本発明も継承関係の問題とは無関係である。

５．６　配置演算子

パラメトリックモデルの組立(configuration)は、構造融合、スロット、制約の追加などのメタオペレータを組み合わせて行うことになるが、依然煩雑な作業である。そこで、あらかじめ想定できる組立構造の変更作業については一連のメタオペレータを組み合わせたマクロ操作を用意しておくと便利である。FDL-II ではメソッドまたは配置演算子(configuration operator)によりマクロ操作を定義する。一連のメタオペレータによる組立構造の変更作業をメソッドとして登録しておくと、ユーザはそのメソッドを呼び出すだけで所望の作業を行うことができる。一方、配置演算子は組立構造の変更作業を自動的に行うためのものである。これは、PROMPT[Murthy & Addanki, 87]における modification operator に類似したものである。しかし、PROMPT がネットワーク化されたモデル群(Graph of Models)から適切なモデルを選択する方法をとっているのに対し、配置演算子はメタオペレータ群を用いてモデルのモジュール構造を動的に変更する点で異なる。

配置演算子は、次の書式で表現する。

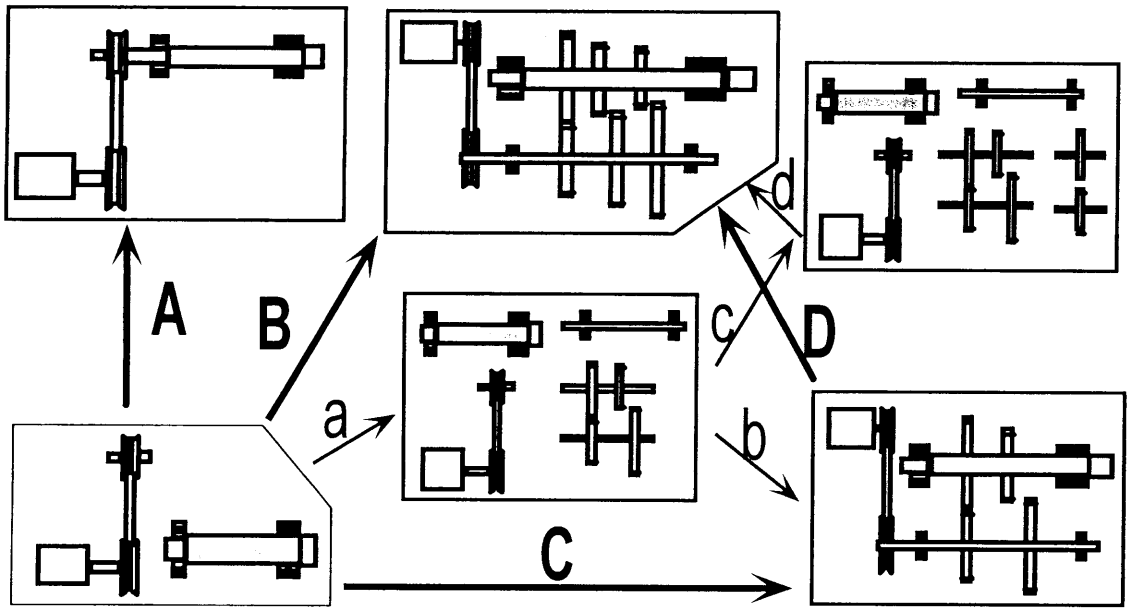
〈起動条件〉 ==> 〈構造変更操作列〉; (5.32)

〈起動条件〉は、配置演算子を起動するための条件を記述する部分で、等式、不等式、述語を用いて表現する。〈構造変更操作列〉には:add_part/4、:merge/2、:swap/2 などのメタオペレータ列が記述される。

この記述形式は活性化条件付き多属性制約のものと同じであるが、配置演算子の起動条件は満たされると一度だけ実行されるのに対し、多属性制約は活性化条件が満たされている限り、何度でも呼び出し可能という違いがある。

配置演算子の記述例を一つ示す。

```
log((@main_shaft!max_rot) * (@main_shaft!req_trq)/
(@motor!flw_shaft!max_trq* @motor!flw_shaft!max_rot))=X,
log((@motor!flw_shaft!max_rot/ @motor!flw_shaft!base_rot))= Y,
floor((X/Y+0.1)) + 1 = Req_num_of_pairs,
Req_num_of_pairs = 2
==>
:add_part(self,rot_counter_shaft,#rot_counter_shaft),
:add_part(self,parallel_gears,#parallel_gears),
:merge(belt_drive!flw_shaft,rot_counter_shaft!counter_shaft),
:merge(rot_counter_shaft!counter_shaft,parallel_gears!drv_shaft),
:merge(main_shaft,parallel_gears!flw_shaft);                    (5.33)
```



A,B,C,D: configuration operations
a,b,c,d: other meta operations (:add_part/4, :merge/2)

Fig. 5.14 Configuration operators

これは旋盤設計の例題からとったものである。Req_num_of_pairs は歯車減速機に必要となる歯車対の数を表し、計算の結果その値が２であった場合、この配置演算子が起動される。<構造変更操作列>部は、中間軸、２組の歯車対を:add_part/3 オペレータにより導入し、それらを:merge/2 オペレータにより組み立てる作業が記述されている。

例にあるように配置演算子内では、配置演算子が記述されているオブジェクトを'self'により参照する。スロット内の格納物は、多属性制約と同等の記法で参照する。

５．７　動的構造変更のためのグラフィックインタフェイス

部分モデルの生成／削除、構造融合、構造交換、アセンブリなどの（メタ）オペレーションを行うためのインタフェイス、動的構造変更用グラフィックエディタを、工作機械を例題に作成した。図 **5.15** に示す。上方にある７つのサブモデルは、設計モデルライブラリに相当し、必要なものを選び、下部にある大きめのウインドウ内に絵を配置すると、対象のオブジェクトが自動的に生成される。下方にある"merge", "swap", "assemble", "delete"などのボタンは、それぞれ構造融合、構造交換、アセンブル、サブモデルの消去を行うためのもので、対象とするサブモデルを下部ウインドウ上で指定することにより、これらの操作を行うことができる。操作の結果、サブモデルが他のモデルに組み付け、ないし、消去されるが、その様子もアニメーションにより表示される。"display"ボタンは、グラフィックエディタで構築中のモデルの構造化スプレッドシートインタフェイスを生成するためのものである。"interpreter"ボタンはプログラムのコントロールを動的構造変更用グラフィックエディタから FDL インタプリタのトップレベルに移すためのものである。FDL イン

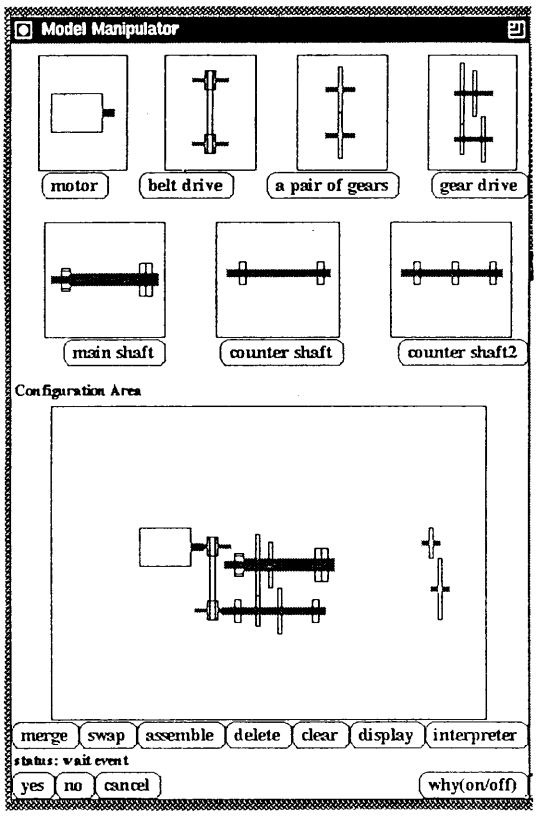


Fig. 5.15 Graphic editor for dynamic structural modification

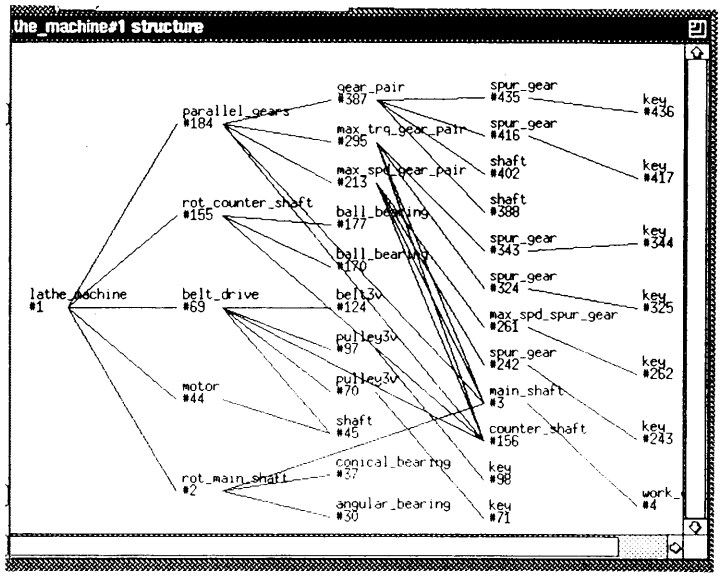


Fig. 5.16 Data structure display window

タプリタから動的構造変更用グラフィックエディタへ制御を返すには、FDL インタプリタ上で、"gmdraw"と入力すればよい。

グラフィックエディタ上で構築中の対象モデルのデータ構造は、**図 5.16** に示すデータ構造表示ウインドウで見ることができる。このデータ構造表示は、モデルの動的構造変更操作に呼応して、最新情報に自動更新する。

５．８　第５章のまとめ

本章では、動的構造変更機能を中心に詳述した。**5.1** ではメタ操作のうち、スロット、制約、ローカル述語の追加削除に関するものを導入した。**5.2** では構造融合操作の定義と意義、使用例を、**5.3** では構造交換操作の定義と使用例を示した。 **5.4** はアセンブル操作の意義と使用例を示した。**5.5** は動的に変更されるオブジェクトの継承関係の無矛盾性の管理に関するもので、was_a リンクと was リンクを導入し、**5.6** では状況を判断して動的構造変更をオブジェクト自ら、自動的に行うための配置演算子を導入した。**5.7** は動的構造変更機能のために新たに導入したグラフィックインタフェイスを示した。

第６章　例題による評価

第６章では、旋盤設計とエレベータ設計を例題にとりあげ、FDL を評価する。旋盤設計は、構造を固定的に取り扱った場合（**6.1**）、構造が成長的に変化する場合（**6.2**）の２例を取り上げる。エレベータ設計（**6.3**）はウエスチングハウス社で実際に業務で使われていた、大規模設計問題である。最後に、国内工作機械メーカに FDL の実験評価を依頼した際の第３者によるレポートを **6.4** に示す。

６．１　旋盤設計（構造固定）

旋盤設計は長い歴史があり、枯れた設計とみなすこともできるが、設計手順、方法などが整理されているわけではなく、過去に蓄積した大量の設計事例、設計データを参考に、経験を積んだ設計者がそれぞれのやり方で設計を行っているのが実情である。

筆者らは、かつて日立精機株式会社に依頼して、旋盤設計の方法をのべ 10 数時間に渡って講義を受け、その結果をレポートにまとめている。後に他の設計者に伺うと、それは通常ノウハウ的で未整理な設計者の設計知識を整理し顕在化したという意味で価値のあることで、教育目的にも有用である。しかし、旋盤設計がこのような手順で進められるとは限らず、設計者はそれぞれが自由な手順で設計を進めているのが実情である。またそこに書かれた設計知識は絶対のものではなく、実際には試作により旋盤の性能が最終的に決定されるということである。

このような背景から、旋盤の設計では自動設計は不適切であり、設計者自身による試行錯誤、評価を受けながら設計支援を行う会話型設計が適当である。FDL には生成検証法による自動設計の能力もあるが、ここでは、構造化スプレッドシートなどのユーザインタフェイスを使って会話形式で設計を進めることとした。

6.1.1　旋盤設計の概要

旋盤設計のうち、モータ動力、軸仕様、歯車減速機構の仕様決定に関する部分について、その概要を紹介する。全般にわたるより詳しい内容については文献[Inoue et al., 88]を参照されたい。

・要求仕様

対象ワーク材質	S48C
工具材質	超硬
最大ワーク径	165mm
最大回転数	6000rpm
最大切り込み深さ	2.0mm
最大送り速度	0.5mm/rev.

最大切り込み面積	1.0mm^2（最大ワーク径の 1/2 の径で）
主軸貫通穴径	50mm
最大ドリル径	40mm
寿命	30,000hours

・設計内容

(1) 最適切削速度

ワーク材質：S48C、工具材料：超硬の条件より 100 ～ 150m/min.とする。

(2) 比切削抵抗　200Kg/mm^{2}とする。

(3) 最大切削力　比切削抵抗×最大切り込み面積＝200Kg/mm^{2}X1mm^2＝200Kg

(4) 最大切削動力　最大切削力×最大切削速度／6120＝200 x 150／6120＝4.9KW

(5) モータ要求動力　切削動力／機械効率（0.75）＝4.9／0.75＝6.5KW

(6) 旋削最大トルク　最大切削力×0.5×最大ワーク径×0.5＝8.25Kg・m

(7) ドリル加工動力

0.6×0.75×10^{-2}×送り速度×ドリル切削速度(30m/min)×ドリル径×2.2= 4.75KW

(8) ドリル回転数　ドリル切削速度／（π・ドリル径）＝ 239rpm

(9) ドリル加工に必要なトルク　974 ×ドリル加工動力／回転数＝ 19.4Kg・m

(10) モータ軸／主軸の回転比

高速側回転比＝　モータ最高回転数／主軸最高回転数＝ 1.0

低速側回転比＝　主軸最大トルク／（モータ 30 分定格最大トルク×機械効率）＝ 5.31

(11) 主軸平均外径をねじり剛性の式から求める

ねじり剛性の式（びびり振動を起こさない条件）

32 ×ねじりモーメント／(π×せん断弾性係数×(主軸平均外径^4－主軸貫通穴径^4))

≦　θ／l ＝ 1/40 deg/m= π／(40 × 180 × 1000) rad/mm

ねじりモーメントは荒旋削最大ﾄﾙｸを用いる。鋼のせん断弾性係数は 8200Kg/mm^2

以上より、主軸平均外径は 73.9mm 以上と計算され、切り上げにより 75mm を採用する。

(12) ベアリングの選択

過去の設計例より、主軸前部（前部とはワーク取りつけ側）に円筒コロ軸受け、主軸後部にアンギュラー玉軸受けのマウントを選択する。他の設計例として、前部にアンギュラー玉軸受け 3 個、後部に円筒コロ軸受けのマウントもある。

(13) 軸受け間距離

Schenk の動剛性計算モデル

530・(主軸平均外径^4－主軸貫通穴径^4) ／軸受け間距離^3　≦　100Kgf／μ m

軸受け間距離＝ 237.9mm と計算されるが、切り上げて 240mm を採用する。

(14) プーリ軸外径

ここでもねじり剛性の式を用いるが、単位長さあたりのねじれ角(θ／l)は 0.3deg/m で十分である。この条件より外径＝ 30.3mm となるが、切り上げて 35mm とする。

(15) 歯車選定

・最大ギアピッチ円径の決定

ギアピッチ円径 ≦　最大ギア周速／（π×最大回転数）

最大ギア周速＝ 2000m/min.　最大回転数＝ 6000rpm より

ギアピッチ円径 ≦ 106mm

・最小ギアピッチ円径の決定

ギアピッチ円径 ≧ 軸径＋キー高さ＋歯車肉厚＋歯の高さ

歯車肉厚＝５・モジュール、　歯の高さ＝２．５・モジュール

Table 6.1　Relations between shaft diameter and key size (JIS B 1301)

キーの呼び寸法 幅 × 高さ		適応する軸径 以上	未満	キーの長さ
5	5	13	20	10 ～ 56
7	7	20	30	14 ～ 90
10	8	30	40	18 ～ 112
12	8	40	50	22.4 ～ 140
15	10	50	60	28 ～ 160
18	12	60	70	35.5 ～ 200
20	13	70	80	45 ～ 224
24	16	80	95	56 ～ 250
28	18	95	110	63 ～ 315
32	20	110	125	80 ～ 355
35	22	125	140	100 ～ 400
38	24	140	160	112 ～ 400

キー高さは表 **6.1** より求める。

軸径＝ 75mm，キー高さ＝ 15mm なので

75 ＋ 15 ＋ 7.5 ×モジュール≦ 106mm

モジュール≦ 2.29

工作機械で用いられるモジュールは 2, 2.5, 3, 4, 5, 8, 12 から選択するという制約があるので、モジュール＝ 2 を選択する。よって最小ギアピッチ円径＝ 104mm となる。各歯車の歯数は次の(6.1)～(6.8)式を満たす値を選択する。

$$52 \leqq Zs1 \leqq 281 \tag{6.1}$$

$$52 \leqq Zs2 \leqq 53 \tag{6.2}$$

$$29 \leqq Zp1 \leqq 53.0y \tag{6.3}$$

$$29 \leqq Zp2 \leqq 53.0y \tag{6.4}$$

$$Zs1 + Zp1 = Zs2 + Zp2 \tag{6.5}$$

$$Zs1 \,/\, Zp1 \geqq \text{低速側回転比} \,/\, y \tag{6.6}$$

$$Zs2 \,/\, Zp2 \leqq 1 \,/\, y \tag{6.7}$$

$$2(Zs1 + Zp1) \leqq \text{主軸軸受け径} + \text{プーリ軸軸受け径} + 10 \tag{6.8}$$

Zs1、Zp1、Zs2、Zp2：歯車歯数（図 **6.1** 参照）

Dp：プーリ軸側プーリピッチ円径

Dm：モータ軸側プーリピッチ円径

y　：プーリ比(Dp/Dm)

(6.1)～(6.4)は最小ギア径及び最大ギア周速の条件、(6.5)は軸間距離一定の条件、(6.6)は主軸低速運転時の主軸とモータの回転比の条件、(6.7)は主軸高速運転時の主軸とモータの回転比の条件、(6.8)は軸受け間のすきまを 10mm 以上とる条件である。

6.1.2 FDL による旋盤設計（構造固定）

ここでは、**6.1.1** に示した旋盤設計を FDL で表現する。旋盤は**図 6.1** に示すような構成をとることとし、構造の変更は考慮しない。モータ、ベルトドライブ、中間軸、主軸、歯車ペアなどから構成されている。

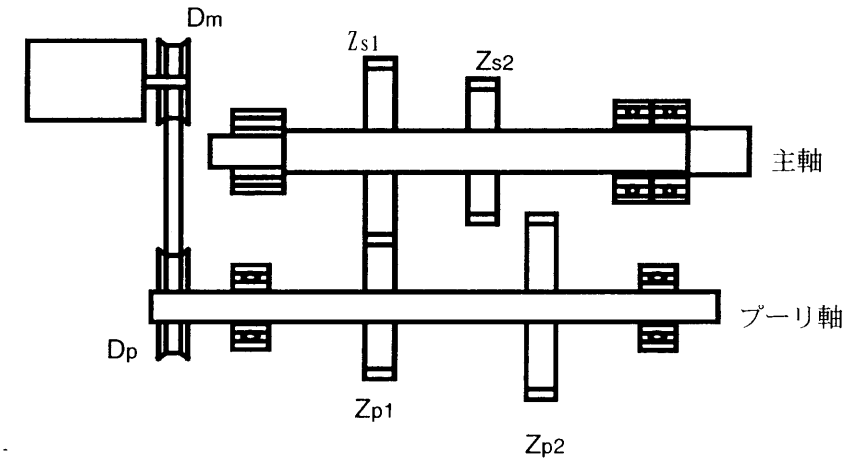


Fig.6.1 Lathe machine structure

図 6.2 に FDL で記述した旋盤モデルのデータ構造を示す。これは、部品 29 個、パラメータ 256 個、同一性制約 101 個、多属性制約 26 個、A型データベース 16 個（主軸材料、中間軸材料、モータ、ｖベルト、プーリ、キーの選択に使用)、B型データベース 4 個（軸受の選択に使用)から構成されている。旋盤モデルの FDL による記述は付録４に示す。**6.1.1** での内容とほぼ同様の内容でモデリングしてあるが、多少異なる部分もある。要求仕様として与える値の数は 12 個、ユーザが決める値の数 2 個で、残りのパラメータ値はすべて制約解決により自動的に決まる。

会話型パラメトリック設計のためのユーザ環境を**図 6.3** に示す。これらのマルチウインドウはモデルの構成を反映した親子関係を持ち、スロットの値は制約起動による連動修正が行われる。このようなことから、構造化スプレッドシートと呼んでいる。

各パラメータの初期値は nil である。まず、最大ワーク径、最大ドリル径、最大ドリル切削速度、最大切削速度、比切削抵抗、最大切込面積、主軸材料、主軸最大回転数、主軸平均回転数、主軸内径、軸受要求寿命、の値を要求仕様として[]付きで入力する。これらは@付スロットであり、システム側が入力を求めていることが明示されている。次に、やはり入力が求められているプーリ比を 1.0 と仮に決めて入力する。すると歯車歯数の存在

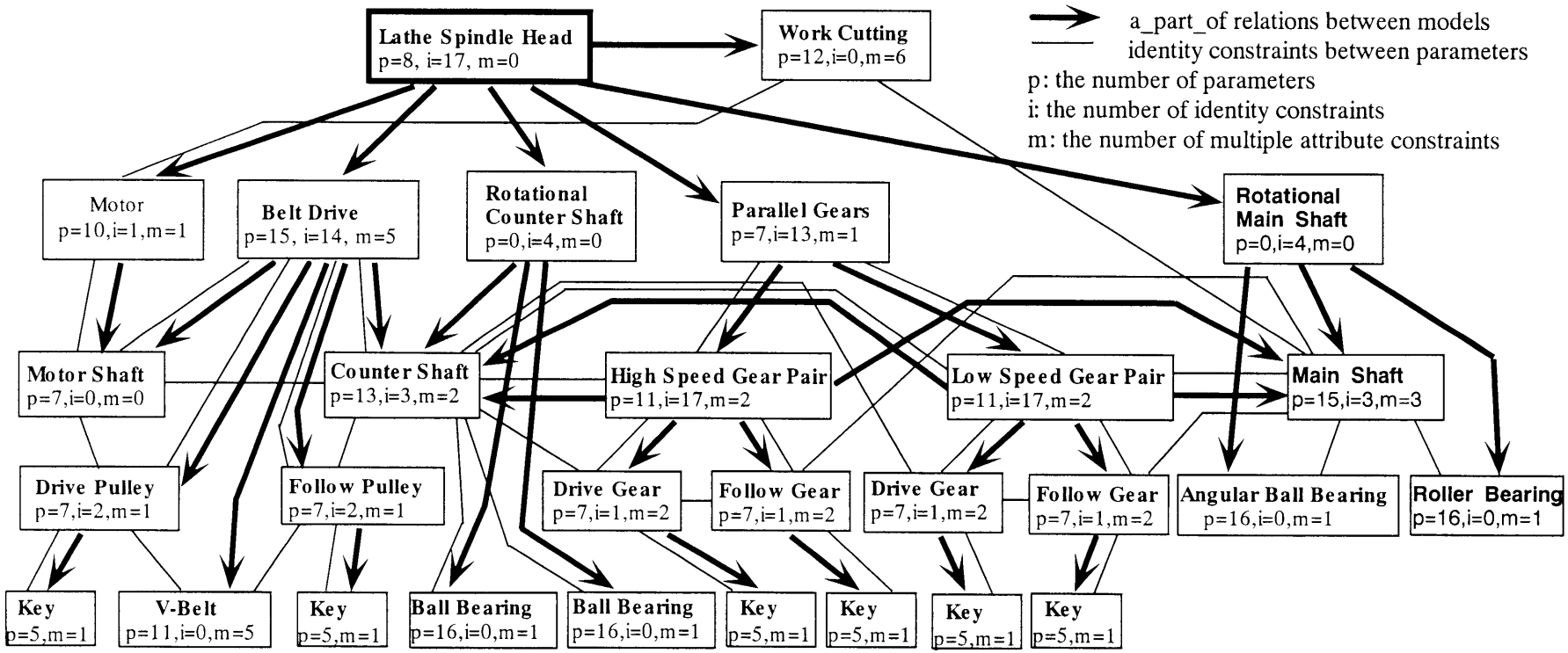


Fig. 6.2 Lathe model data structure

可能範囲が定まるので値生成器が自動的に起動され、生成検証法により連立不等式を満たす値を求めるが、すべて失敗する。そこでプーリ比を 2.0 に変更し、再び生成器を起動すると今度は連立不等式を満たす値が見つかる。さらに、歯車の数をユーザ自身が入力して別解を見つけることもできる。**図 6.3** には、こうして得られた設計結果例の一部が示されている。

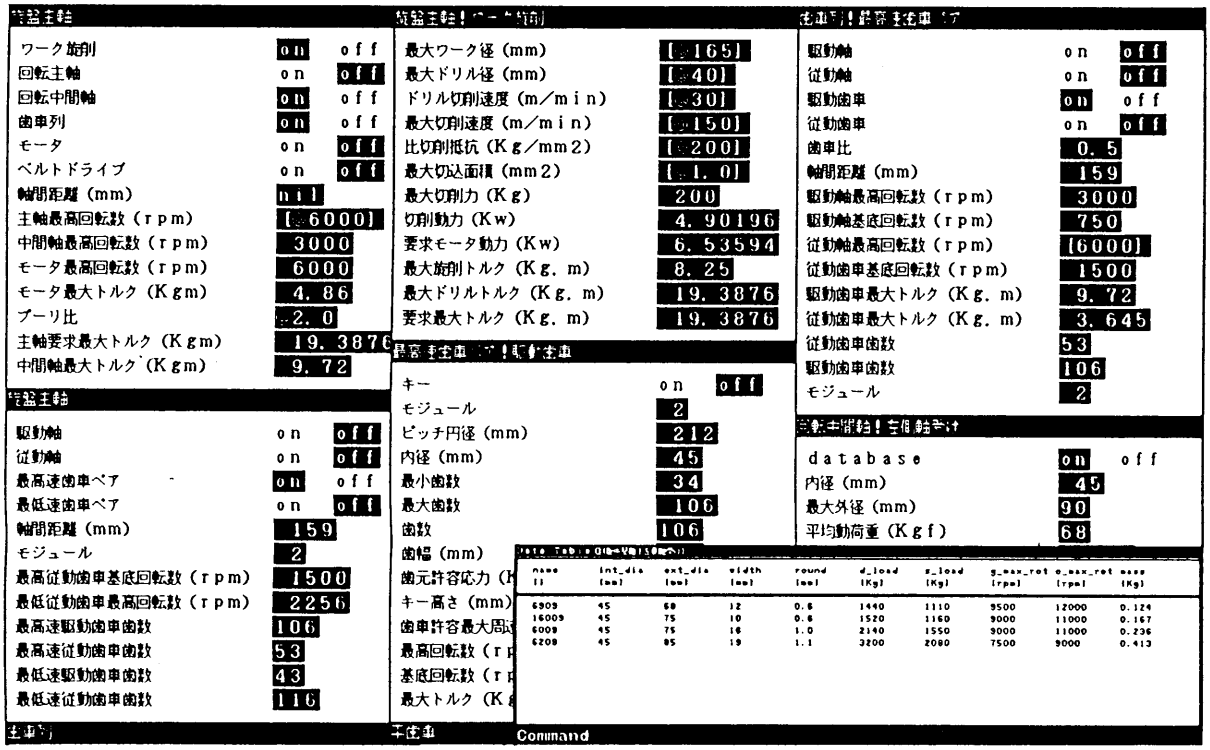


Fig. 6.3 Lathe design by FDL design environment

6. 2　旋盤設計(構造可変)

6.1 では、旋盤の構造を最初から２段変速として固定的に扱ったが、実際は要求仕様により必要となる変速段数が変化し、それに伴い歯車変速機構の構成が大きく変化する。

旋盤に使用する動力モータの特性は図 **6.4** に示すように、最低回転数(40 rpm)と基底回転数(base rotation)の間は最大トルクを出し、基底回転数と最大回転数(max rotation)の間は定格出力を出せるものとする。主軸の出力は最大トルク出力時以外は、出力一定であることが求められる。この条件を満たすには、主軸に要求される最大回転数 Nr、最大トルク Tr、動力モータの基底回転数 Nb、最大トルク Tm、最大回転数 Nm とし、log((Tr･Nr)/(Tm･Nm))/log(Nm/Nb)=A とおくと、旋盤は floor(A+k)+1 の変速段数を持つ必要がある。ここで floor(X)は X の小数点以下を切り捨てる関数、k は安全係数である。主軸トルク、モータの選定も種々の計算を行った結果決まるものであり、変速機構の構成（変速段数）は設計がある程度進まないと決めることができない。

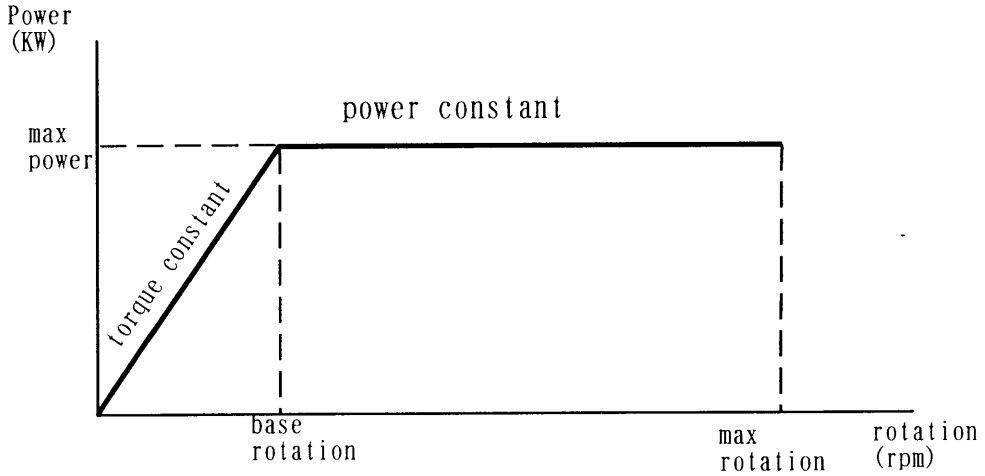


Fig. 6.4　Characteristics of motor power

FDL は、動的構造変更機能を有するので、あらかじめ設計対象の構成が確定していなくても、不完全な設計モデルで設計をスタートさせ、設計の進展につれて必要な構成が判明した段階で、そのような構成をとるモデルを構築することができる。また、要求仕様の変更や設計上の試行錯誤で設計モデルの構成を変更する必要が生じた場合にも、対応が可能である。

FDL では、構造融合、構造交換、部品追加／消去などの演算子を使って、ユーザ自らが設計モデルの動的構造変更を行うこともできるが、あらかじめ配置演算子を用意しておいて自動的に構造変更を行うこともできる。**6.2.1** でユーザ操作による構造変更、**6.2.2** で配置演算子による自動構造変更の実行例を示す。

6.2.1 ユーザ操作による構造構築

旋盤設計は主軸、ベルトドライブ、モータのみから構成される不完全なモデルからスタートする。不完全な旋盤モデルのFDL記述を図6.5に示す。6.1.2（旋盤設計－構造固定）の場合と同様、要求仕様を順次入力していき、主軸及びモータ軸それぞれの最大回転数と最大トルクが決定された段階で、制約式が自動的に起動され、必要な変速段数が決まる。この情報に基づいて必要に応じて歯車ペア等を追加してモデルを構成する。

モデルをマニュアルで構築していく様子を図6.6から図6.9に示す。図6.6は、旋盤の不完全なモデルを対象に要求仕様を順次入力し、必要な歯車ペアの数が２段と決まった（lathe_machine#1 の req_num_of_pairs が２を表示）段階である。ここで、左下のウィンドウ"Model Manipulator"は、モデル構築用のライブラリ、モデルの２次元表示領域、種々の指示を行うためのボタン群から構成されており、ユーザーの指示に従い、組立／分解の様子をグラフィカルに表示する。なお、"why(on/off)"ボタンは、個々のパラメータの値を求めるのに使われた制約式を表示するためのものである。右下のウインドウはモデルのデータ構造を表示するためのもので、モデルが構造変更する度に再描画される。右上のいくつかのウインドウは、構造化スプレッド型インタフェイスを構成する、パラメータ入力／表示用のインタフェイスである。図6.7から図6.9についても同様である。

図6.7はベアリング付き中間軸（rot_counter_shaft）を追加した段階を示している。ここではオペレータ

```
:add_part(#1,rot_counter_shaft)                                    (6.9)
```

を実行している。図6.8はさらに、２段歯車ペア（parallel_gears）を追加した段階で、次のオペレータを実行した。

```
:add_part(#1,parallel_gears)                                       (6.10)
```

図6.9は２段減速構造をもつ旋盤のモデル構築が完成した段階で、ベルトドライブの従動軸(belt_drive!flw_shaft)と中間軸（roc_counter_shaft!counter_shaft）の構造融合、中間軸と２段歯車ペアの駆動軸の構造融合、及びに、２段軸歯車ペアの従動軸と主軸の構造融合の核操作を実行する。これらは次の３つのオペレータの実行による。

```
:merge(#1!belt_drive!flw_shaft, #1!rot_counter_shaft!couter_shaft)
:merge(#1!rot_counter_shaft!couter_shaft, #1!parallel_gears!drv_shaft)
:merge(#1!parallel_gears!flw_shaft, #1!rot_main_shaft!main_shaft)  (6.11)
```

このように、モデルの組立作業を行った後に、さらに詳細なパラメトリック設計を進めていくことになる。この過程は6.1と同様である。

```
class lathe_machine
part
 rot_main_shaft:= #rot_main_shaft,
 motor:= #motor,   belt_drive:= #belt_drive;
parameter
 req_num_of_pairs integer [];
where
 motor!flw_shaft==belt_drive!drv_shaft;
  @rot_main_shaft!shaft!req_power/0.75 = ^motor!req_power;
  log((@rot_main_shaft!shaft!max_rot)  *
       (@rot_main_shaft!shaft!req_trq)/
       (@motor!flw_shaft!max_trq * @motor!flw_shaft!max_rot)) = X,
  log((@motor!flw_shaft!max_rot/ @motor!flw_shaft!base_rot)) = Y,
  floor((X/Y+0.1))  + 1 = ^req_num_of_pairs;
end.
```

Fig.6.5 A lathe description without configuration operators

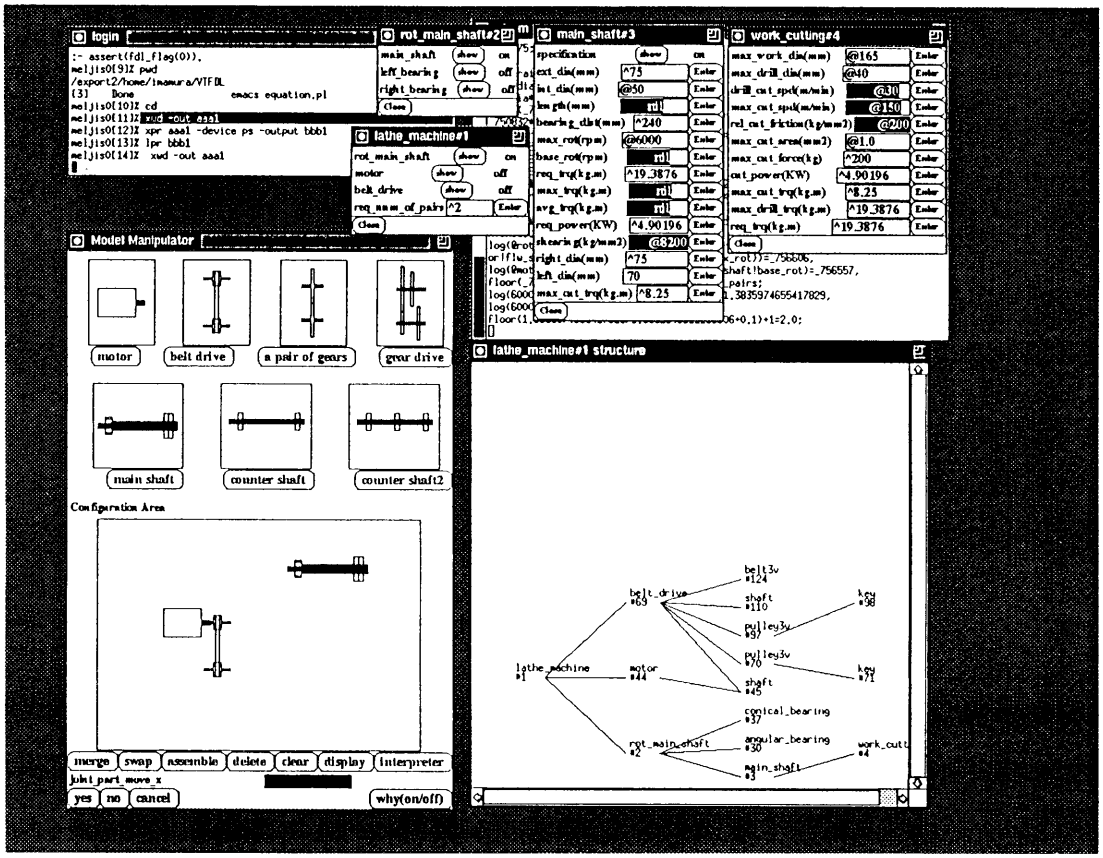


Fig. 6.6 Structuring a lathe model (1)

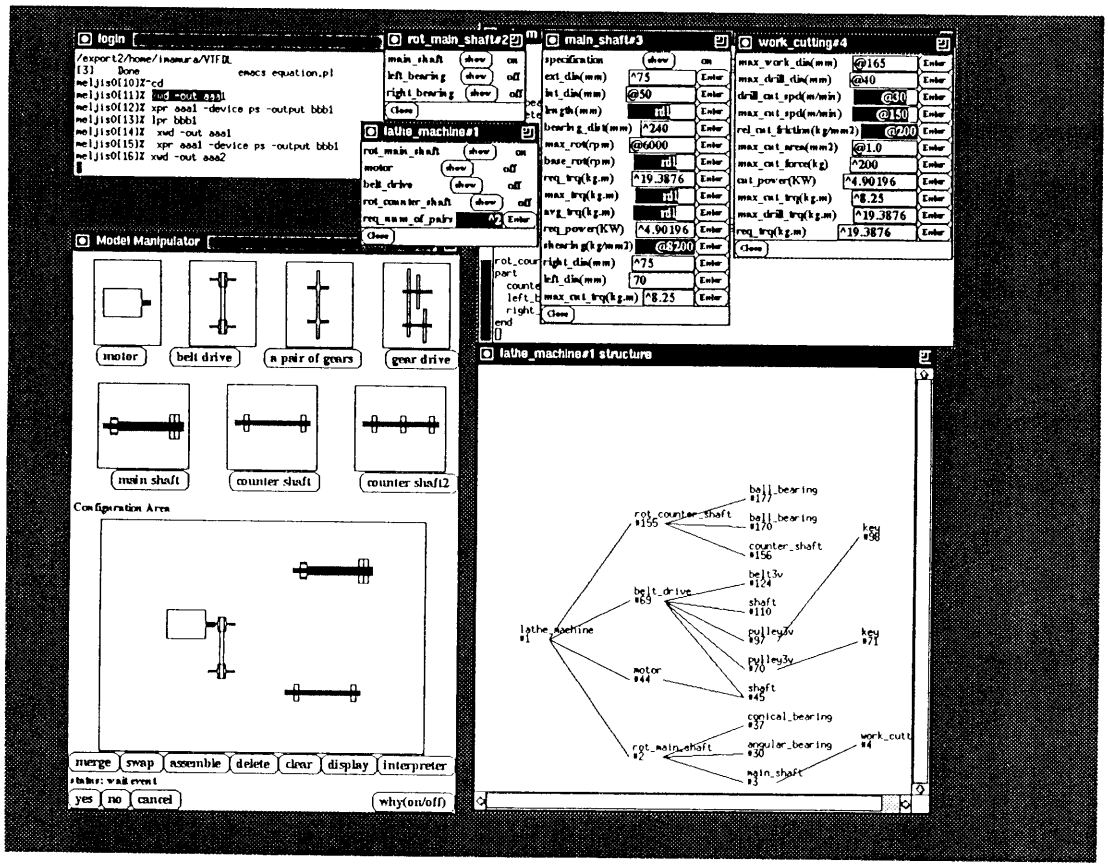


Fig. 6.7 Structuring a lathe model (2)

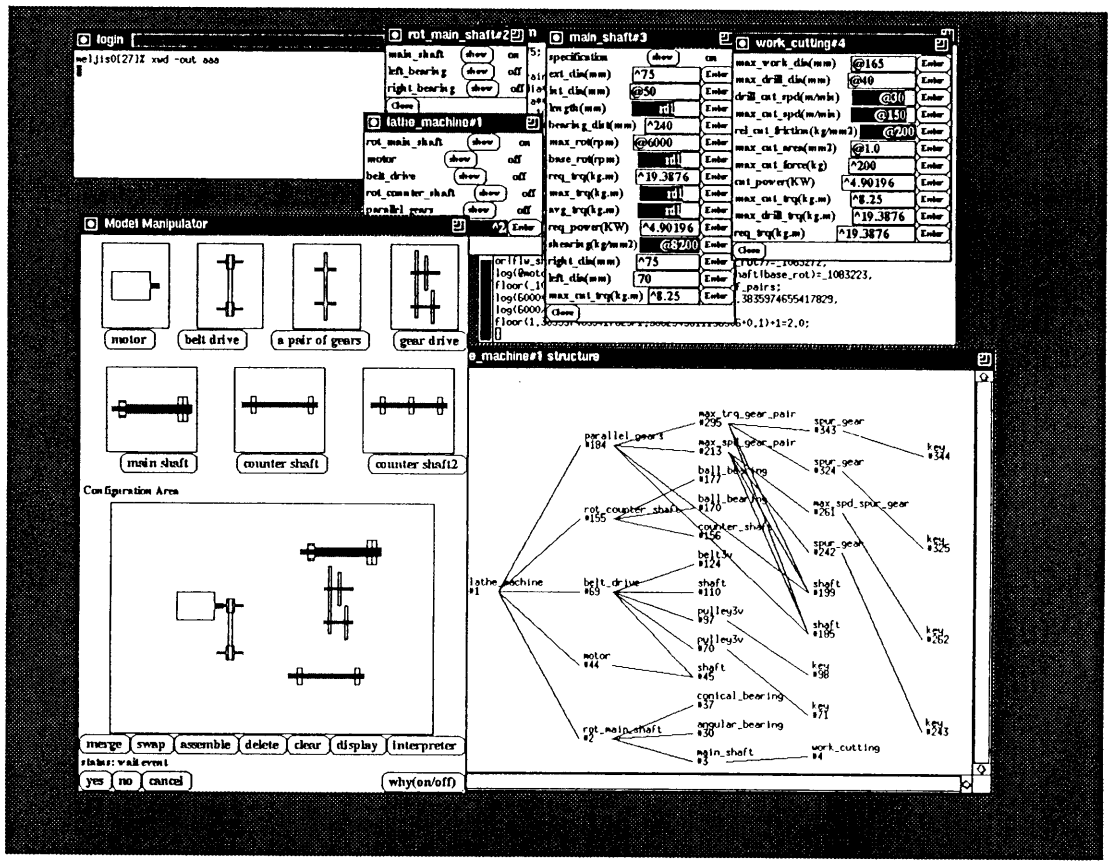


Fig. 6.8 Structuring a lathe model (3)

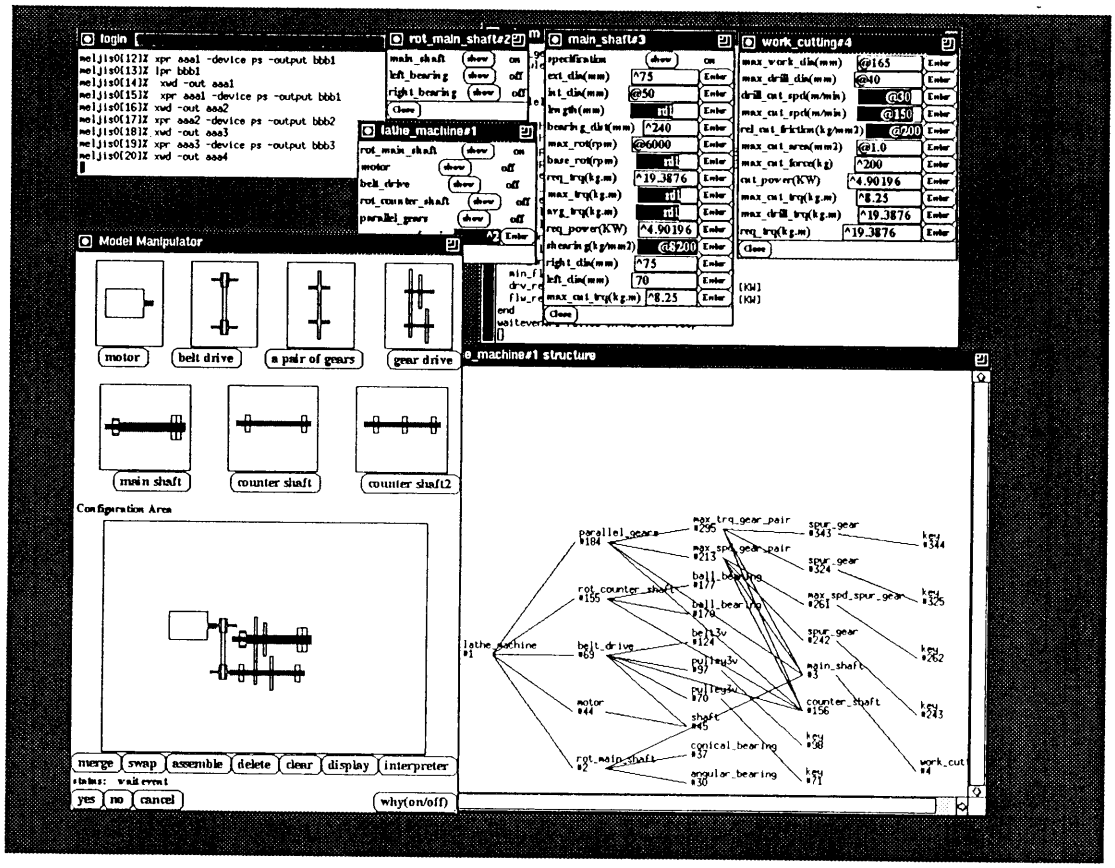


Fig. 6.9 Structuring a lathe model (4)

6.2.2 配置演算子による自動構造構築

6.2.1 ユーザの指令によって行った旋盤モデルの構築は、必要となる配置演算子を用意しておくことにより自動的に行うこともできる。**図 6.10** に配置演算子を取り入れた旋盤モデルの記述例を示す。ここで、 num_of_pairs は現モデルの歯車段数、req_num_of_pairs は必要となる歯車段数を表す。設計計算の結果、両者の値に違いが生じると、起動条件が一致する配置演算子が起動され、モデルの自動構築、自動構造変更操作が行われる。**図 6.10** では、配置演算子として、順番に、０段から２段、２段から３段、３段から２段、０段から３段、０段から４段へ歯車段数変更を行うものが用意されている。なお、このモデルでは auto_conf というパラメータにより、配置演算子の起動を制御できるようにしている。auto_conf=1 の場合は、すべての配置演算子は活動可能であり、auto_conf=0 の場合は活動を抑止される。後者の場合は、ユーザ指令によるモデル構築を行うことになる。

```
class lathe_machine
part
 rot_main_shaft:= #rot_main_shaft,
 motor:= #motor,   belt_drive:= #belt_drive;
parameter
 num_of_pairs integer [default(0)],
 req_num_of_pairs integer [],
 auto_conf integer [default(1)];
where
 motor!flw_shaft==belt_drive!drv_shaft;
  @rot_main_shaft!shaft!req_power/0.75 = ^motor!req_power;
  log((@rot_main_shaft!shaft!max_rot)  * (@rot_main_shaft!shaft!req_trq)/
       (@motor!flw_shaft!max_trq * @motor!flw_shaft!max_rot)) = X,
  log((@motor!flw_shaft!max_rot/ @motor!flw_shaft!base_rot)) = Y,
  floor((X/Y+0.1))  + 1 = ^req_num_of_pairs;
 @req_num_of_pairs=2, @num_of_pairs=0, @auto_conf=1 ==>
    :add_part(self,rot_counter_shaft_1,#rot_counter_shaft,2),
    :add_part(self,parallel_gears_1,#parallel_gears,3),
    :merge(belt_drive!flw_shaft,rot_counter_shaft_1!shaft),
    :merge(rot_counter_shaft_1!shaft, parallel_gears_1!drv_shaft),
    :merge(rot_main_shaft!shaft,parallel_gears_1!flw_shaft),
    :set(self,num_of_pairs,2);
 @req_num_of_pairs=3, @num_of_pairs=2, @auto_conf=1 ==>
  :assemble(parallel_gears_1, gear_pair_2,#gear_pair),
  :set(self,num_of_pairs,3);
 @req_num_of_pairs=2, @num_of_pairs=3, @auto_conf=1 ==>
  :delete_slot(parallel_gears_1, gear_pair_2),
  :set(self,num_of_pairs,2);
 @req_num_of_pairs=3, @num_of_pairs=0, @auto_conf=1 ==>
    :add_part(self,rot_counter_shaft_1,#rot_counter_shaft,2),
    :add_part(self,parallel_gears_1,#parallel_gears,3),
    :merge(belt_drive!flw_shaft,rot_counter_shaft_1!shaft),
    :merge(rot_counter_shaft_1!shaft, parallel_gears_1!drv_shaft),
   :merge(rot_main_shaft!shaft,parallel_gears_1!flw_shaft),
   :assemble(parallel_gears_1, gear_pair_2,#gear_pair),
   :set(self,num_of_pairs,3);
 @req_num_of_pairs=4, @num_of_pairs=0, @auto_conf=1 ==>
    :add_part(self,rot_counter_shaft_1,#rot_counter_shaft,2),
    :add_part(self,parallel_gears_1,#parallel_gears,3),
    :merge(belt_drive!flw_shaft,rot_counter_shaft_1!shaft),
    :merge(rot_counter_shaft_1!shaf,   parallel_gears_1!drv_shaft),
   :add_part(self,rot_counter_shaft_2,#rot_counter_shaft,4),
   :add_part(self,parallel_gears_2,#parallel_gears,5),
   :merge(rot_counter_shaft_2!shaft,parallel_gears_1!flw_shaft),
   :merge(rot_counter_shaft_2!shaft,parallel_gears_2!drv_shaft),
   :merge(rot_main_shaft!shaft,parallel_gears_2!flw_shaft),
   :set(self,num_of_pairs,4);
                                        :
                                        :
end.
```

Fig.6.10 A lathe description with configuration operators

6．3　エレベータ設計

本節では、VT システムと呼ばれるウエスティングハウス社のエレベータ設計用エキスパートシステムであつかった設計問題を取り上げる。VT の設計モデルは、大規模コンフィギュレーション設計の典型的な問題として，スタンフォード大学のオントロジープロジェクトから公開されており、コンフィギュレーション設計のベンチマークテスト用に広く提供されている。ＶＴ問題は，次のような特徴を持つ[Yost & Rotherfluth, 96]。

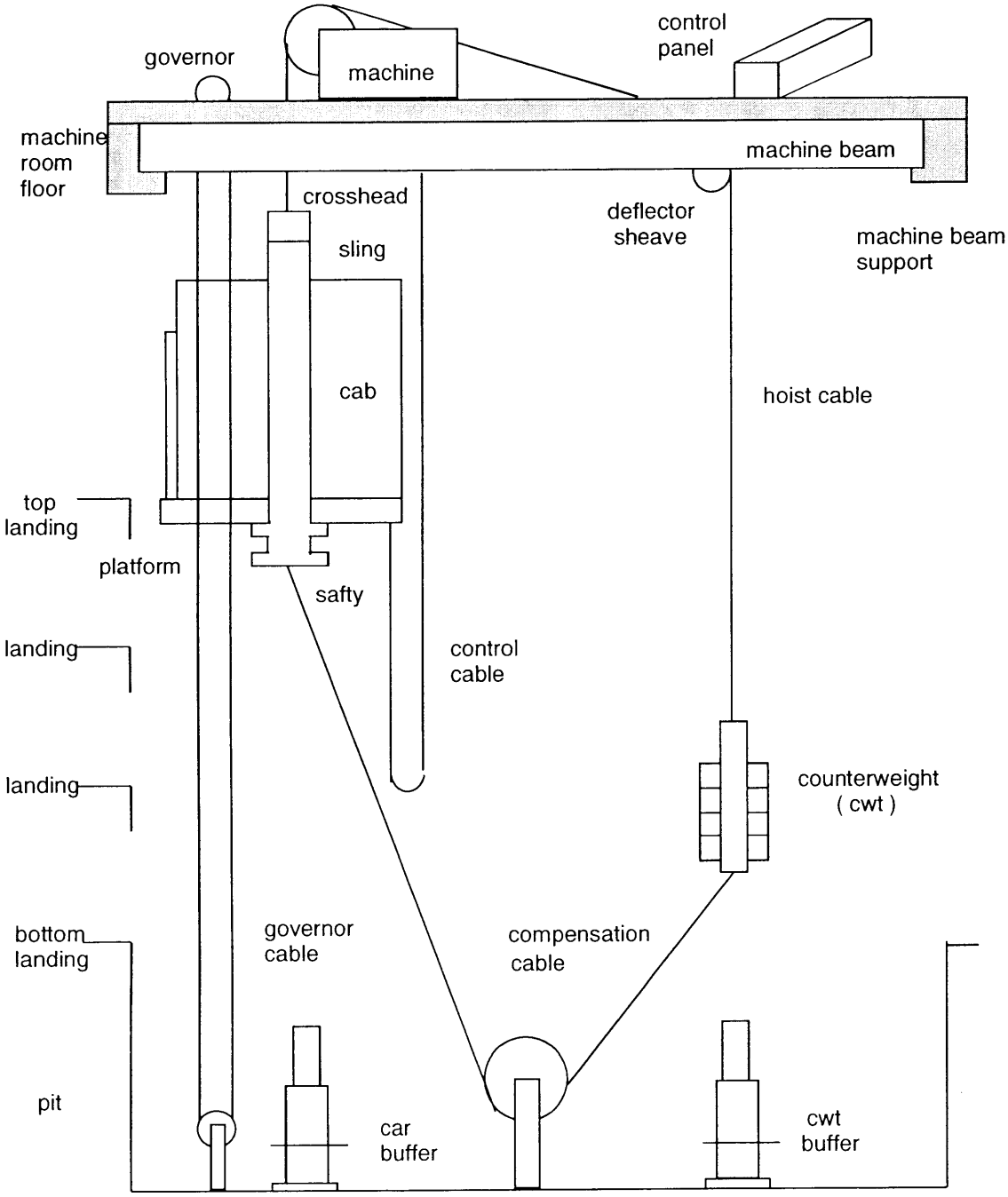


Fig. 6.11　Elevator system

（1）大規模設計問題である。

コンポーネント数 27 個，パラメータ数 301 個，多属性制約数 295 個，等値制約数 7 個、単一属性制約数 34 個からなる大規模設計問題である。

（2）過小制約問題である。

要求仕様として入力された 26 個のパラメータ値に対して、設計式（制約）の数が 300 以上、データベースから選択すべきコンポーネントの数が 25 個程度あるので、解を決定

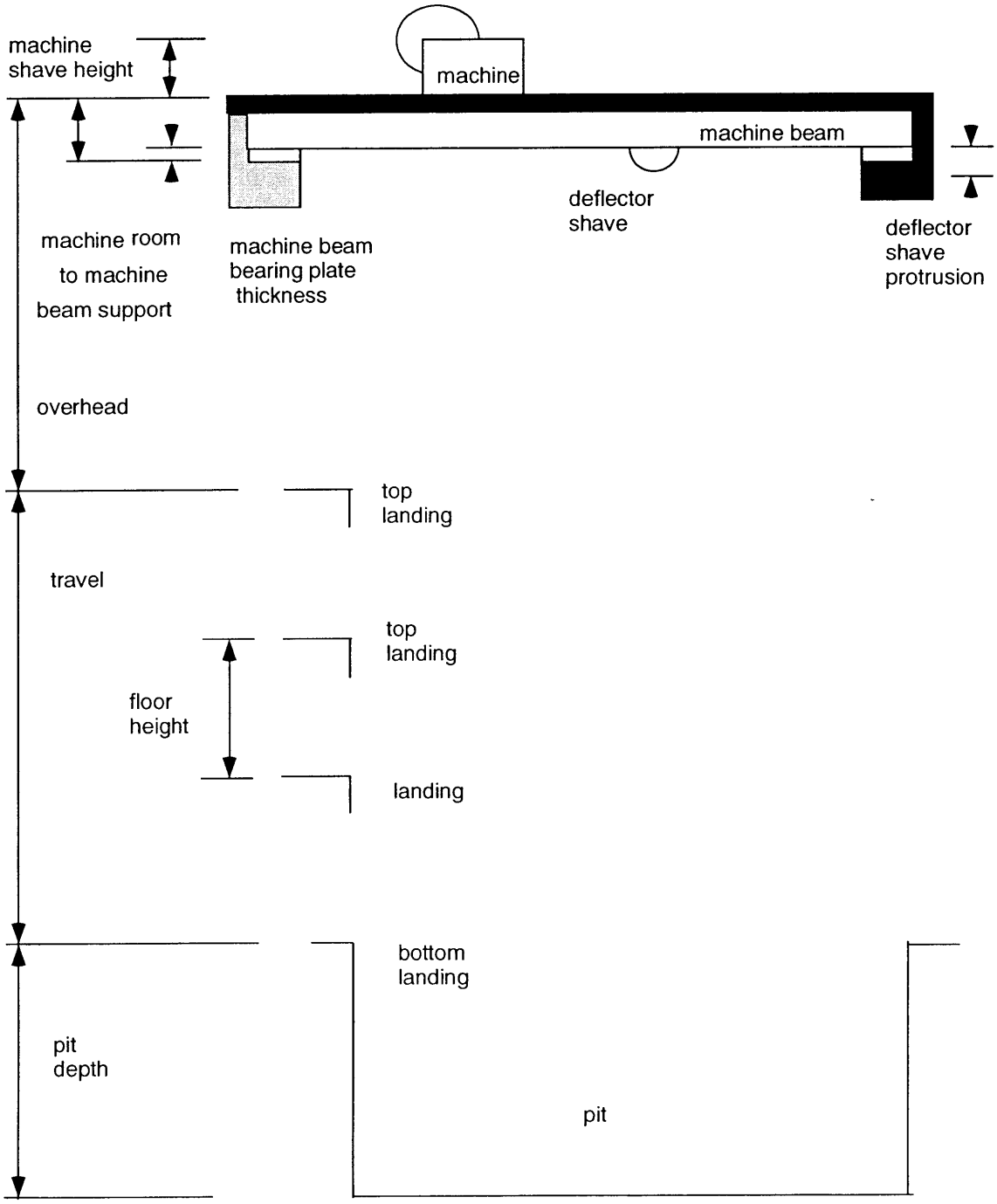


Fig. 6.12 Vertical hoistway dimensions

的に求めることができない。未決定のパラメータの解の組合せから解空間の大きさを計算すると 10^{17} のオーダーである。これは、解候補1個あたり制約計算に 10 秒かかると仮定すると、すべての解候補を検証するのに1千億年かかる計算になる。

（3）実効制約集合が動的に変化する

多くの設計計算式が条件付きで成立する。つまり、計算中に解くべき制約集合が変化する。

以上のことから、VT 問題の解空間は巨大、非単調、かつ可変といえる。このような性格を持つ VT 問題は設計システムの評価用の例題として十分な内容を持つものであり、FDL でも評価用事例として採用することにした。

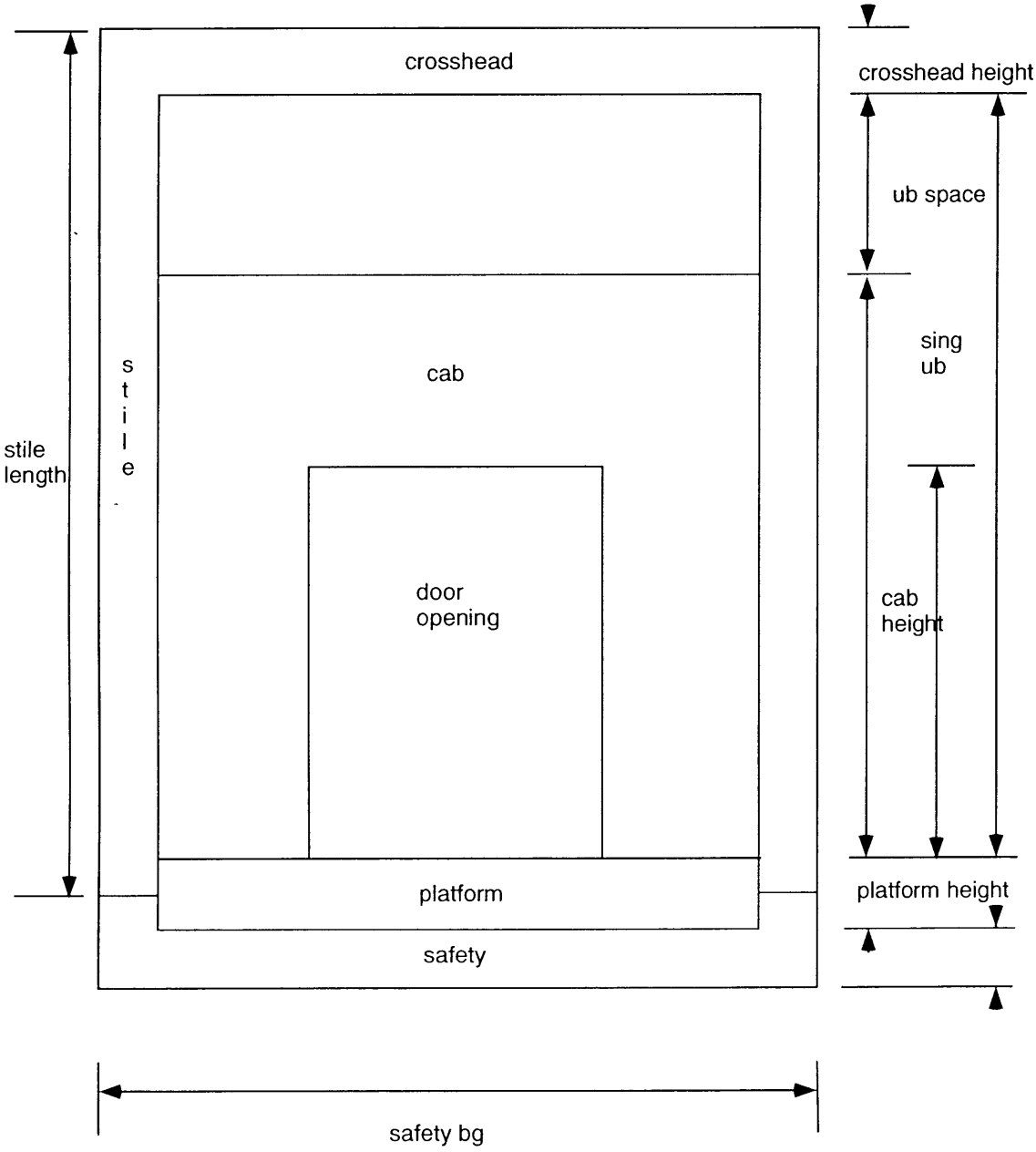


Fig.6.13 Car dimensions

6.3.1 エレベータ設計問題の概要

エレベータシステムは**図 6.11** に示すようなコンポーンネントから構成される。主要なコンポーネントは次のようなものである。

（1）エレベータの主要なコンポーネント

エレベータ通路(hoistway)：ピット床からマシン室床まで垂直にのびるエレベータの通路。エレベータかご（car assembly)はエレベータ通路の前側に位置し、釣り合いおもり(counterweight)は後ろ側に位置する。エレベータ通路は、ピット(pit)、行程(travel)、頭上空間(overhead)から構成される。ピットはボトムランディング(bottom landing)から下に位置する空間で、種々の機械が納められている。頭上空間はトップランディング(top landing)と機械室床の間の空間である。行程はボトムランディングとトップランディングの間の空間である。

エレベータかご(car assembly)： 乗客昇降室(passenger cab)、支持構造(supporting structure)及びに安全メカニズム(safety mechanisms)から構成される。支持構造はプラット

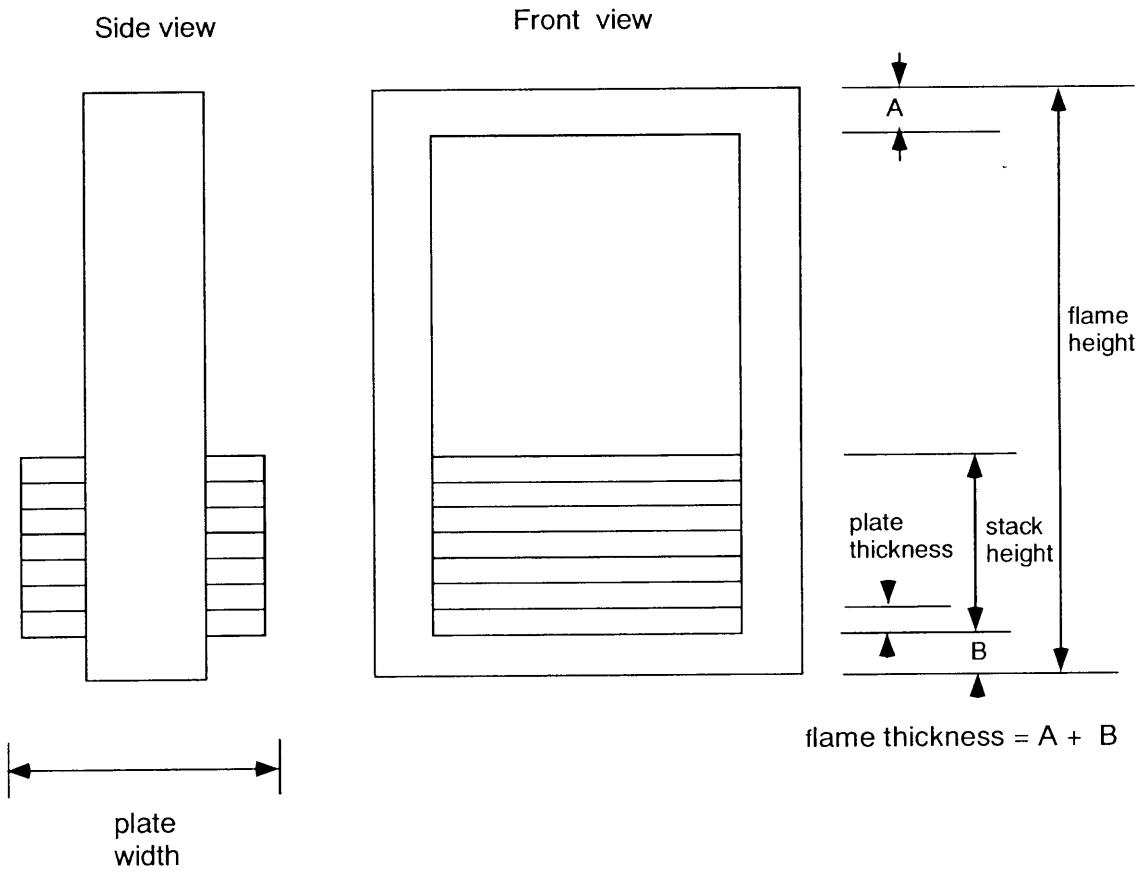


Fig. 6.14 Counterweight dimensions

フォーム(platform)とつり索(sling)から構成される。つり索は３本の梁(beam)から構成され、縦方向の２本の梁をスタイル(stile),乗客昇降室上部にある横方向の梁をクロスヘッド(crosshead)と呼ぶ。エレベータケーブルはクロスヘッドに取り付けられ、スタイルに取り付けられたガイドシュー(guideshoes)はガイドレールでガイドされる。

釣り合いおもり(counterweight assembly)： エレベータかごと積載荷重の一部と釣り合いをとるためのもので、直方体のフレームと数枚のおもりプレートから構成される。釣り合いおもりはエレベータ通路の後部壁面にＵブラケットによりガイドされる。

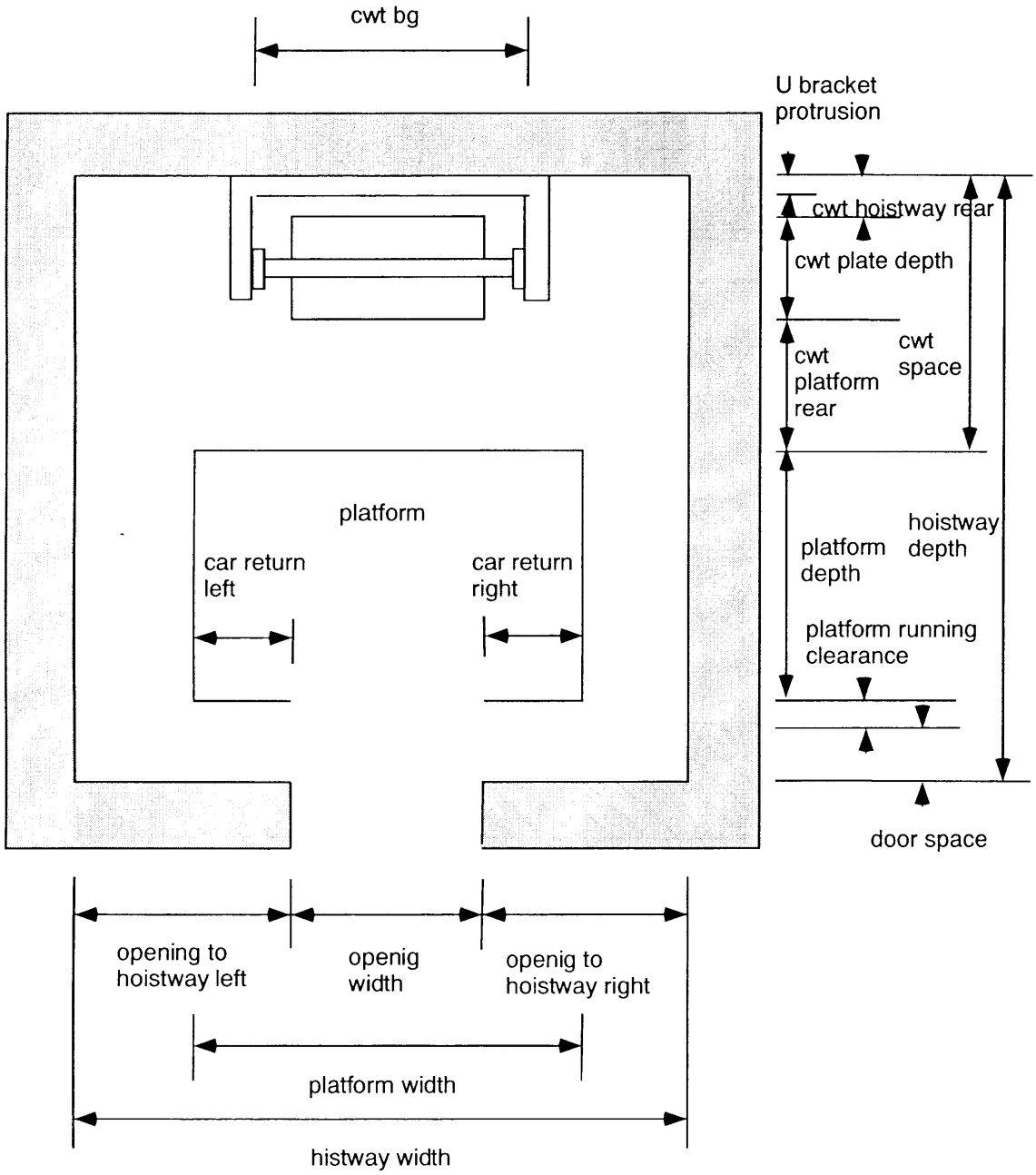
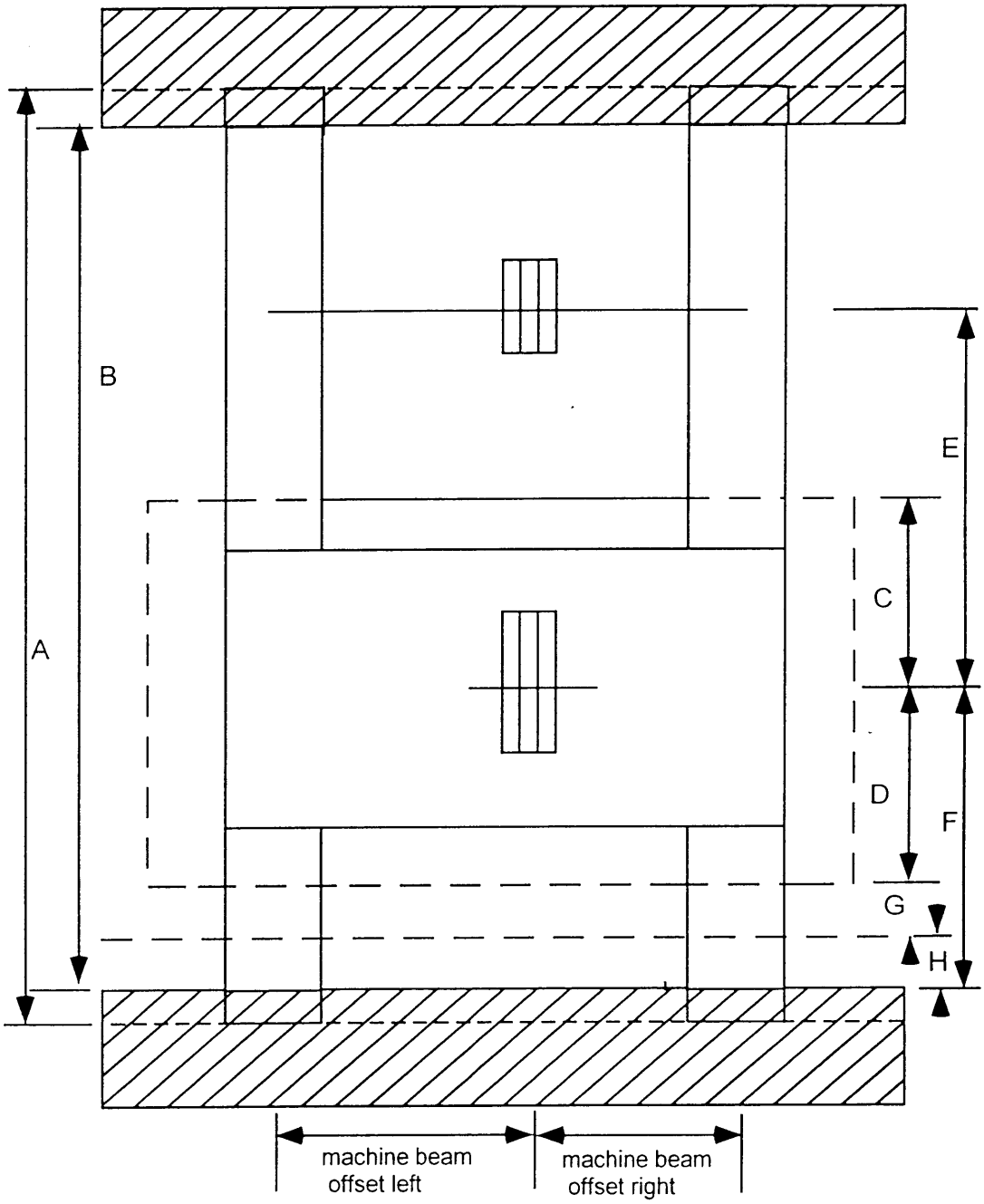


Fig.6.15 Horizontal hoistway dimensions



A: machine beam length
B: machine beam support distance
C: car cable hitch to platform rear
D: car cable hitch to platform front
E: car cable hitch to cwt cable hitch
F: machine shave center to front machine beam support
G: platform to hoistway front
H: hoistway to front machine beam support

Fig.6.16 Overhead hoistway dimensions

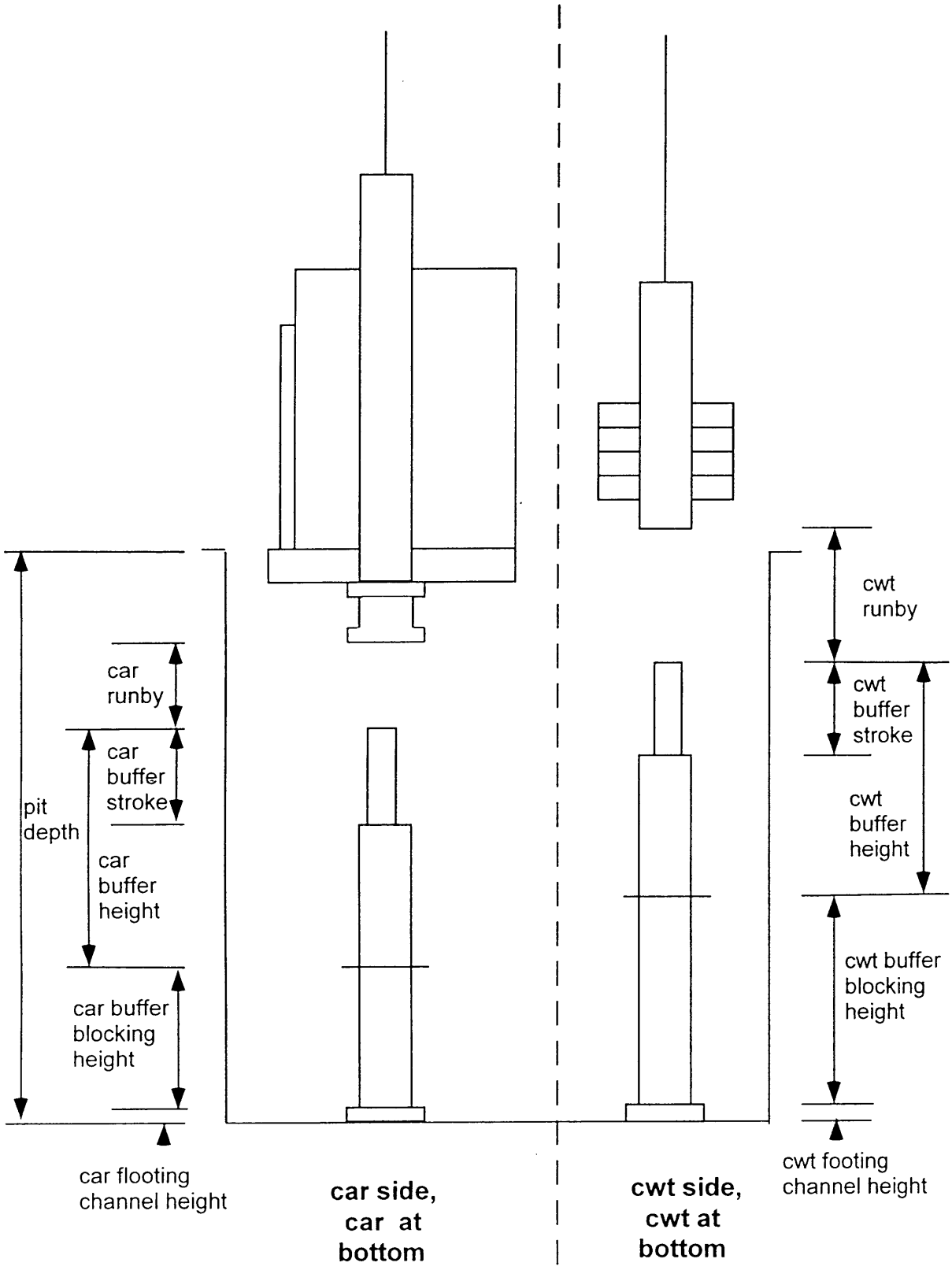


Fig.6.17 Pit dimensions

機械ユニット(machine unit)： モータとエレベータケーブルを導く滑車(traction sheave)からなり、エレベータかごの真上にある梁(machine beam)によって補強される機械室フロアの上に設置される。2番目の滑車はディフレクター滑車(deflector sheave)と呼ばれ、エレベータケーブルを釣り合いおもりの真上に導く。

安全装置(safety mechanism)： 過速度調速装置(overspeed governor)とバッファーがある。過速度調速装置はプラットフォームにつなげられたループ状のガバナーケーブルと機械室内の速度センサーからなり、速度オーバーが検出されると、セーフティー(safety)と呼ばれるブレーキメカニズムが働く。セーフティーはプラットフォームの下に取り付けられ、ガイドレールをグリップする。バッファーはピット内に設置されたオイルに満たされたピストンで、衝撃力を吸収する。エレベータかごと釣り合いおもりに一つずつ設置される。

他のケーブル： エレベータケーブルとガバナーケーブル以外に、エレベータかごと釣り合いおもりの間をつなげる補償ケーブルと制御ケーブルがある。

（2）設計要求仕様

エレベータ設計には、顧客から次のような要求仕様が与えられなければならない。

・最大積載重量： 2000ポンド以上4000ポンド以下。
・エレベータスピード： 200，250，300，350，400feet/min.から選択。
・ドア開閉タイプ(door opening)： センターとサイドがある。
・ドアスピード： シングルとダブルがある。
・ドアの開く方向(opening strike side)：左と右がある。
・プラットフォームの幅と奥行き： インチ表示。
・エレベータキャブの高さ： インチ表示。
・インターコム、ランタン(lantern)、エレベータ電話、フロア位置表示装置： いずれも装備の有無をyes、noで表す。
・オーバーヘッド(overhead)： 最上階フロアと機械室フロアの距離。インチ表示。
・トラベル(travel)： 最上階(top landing)と最下階(bottom landing)の距離。インチ表示。
・ピット深さ(pit depth)： 最下階フロアから下のピットの深さ。インチ表示。
・フロア間距離(floor height)： 各フロア間の距離。すべて等距離とする。インチ表示。
・エレベータ通路幅と奥行き： インチ表示。
・フロア数(number of openings)：エレベータが止まるフロアの数。
・各フロア建物開口部の幅と高さ(opening width and depth)： すべてのフロアで同じサイズとする。インチ表示。
・開口部左側とエレベータ通路左側の距離(opening to hoistway left)：インチ表示。
・機械室支持タイプ(machine beam support type) ：pocketとsteelがある。pocketは鉄筋コンクリート壁につくったくぼみによる支持、steelはI型の梁による支持である。
・機械室支持梁間距離(machin beam support distance)：前梁と後梁の距離。インチ表示。

・エレベータ通路前部と前梁の距離(hoistway to front machine beam support)：インチ表示。
以降、単位表示は省略するが、長さはインチ、重さはポンド単位を使う。

（3）建物及びエレベータ部品の寸法

図 6.12 から**図 6.17** にかけて、建物及びエレベータ部品の寸法を示す。**図 6.12** はエレベータ通路周りの寸法、**図 6.13** はエレベータかごの寸法、**図 6.14** は釣り合いおもりの寸法、**図 6.15** はエレベータ通路断面の寸法、**図 6.16** はオーバーヘッドの寸法、**図 6.17** はピットの寸法を表す。

（4）コンポーネントの選択

エレベータシステムを構成するコンポーネントとして、ドア、プラットフォーム、つり索、セーフティー、エレベータかご、エレベータバッファー、釣り合いおもり、釣り合いおもりバッファー、制御ケーブル、エレベータケーブル、補償ケーブル、ガバナーケーブル、ディフレクター滑車、機械、(機械室の) 梁、モータがあり、それぞれあらかじめ用意されたモデル集合からルール、計算式などを使って適切なモデル選択する。コンポーネント選択についてすべて記述すると大分量になるので、ドア、プラットフォーム、つり索、セーフティーについてのみ紹介し、残りは文献に依存することにする。

(4a)　ドア

ドアはスピードがシングルとダブルの２タイプ、開閉タイプが左、右、中央の３タイプ、計６タイプがある。ドアの上にはドアの駆動装置があり、エンジンとヘッダー(header)からなる。エンジン重量は 135 ポンド、ヘッダー重量は定数×ドアの開く幅である。定数はシングルスピード左右開閉タイプが 1.25、ダブルスピード左右開閉タイプが 1.5、シングルスピード中央開閉タイプが 1.33、ダブルスピード中央開閉タイプが 2.083 である。

(4b)　プラットフォーム

2.5B、4B、6B の３種類のモデルがあり、2.5B が最小、6B が最大モデルである。プラットフォームの幅を W、奥行きを D、エレベータ積載重量を C とおくと、C ≦ 2500、W ≦ 84、D ≦ 60 の条件ではモデル 2.5B を、C ＞ 2500、W ≦ 128、D ≦ 108 ないし C ＞ 2500、W ≦ 115、D ≦ 126 の条件ではモデル 4B を、それ以外の条件ではモデル 6B を選択する。

プラットフォームの重量はモデルごとに次の式で求める。

Model2.5B: S+0.318(5.06W+0.11WD+(D-7.6)(3.14+0.8AP))

Model4B:　35+WX+2.774D+0.03WD+0.226Q+AP(0.226D-Z)

Model6B:　35+WX+3.228D+0.34Q+AP(0.226D-Z)

W: 幅、D: 奥行き、AP, S, X, Z: ウエイト係数、Q: ドア開口幅

AP= |(W-0.125)/9.5|　for Model4B,　　AP= 3　for Model2.5B if W ≦ 60,

AP= 4　for Model2.5B if 60＜W ≦ 76,　　AP= 5　for Model2.5B if W ＞ 76,

AP= 5　for Model6B if W ≦ 67,　　AP= 6　for Model6B if 67＜W ≦ 76.5,

AP= 7　for Model6B if 76.5＜W ≦ 86,　　AP= 8　for Model6B if 86＜W ≦ 96.5,

AP= 9　for Model6B if 96.5＜W ≦ 105,　AP= 10 for Model6B if W>105

S=63 if capacity= 2000 & D ≦ 53,　otherwise S= 71

X=2.26 for model4B single speed door, X=2.354 for model4Bdouble speed door

X=2.441 for model6B single speed door, X=2.394 for model6Bdouble speed door

Z=1.72 for single speed doors, Z=2.0 for double speed doors

(4c) つり索(sling)

つり索には 2.5B-18, 2.5B-21, 4B-HOSP, 4B-GP, 6C の 5 種類のモデルがあり、重量は式 AW+BL+C により求められる。ここで、Wはプラットフォームの幅、L はスタイル(stile)の長さ、A,B,C は定数で次のような値をとる。

A= 1.5, B= 1.002, C= 56 for sling model 2.5B-18

A=1.75, B= 1.002, C= 94 for sling model 2.5B-21

A=1.8, B= 1.2, C= 223 for sling model 4B-HOSP

A=2.5, B= 1.6, C= 223 for sling model 4B-GP

A=3.1, B= 2.2, C= 317 for sling model 6C

(4d) セーフティ(safety)

B1, B4, B6 の 3 種類のセーフティがある。プラットフォーム幅を W で表すと、モデルの選択基準は次の通り。

model B1 if $W \leqq 93$, model B4 if $93 < W \leqq 114$, model B6 otherwise.

セーフティービームの長さは次式で表される。

W+2.25 for model B1 and B4, W+2.625 for model B6

セーフティービームの高さはモデル B1 で 9、モデル B4 で 10、モデル B6 で 13 である。

セーフティービームの重量は AW+B で表され、A,B の値は次の通り。

A= 1.69, B= 1 for model B1, A= 2.3, B= 540 for model B4,

A= 2.6, B=1035 for model B6

(5) 制約計算

各ケーブルの荷重条件、各種寸法の範囲、エレベータの上方マージン距離(car overtravel)、釣り合いおもりの数、つり索モデル、エレベータケーブル安全係数(hoist cable safety factor)、トラクション比(traction ratio)などに関して、制約関係が存在し、それらを満たす適切な値を探索する必要がある。制約の内容については、文献に依存することにする。

6.3.2 要求仕様

エレベータ設計への要求仕様として次のような値をとった。

car cab height	96
car capacity	3000
car intercom	no
car lantern	no
car phone	yes
car position indicator	yes
car speed	250
door opening strike side	right
door opening type	side
door speed	double
hoistway depth	110
hoistway floor height	165
hoistway overhead	192
hoistway pit depth	72
hoistway travel	729
hoistway width	90
machine beam support bottom to machine room top	16
machine beam support distance	118
machine beam support front to hoistway	3
machine beam support type	pocket
opening count	6
opening height	84
opening to hoistway left	32
opening width	42
platform depth	84
platform width	70

6.3.3 パラメータ値の生成と検証

VT 問題は過小制約問題（パラメータ数に比較し、制約関係が過小であり、パラメータ値を決定的に決めることができない）であり、要求仕様入力と制約伝播の組み合わせで解を求めることはできない。また、VT で扱われる制約は多様（等式、不等式、述語論理による関係、条件による値の決定などがある）かつ動的（条件により有効な制約式そのものが変化する）であり、線形計画法やダイナミックプログラミングのような最適化手法の適用も困難である。そこで、要求仕様入力による制約計算によっても得ることのできないパ

ラメータ値に関しては、仮定値の生成と制約計算による無矛盾性の検証を組み合わせた、生成検証法により値を求めることにした。

値の自動生成を行ったのは以下のパラメータである。

パラメータ（モデル）名	生成値
opening to hoistway left	[31, 32, 33, 34, 35]
sling model	[m1, m2, m3, m4, m5]
counterweight between guiderails value	[28, 38, 54]
counterweight buffer quantity	[1, 2]
counterweight plate depth	[7.0, 8.0, 9.0, 10.0, 11.0, 12.0]
car supplement weight	[0, 100, 200, 300, 400, 500]
machine beam model	[m1,m2,m3,m4,m5,m6,m7,m8,m9,m10]
machine model	[m1,m2,m3,m4]
hoist cable model	[m1,m2,m3,m4,m5,m6,m7,m8]
car guiderail model	[m1,m2,m3,m4,m5]

仮定値生成の数はかなり減らすことができた。**4.4.3** で述べたように、FDL では多属性制約において@と^が二重に指定された場合、^が優先される仕組みをもつ。別の言い方をすると、制約の依存関係を自動的に解析し、入力変数、出力変数の整理を自動的に行なうことができる。その結果、エレベータ設計の例では入力変数をかなり減らすことができ、仮定値生成数も大幅に削減できた。

スタンフォード大学が公開している VT 問題の LISP 表現では、制約の依存関係を利用した仮定値の生成ができなかったので、仮定値の組み合わせの数は

$$5 \times 1 \times 1 \times 1 \times 2 \times 2 \times 3 \times 2 \times 2 \times 4 \times 5 \times 6 \times 3 \times 2 \times 2 \times 4 \times 2 \times 6 \times 5 \times 2 \times 3 \times 2 \times 7 \times 9 \times 6 \times 2 \times 5 \times 7 \times 6 \times 6 \times 4 \times 4 \times 4 \times 4 \times 2 \times 6 \times 3 \fallingdotseq 8.74E+18 \quad (6.12)$$

となったのに対し、FDL では仮定値の生成数を

$$5 \times 5 \times 3 \times 2 \times 6 \times 6 \times 10 \times 4 \times 8 \times \cdot 5 = \quad 8.64E+6 \quad (6.13)$$

に絞ることができた。

6.3.4 設計計算結果

計算は SUN ULTRA1（メモリ 256MB）、Sicstus Prolog Ver.3.7 で行った。

最初の設計解を見つけ出すまで 6 分 26 秒の時間がかかった。また、全解探索を行ったところ、458 通りの解を 10 時間 5 分 6 秒で求めることができた。

6.3.5 考察

本件は、ストラスクライド CAD センター研究員 Dr. Iain Donldson に、FDL の設計能力の評価用に薦められて取り組んだものである。当センターでも、この問題への取り組みを検討したことがあるが、設計問題表現が多様かつ複雑で、計算量も大規模のため、断念した経緯がある。国内では、九州工業大学長澤勲教授、梅田助手のグループが ADL 設計言

語を使って設計解を求めることに成功しているが、最初の解を求めるのに8時間程度かかったとされている。このように本設計問題は非常に困難な課題で、これを短時間で解ければFDLの高度な制約解決能力を示すことができると考えた。

しかしながら、本件の取り組みにはかなりの時間を必要とした。理由は次のようなことにある。

(1) 本件は、スタンフォード大学 Knowledge Sharing Effort Public Library (http://www-ksl.stanford.edu/knowledge-sharing/) が公開する LISP で記述した VT 問題を FDL の記述に書き替えたものを最初は使用した。オリジナルの LISP 記述は 500KB 程度あり、その書き換え作業に数ヶ月の時間を費やした。

(2) VT 問題では、生成検証法、起動条件付き制約表現、デバッグ時に大量の初期値（要求仕様など）を繰り返し入力するためのマクロファイルなどの開発を新たに行う必要があった。

(3) スタンフォード大学の LISP 記述は、表現のミスが多々あり、制約記述の相互矛盾のため解を求めることが不可能なものであった。しかしながら、当時は当該 LISP 記述以外に参考とするものがなかったので、表現ミスの存在も明確に判断できず、試行錯誤に多くの時間を費やした。スタンフォード大学側も大量のミスの存在を認知しておらず、適切なアドバイスを受けることができなかった。

(4) 後に、九州工業大学梅沢助手の助けで VT 問題を英語で表現したオリジナルのドキュメントを入手することができ、それをもとに、ミスの修正や、制約式の表現形態の変更を行うことができ、問題解決に至ることができた。

本件に取り組んだ結果、FDL に関して次のような評価を与えることができると考えている。

(1) 高い対象表現力

VT 問題の LISP 記述は 500KB 程度が必要であったのに対し、FDL 記述では 70KB 程度であった。また記述の読みやすさを比べても、LISP では括弧の多い独特の表現で内容の判読に時間がかかるのに対し、FDL では数式は本来の形式に近い形でわかりやすく表現しており、ルール記述も判読しやすい。また、単一属性制約と多属性制約を異なる形式で表現していることも、読みやすくしている。

(2) 高い制約解決能力

最初の解を求めるのに6分程度、全ての解（458 通り）を求めるのに 10 時間程度ですんだ。オリジナルの VT システムは VAX780 システムで1つの解を求めるのに、7時間から21時間程度かかっている。マシン、OS などの違いがあり、計算時間の単純比較は適当ではないが、オリジナルのシステムが VT 用に特別にチューニングされた専用システムであることを考慮すれば、汎用設計言語である FDL の制約解決能力の高さを示すことができたと考える。

4.4.3

制約解決能力の高さについて考察すると次のような理由が考えられる。

(1) FDL の表記が読みやすくわかりやすいので、表現上の冗長性や無駄などを LISP から

FDL への変換の過程で自然に排除できた。
(2) **6.3.3** で述べたように、制約間の依存関係を使って入力変数の数を減少させることができ、仮定値生成数も大幅に削減できた
(3) FDL では制約解決で変数の値を決定していく過程を記録し、その記録を使って効率よく必要なポイントまで戻って別解を求める計算に入れるようになっているので、制約解決におけるバックトラックを高速に行うことができた。

6．4　実験評価

FDL は英国ストラスクライド大学と国内工作機械メーカーに評価を依頼した。このうち、国内工作機械メーカによる実験評価については、第3者による詳細な記録[IPA, 97]が残っているのでここに転載する。

FDL-II 実験報告書
実験者：　日立精機株式会社技術本部開発部　高下二郎理事
立ち会い：　機械技術研究所　今村　聡
　　株式会社富士総合研究所　安達　康

Ⅰ　実験主体における、実験ソフトウエアの対象とする分野に関する予備知識、および経験

実験主体は、規模の大きい、あるいは商品としてのソフトウエアを直接製作した経験はないが、実験データを分析、整理するためのプログラムとか、ある種の設計に要する計算をするためのプログラムを自作するといった程度の経験がある。使用言語はパソコン上での FORTRAN、BASIC、一部アセンブラを経験している。また、実験ソフトウエアの対象である機械設計に関しては、豊富な実務経験ならびに、専門的な知識を有している。

実験を開始するにあたり、FDL-II に関する概略やデモが実験主体に対し約 1 日にわたって説明が行われた。

Ⅱ　ソフトウエアのインストールについて
略

Ⅲ　実験課題
(1) 実験課題の選定理由

機械の設計過程には、概念設計・基本設計・詳細設計の諸段階から構成されている。(中略)

本実験の対象である FDL-II は、機械の設計過程における基本設計段階を支援する制約対象指向言語である。基本設計に対する計算機支援としては、対象の構造と特性を表すパラメータ集合が既知である場合に利用されるパラメトリック設計支援と、あらかじめ与えられた構成要素の組み合わせにより、制約条件を満たす構造を求める組み合わせ（コンフィギュレーション）設計支援があり、FDL-II はこの両者を同時並行に行うことが可能で

ある。

このような特性を生かした機械設計の実験を行うため、課題は、実験対象となる機械の要求仕様が明確であり、なおかつ FDL-II の設計で得られた結果と比較検討が可能であることが望ましい。

そこで、日立精機株式会社において製品化されている「剛速 NC 旋盤 TG series」の TG25 を実験課題に選定し、この TG25 の要求仕様を用いて、FDL-II による機械設計を試みるものである。（注：カタログに記載されている製品仕様をそのまま要求仕様として使った）

(2) 課題の詳細につて

TG25 の要求仕様による、旋盤設計の実験を行う。旋盤は、いろいろな構成が考えられ、最適な構成は要求仕様に基づく設計計算により明らかとなる。また、要求仕様の変更や用意される機械部品により構成も大きく変わる。FDL-II では、主軸に対して必要に応じて機械部品をライブラリより取り出し、組み付けることにより、要求仕様を満足させた旋盤の構成を作り上げることができる。

以下に実験で使用した TG25 の要求仕様項目を記述する。

・最大切り込み面積、最大ワーク径、主軸最大回転数、最大ドリル径、軸受け要求寿命、
　主軸材料、主軸貫通穴径、プーリ比、ギア比

なお、プーリ比、ギア比については、設計過程において試行錯誤しながら入力する項目である。

Ⅳ　FDL-II を用いた課題の処理

(1) プログラムのサイズ・コーディング量

FDL-II 本体は、全体で約 150KB(Prolog のソースコード)である。なお,Sicstus　Prolog 及びグラフィックパッケージは、その中に含まれていない。今回の課題を実行するにあたり、準備したライブラリのサイズは 47KB である。

(2) 実行時にパフォーマンス

実行時の処理速度は、要求仕様の数値を入力してから結果が得られるまで数秒である。組み合わせ設計用ウインドウで、構成要素を追加する操作（モデルの構造融合）においても、グラフィック処理の操作性が劣ることはなかった。

(3) プログラムの変更のしやすさ

FDL-II は動的構造変更が行える機能を有しており、その機能についての操作性に関して、以下のようなことを感じた。

・動的構造変更の操作は、コマンドベースで行われるため、UNIX の知識がないユーザにとっては、操作方法を改善することが望まれる。

・機械構造が決定したものに、後から機械部品を追加するような処理も可能であるが、その時のコマンド入力に厳密性を要する。

Ⅴ　実験結果

要求仕様を満足させる機械構造の設計は、うまくいったといえる。要求仕様を入力した結果、それを満足する部品をライブラリから探し出し、構造を決定することができた。

この計算結果と、実際の TG25 を比較すると、その計算結果は、実際の TG25 の数値とは必ずしも一致していない。その原因としては、モデルの制約式や要求仕様の数値がカタログ値を参考にしていたため、これを十分に満足させるような計算結果を FDL-II が行ったと思われ、実機よりもオーバースペック的な結果が得られたのではないかと考えられる。一方、機械構造については、設計者が検討する際の原案となりうるものが出力結果として得られた。

（注：設計式は要求を確実に満たす緒元を求めるようにできている。一方、製品開発では実機を試作し、その結果予想以上の性能が得られれば、それを製品仕様として採用する。）

FDL-II は、要求仕様を満たす設計を行うが、必ずしも最適な機械構造結果が得られるわけではない。また、実際の設計においては、過去に製造した製品から得られる経験的な知識や製造時のことを考慮して部品を決定しているため、FDL-II の計算結果だけで、明確な設計をすることは難しい。また、プーリ比やギア比などは、設計過程において、設計者が決定する項目であり、設計経験が必要とされる。（注：　ここの記述は FDL-II の能力というより、むしろ、旋盤モデルをどのように表現したかということに依存する。プーリ比、ギア比については generate and test を使えば自動的に求めることができるが、この時点ではその仕組みを取り入れていなかった）

しかしながら、FDL-II で得られた結果に、実験者の経験と知識から修正を加えていくことで、モデルの構築を進めていくことが可能であり、従来、設計者が試行錯誤して考えていた構造などを、簡単に計算結果として解答が得られるため、設計のコンセプトを固めることにも役に立つようである。（注：　本段落は高下氏のコメントに基づく）

Ⅵ　FDL-II の利用についての展望

FDL-II は、機械設計の基本設計段階の支援ツールとして開発されているが、コンセプトモデルの設計に使用できるかもしれない。また、現在、UNIX 上でしか稼働しないが、一般ユーザに使用してもらうためには、PC で稼働できることが望ましく思える。さらに、CAD ソフトとうまく連携ができるようになれば、活用範囲も広がるように思える。

Ⅶ　今回の実験で感じたこと

旋盤の要求仕様の与え方を考えると、現状では、全てが性能上の数値として与えられるわけではなく、一部は、ある種の感性に訴える言葉で表現される場合があり得る。一例として、「日立精機らしい削りを実現する機械」というような一文があるとすると、これを数値に置き換えるときに、切削性能数値でなく、モータ馬力であるとか、主軸外径寸法に反映させると言うことが実際には行われる。この場合、現状のやり方が正しいか、という議論もあり得るが、最終的な性能で表される要求仕様ではなくて、中間的な構造上の仕様（手段）に置き換え、さらに詳細化を図るという手順が採れるのがよいと思う。

別の言い方をすれば、当初は、目的を果たすための手段であったけれど、途中からは目的に姿を変えるということがある。FDL-II によれば、この場合でも、やり方で対処できるような気がする。

６．５　　６章のまとめ

本章では、FDL の設計への適用例を紹介した。**6.1** は、構造をアプリオリに与えた旋盤設計の例を示した。パラメトリック設計ないしコンフィギュレーション設計の第１レベルに相当する設計であり、構造は固定的に扱われる。**6.2** も旋盤設計ではあるが、構造は与えられず、設計の進展に従って確定する場合である。コンフィギュレーション設計の第２レベルに相当する。**6.3** はエレベータ設計である。コンフィギュレーション設計の第１レベルのものであるが、大規模で制約式が条件により変化するため、解探索の困難度がきわめて高いものである。FDL の設計解探索能力の高さを示すために取り組んだ。**6.4** は、工作機械メーカのベテラン技術者が実験評価した結果を第３者によりレポートしたものを載せた。FDL の評価は作成した対象モデルの構成方法にも依存するが、レポートの内容はおおむね妥当と思っている。

7章　結論

本章では、FDL の特徴を再整理した後、本研究によるパラメトリック／コンフィギュレーション設計に課題の解決に対する取り組みの結果を整理し、本研究で開発した技術を従来技術と比較する。最後に、工作機械メーカー、ストラスクライド大学 CAD センターによる評価を結果を受けて、今後の課題をまとめる。

7．1　言語の特徴

本研究では基本設計支援のための制約対象指向言語ＦＤＬを開発した。FDL の開発は FDL-I と FDL-II の２段階で行った。FDL-I はあらかじめ設計対象の構造が確定し、パラメータ値の決定のみを行うパラメトリック設計支援のみ適用可能であり、モデルの構造が動的に変化するよりダイナミックな設計への対応ができなかった。そこで,FDL-I の基本機能を継承しつつ、新たに設計モデルの動的構造変更機能を導入した FDL-II 言語を新たに開発した。

FDL-I の機能を整理すると次のようになる。

(1) オブジェクトのプログラム（メソッド）部分を制約により記述する対象指向言語である。
(2) FDL は単一属性制約、同一性制約、多属性制約の３種類の制約を扱う。 多属性制約の制約評価系は等式、不等式、述語、起動条件付き制約表現、if then ルール、generate and test 用値発生器などを扱うことができる。
(3) 多属性制約と関連づけられた関係データベース機能を提供する。
(4) オブジェクトごとに、その部品スロットとパラメータスロットの構成を反映した構造化スプレッドシート型インタフェイスとして提供する。

FDL-II には FDL-I の機能に加えて次の機能を導入した。

(5) オブジェクトの構成要素であるスロット（及びその中身）、制約、メソッド、ローカル述語を追加または削除するメタ操作。
(6) オブジェクト同士のアセンブリを容易に行うための構造融合メタ操作。
(7) すでにアセンブリされたオブジェクトを他のオブジェクトと交換するための構造交換メタ操作。
(8) 動的変更を受けたインスタンスオブジェクトとその元クラスオブジェクトの関係を管理するための was_a リンク。
(9) 動的変更を受けたインスタンスオブジェクトとそのオリジナルのクラスオブジェクトとの関係を管理するための was リンク。
(10) 環境（例えば、設計要求、製造環境、使用環境）に応じてオブジェクトを自律的に動的変更するための配置演算子。
(11) メタ操作をビジュアルに行うための、ビジュアルプログラミング環境。

これらのうち、構造融合、構造交換、was_a リンク、was リンクはオリジナルの技術で

あり、was_a リンク、was リンクについては日米の特許を取得した。

７．２　パラメトリック／コンフィギュレーション設計の課題への対応

3.6 で述べたように、本研究ではパラメトリック／コンフィギュレーション設計の課題を解決することを主要目的として設定した。課題としては次の項目を掲げた。

(1) パラメータ間の関係記述力が不十分

パラメータ間の関係表現を数式あるいは関数式のみに依存しているシステムが多いが、それだけでは表現能力の点で不十分で、より多様な表現が求められることが多い。

(2) 設計対象の記述性が低い

多くの設計システムで設計対象の記述性に問題がある。モジュール構造を持たない記述形式をとるものは、全体の構成、内容が把握しにくい。モジュール構造を持つものでも、既存言語を使っているものは記述性が高いといえないものが多い。

(3) 設計解探索能力の問題

数式のみで、かつ、静的に設計対象のパラメータ間関係が表現されている場合は、数理的手法を適用して（最適）設計解を効率的に求めることができる。しかし数式に加えて、ルール記述やテキストの扱いが含まれていたり、内容が動的に変化しうる場合は数理的方法を適用することは困難で、このような複雑な設計問題を解けるシステムは多くない。

(4) 設計対象のモジュール構造と組立

ルール集合、数式（制約）集合、スプレッドシートなど、モジュール構造を持たない設計対象表現では、プログラム全体の構造の理解、変更、再利用などが難しくなる。また、設計モデルがモジュール構造を持つものでも、モジュール間の関係が複雑かつ一様でない場合が多く、その場合モジュールの組立（関係付け）が煩わしい作業となる。

(5) 設計モデル構造の柔軟性

設計支援の自由度、設計解探索解の範囲を広げるためには、設計モデルを設計途中で動的に成長、変更、交換、削除することが可能である必要があるが、パラメトリック／コンフィギュレーション設計においてこのような能力をもつものは、非常に限られている。

これらの課題に対して、**3.6** で述べたように、本研究では、オブジェクト指向、制約解決、オブジェクトの動的構造変更のアプローチを採用して、解決を目指した。それぞれの課題に対する、本研究の取り組みの結果を整理すると、次のようになる。

(1) パラメータ間の関係記述力

ＦＤＬでは、数式、不等式、Prolog 述語、ルールにより制約を表現し、整数、有理数、ストリングのデータタイプを扱うことができるので、十分なパラメータ間の関係記述力を

持つことができた。

(2) 設計対象の記述性

言語表現形式は、設計者の立場から、設計対象の表現に適切な形式をとり、文法も明確に示した（付録 1、2）。従来も設計用言語の開発はいくつかあるが、インプリメンテーションに使った言語形式の影響を強く受けて、表現が理解しにくいなどの問題があった。FDL ではそのような問題を解決していると考えている。

記述性の評価に関しては、**6.3** で取り上げた VT 問題記述が参考になる。オリジナルの VT 記述は CLOS によると、約 500KB のコード量になったが、FDL-II で同等のプログラムを書くと、約 70KB で済む。また、記述そのものも、CLOS では LISP 独特の多重括弧の表現を使っているので読みやすいとは言い難いが、同等の記述を FDL では、自然な数式表現で行っているので、はるかに読みやすいと考える。

(3) 設計解探索能力

設計問題空間が静的に変化せず、しかも数式、不等式で表現されている場合は、数理的手法による設計解探索の効率的で、設計解の評価も比較的容易である。しかし、ストリングの扱いを含んだり、設計問題空間自体が動的に変化する場合は、数理的手法の適用は困難である。**6.2** の構造可変な旋盤設計,**6.3** のエレベータ設計は、設計問題空間が動的に変化する設計例題である。特に **6.2** は、設計モデルそのものが大幅に変更される場合で、汎用性の高い技術でこのような問題を扱える設計システムは、3 章で調査した範囲では見あたらない。また、**6.3** のエレベータ設計は条件に応じて使用される制約が変化する場合で、VT などのエキスパートシステムで対応可能であったが、FDL ではすべての解を比較的短時間で探索することができたことで、探索能力の高さを示すことができた。

(4) 設計対象のモジュール構造と組立

FDL の設計対象表現ではモジュール構造を持っているので、そうでないものに比較し、プログラム全体の構造の理解、変更、再利用が用意である。モジュールの組立は、モジュール間の複雑な関係を個別に記述していくのが一般的で、煩わしく間違いやすい作業であったが、FDL では構造融合とアセンブリ操作を導入することにより、電子回路部品の接続のように、組み立て作業を用意にした。

(5) 設計モデル構造の柔軟性

設計モデルの構造を設計途中で動的に変更することができるように、5 章で述べたように、構造融合、構造交換、スロット／制約／ローカル述語の追加、削除のメタ操作、及びに、was_a リンクや was リンクなどの動的変更を受けたオブジェクトの継承関係管理メカニズムを導入した。これらの結果、**6.2** で示したように、設計モデル構造を設計途中での動的変更が可能になった。

７．３　今後の課題

6.4 で述べたように、英国ストラスクライド大学及び国内工作機械メーカに FDL の評価を依頼した。その結果、動的構造変更機能、記述形式に関して高い評価が得られた反面、次のような課題も示された。

(1) メタ操作のためのエディタ型インタフェイス

メタ操作を用いた変更操作は入力テキスト量が多量になることが多いが、一般のコマンド入力と同様、入力を失敗した場合すべてをやり直さなければならず、使い易いとはいえない。モデルの動的変更操作のためのエディタ型のインタフェイスを提供する必要がある。

(2) メタ操作に対する undo 機能の導入。

値の入力、制約起動などはもとの状態に戻すのは容易であるが、構造融合、構造交換などのメタ操作をキャンセル（undo）するには、複雑な操作が必要であり、消去されてしまったデータ構造をとりもどす必要があるなど、困難な課題であるが、要望も強い。

(3) 設計ライブラリの充実。

本研究では旋盤設計、エレベータ設計を例題にとりあげ、その関連ライブラリは必要に応じて整備したが、汎用性を持たせるには、より広範囲にわたる設計ライブラリを充実させる必要がある。設計ライブラリの構成方法はそれ自体、大きな課題となる。オントロジー関連研究としての発展も視野に入れる必要がある。

(4) 既存 CAD とのリンク。

既存 CAD とのリンクについては、パラメトリック図面 CAD がすでに実用化されているので、モデルの構造が固定的な場合は比較的容易に実現可能である。しかし、FDL ではモデルの動的構造変更機能を有しているので、この機能を生かすには、図面 CAD においても動的構造変更機能を持たせる必要がある。これには、組み立て図面を部品図面のコンフィギュレーションにより構築する技術が必要となり、依然未解決な課題である。

これらの研究課題のうち（３）、（４）は特に大きな問題である。一方、今後の研究の方向として、進化型設計という研究課題[Imamura,00]を考えている。これは、既存の設備、大型機械のハードウエアの再利用、または設計データを再利用し、進化（機能、性能のポジティブな変化）させるための設計技術である。エレベータ、工作機械、工場設備などを対象に FDL を活用していきたい。

参考文献

[Anderl & Mendgen, 95] Anderl, R. and Mendgen, R., "Modelling with Constratins: Theoretical Foundation and Application", Computer Aided Design, 28, 3, pp.155-168 (1995)

[Andersson et al., 94] Andersson, J., Andersson, S., Boortz, K., Carlsson, M., Nilsson,H., Sjoeland, T. and Widen, J., " SICStus Prolog User's Manual", SICS Technical Report T93:01 (1994)

[Arbab, 91] Arbab, F., "Design Object Representation", Intelligent CAD, III, (Yoshikawa, H., Arbab, F., Tomiyama, T), Elsivier Science Publishers B.B. (North-Holland), pp.31-41 (1991)

[Araya & Mittal, 87] Araya, A. A. and Mittal S., "Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics", IJCAI'87, pp.552-558 (1987)

[Bennett, 97] Bennett III, F.H., "Programming Computers by Means of Natural Selection: Application to Analaog Circuit Synthesis", Proc. Int. Symp. on System Life, JSME, pp.41-50 (1997)

[Bijl, 91] Bijl, A., "Relations, Functions & Constraints without Prescriptions", Intelligent CAD III, NORTH-HOLLAND, pp.79-97 (1991)

[Bracewell et al., 95] Bracewell, R. H., Langdon, P. M., Oh, V. K., Chaplin, R. V., Li, M., Yan, X. T. and Sharpe J. E. E., "Integrated Platform for AI Support of Complex Design -(Part I & Part II):Rapid Development of Schemes from First Principles, Supporting theEmbodiment Process", Proceedings of Knowledge Intensive CAD-I (1995)

[Brachman & Schmolze,85] Brachman, R. J. and Schmolze, J. R., "An Overview of the KL-ONE Knowledge Representation System", Cognitive Science, 9, 2, pp.171-216 (1985)

[Brown & Chandrasekaran, 86] Brown, D.C. and Chandrasekaran, B., "Knowledge and Control for a Mechanical Design Expert System", IEEE COMPUTER, 19, 7, pp.92- (1986)

[Brown & Hwang, 93] Brown, D. R. and Hwang, K., "Solving Fixed Configuration Problems with Genetic Search, Research in Engineering Design", 5, pp.80-87 (1993)

[Chakrabarti & Bligh, 94] Chakrabarti, A. and Bligh, T. P., "An Approach to Functional Synthesis of Solutions in Mechancal Conceptual Design. Part I: Introduction and knowledge Representation, Research in Engineering Design", 6, pp.127-141 (1994)

[Clocksin & Mellish, 87] Clocksin, W. F. and Mellish, C. S., "Programming in Prolog, Third Edition",Springer-Verlag (1987)

[de Kleer & Brown,81] de Kleer, J. and Brown, J. S., "Mental Models of Physical Mechanisms and their Acquisitions", Cognitive Skills and Their Acquisition, Hillsdale, pp.285-309 (1981)

[Dixon & Simmons, 84] Dixon, J. R. and Simmons, M. K., "Expert Systems for Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture", Proceedings of ASME Computers in Engineering Conference (1984)

[Dixon, 86] Dixon, J. R., "Artificial Intelligence and Design: A Mechanical Engineering View", AAAI'86, pp.872-877 (1986)

[Donaldson & MacCallum, 94] Donaldson, I., MacCallum K. J., "The Role of Computational Prototypes in Conceptual Modelsfor Engineering Design", Artificial Intelligence in Design'94, Kluwer Academic Publishers, pp.3-20 (1994)

[Finger & Dixon, 89a] Finger, S. and Dixon, J. R., "A Review of Research in Mechanical Engineering Design, Part I", Research in Engineering Design, 1, 1, pp.51-67 (1989)

[Finger & Dixon, 89b] Finger, S. and Dixon, J. R., "A Review of Research in Mechanical Engineering Design, Part II", Research in Engineering Design, 1, 2, pp.121-137 (1989)

[Finger & Rinderle, 89] Finger, S. and Rinderle J. R., "A Transformational Approach to Mechanical Design Using a Bond Graph Grammar", ASME DTM'93, pp.107-116 (1989)

[Goldberg, 84] Goldberg, A., "SMALLTALK-80", ADDISON-WESLEY (1984)

[Gupta & Jakiela, 92] Gupta, R. and Jakiela, M. J., "Qualitative Simulation of Kinematic Pairs via Small-Scale Interference Detection", ASME DTM'92, pp.351-363 (1992)

[Harris, 86] Harris, D. R., "A Hybrid Structured Object and Constraint Representation Language", AAAI'86, pp.986-990 (1986)

[Hattori, 87] 服部幸英, " 生産過程と計算機の適用", CAD システムの機能と構成 (第1章)、日本機械学会編、技報堂出版 (1987)

[Higuchi & Nagasawa, 92] 樋口達治, 長沢勲, "機械要素の設計管理手法についての考察"、第10回設計シンポジウム講演論文集 pp.91-97 (1992)

[Hoover & Rinderle, 87] Hoover, S. P. and Rinderle, J. R., "A Synthesis Strategy for Mechanical Devices", Research in Engineering Design, 1, 3, pp.87-103 (1989)

[Imamura, 94] 今村聡, "工業設計支援用制約対象指向言語の開発 (第1報) -パラメトリック設計支援について-", 精密工学会誌, 60, 9, pp.1242-1246 (1994)

[Imamura, 97a] 今村聡, "エージェント群の協調作業による機械の組立分解プランニング", 日本機械学会論文集C編, 63, 612, pp.375-381 (1997)

[Imamura, 97b] 今村聡, "工業設計支援用制約対象指向言語の開発 (第2報) -パラメトリック設計とコンフィギュレーション設計の同時支援について-", 精密工学会誌, pp.1400-1404 (1997)

[Imamura, 00] Imamura, S., Masaki, H., Tokunaga, H. and Sawada, H., "Design for Product Evolution", Proc. 2000 Japan-USA Symposium on Flexible Automation (CDROM), (2000)

[Inoue et al.,88] 井上克己, 永井保夫, 藤井裕一, 今村聡, 小島俊雄, "工作機械の設計手法の解析 -旋盤の回転機能部品の設計", ICOT Technical Memorandum 494 (1988).

[IPA, 97] 情報処理振興事業協会、"新ソフトウエア構造化モデル・プロジェクト研究成果の普及体制に関する調査研究報告書" (1997)

[Ishiba, 95] Ishiba, A., Arai, Y., Akasaka, S., Haga, N. and Katsuta,K. "Constraint-based Elevator Design Support System", Proceedings of CAPE'95, CHAPMAN & HALL, pp.205-213 (1995)

[Joscowicz & Sacks, 91] Joscowicz, L. and Sacks, E. P., "Computational Kinematics", Artificial Intelligence, 51, pp.381-416 (1991)

[Kannapan & Marshek, 91] Kannapan, S. M. and Marshek, K. M., "Design Synthetic Reasoning: A Methodology for Mechanical Design", Research in Engineering Design, 2 (1991) 221.

[Karnopp et al., 90] Karnopp, D. C., Margolis, D. L. and Rosenberg, R. C., "System Dynamics: A Unified Approach", Second Edition, A Wiley-Interscience Publication (1990)

[Keuneke & Allemang, 89] Keuneke, A. and Allemang, D., "Exploring the No-Function-In-Structure Principle", Journal of Experimental & Theoretical Artificial Intelligence, pp.79-89 (1989)

[Kimura & Suzuki, 86] Kimura, F. and Suzuki, H., "Variational Product Design by Constraint Propagation and Satisfaction in Product Modeling", Annals of the CIRP, 35, 1, pp.75-78 (1986)

[Kimura et al., 87] Kimura, F., Suzuki, H., Ando, H., Sato, T. and Kinosada A., "Variational Geometry Based on Logical Constraints and its Applications to Product Modelling, Annals of the CIRP, 36, 1, pp.65-68 (1987)

[Kimura et al., 91] Kimura, F., Suzuki, H. and Tanaka, I., "A Pattern-Directed Design System for Manufacturing Assembly", Annals of the CIRP, 40, 1, pp.127-130 (1991)

[Kimura et al., 92] Kimura, F., Suzuki, H. and Takahashi, K., "Product Design Evaluation Based on Effect of Shape Errors for Part Assembly", Annals of the CIRP, 41, 1, pp.193-196 (1992)

[Kimura & Kojima, 95] 木村文彦、小島俊雄編、"製品モデル表現とその利用技術 STEP"、日本規格協会 (1995)

[Klein & Lu, 89] Klein, M. and Lu, S.C.-Y, "Conflict Resolution in Cooperative Design", Artificial Intelligence in Engineering, 4, 4 (1989) 168.

[Kojima et al., 87] Kojima, T., Imamura, S., Sekiguchi, H. and Inoue, K., "Use of Object Oriented Concept on Product Modeling System", Proc. of ICPE, pp.801-806 (1987)

[Kota, 91] Kota, S., "Generic Models for Designing Dwell Mechanisms: A Novel Kinematic Design of Stirling Engines as an Example", Journal of Mechanical Design, ASME, pp.446-450 (1991)

[Kota & Chiou, 92] Kota, S. and Chiou, S.J., "Conceptual Design of Mechanisms Based on Computational Synthesis and Simulation of Kinematic Building Blocks", Research in Engineering Design, 4, 2, pp.75-87 (1992)

[Lafue & Smith, 86] Lafue, G. M. E. and Smith, R. G., "Implementation of a Semantic Integrity Manager with a Knowledge Representation System", Expert Database System, The Benjamin/ Cummings Publishing Company, pp.333-350 (1986)

[Lai & Wilson, 87] Lai, K. and Wilson, W. R. D., "FDL - A Language for Functional Description and Rationalization in Mechanical Design", ASME Computer in Engineering, pp.87-94 (1987)

[MacCallum & Duffy, 87] MacCallum, K. J. and Duffy A., "An Expert System for Preliminary Numerical Design Modelling, Design Studies", 8, 4, pp.231-237 (1987)

[MacCallum, Yu, McGregor., 92] MacCallum, K. J., Yu, B. and McGregor, F. D., "A System for Supporting Design Configuration", Artificial Intelligence in Design'92, Kluwer Academic Publishers (1992)

[Maher, 85] Maher, M. L., "HI-RISE and Beyond: Directions for Expert Systems in Design", Computer Aided Design, pp.420-427 (1985)

[Marcus et al., 88] Marcus, S., Stout, J. and McDermott, J., "VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking", AI Magazine, pp.95-112 (1988)

[Mizoguchi, 99] 溝口理一郎,"オントロジー研究の基礎と応用", 人工知能学会誌, 14, 6, pp.977-988 (1999)

[Mittal et al., 86] Mittal, S. et al., "Pride: An Expert System for the Design of Paper Handling Systems", IEEE Computer, 119, 7, pp.102-114 (1986)

[Moss, 94] Moss, C., "Prolog++ The Power of Object-Oriented and Logic Programming", ADDISON-WESLEY (1994)

[Murakami & Gossard, 92] Murakami, T. and Gossard, D., "Mechanism Concept Retrieval by Behavioral SpecificationUsing Configuration Space", ASME DTM'92, pp.343-350 (1992)

[Murthy & Addanki, 87] Murthy, S. S. and Addanki, S., "PROMPT: An Innovative Design Tool", AAAI'87, pp.637-642 (1987)

[Nagae, 87]長江貞彦, "CAD/CAM の基本概念", コンピュータ設計製図 1,1 (1987)

[Nagai et al., 89] Nagai,Y., Taki,H., Terasaki, S., Yokoyama, T., Inoue, K., "Expert System Architecture for Design Tasks", ICOT TR-451 (1989).

[Nagasawa et al., 84] 長澤勲, 古川由美子, 荒牧重登, "論理プログラミングを基礎とした設計システム記述言語 ADL", 情報処理学会論文誌, 25, 4, pp.606-613 (1984)

[Nagasawa & Furukawa, 86] 長澤勲, 古川由美子, "拘束条件リダクション法を用いた機械設計計算支援システム", 情報処理学会論文誌, 27,1, pp.112-120 (1986)

[Neville & Joskowicz, 93] Neville, D. and Joskowicz, L., "A Representation Language for Mechanical Behavior", ASME DTM'93, pp.1-6 (1993)

[Neville & Weld, 93] Neville, D. and Weld, D. S., "Innovative Design as Systematic Search", AAAI'93, pp.737-742 (1993)

[Nicklaus et al., 87] Nicklaus, D. J., Tong, S. S. and Russo, C. J., "ENGINEOUS: A Knowledge Directed Computer Aided Design Shell", Artificial Intelligence Application Conference, IEEE, pp. 308-314 (1987)

[Paynter, 61] Paynter, H. M., "Analysis and Design of Engineering Systems", MIT Press, Cambridge, MA (1961)

[Persidis & Duffy, 91] Persidis, A. and Duffy, A. H. B., "Learning in Engineering Design", Intelligent CAD III, NORTH-HOLLAND, pp.251-272 (1991)

[Prade, 98] Prade, H.(editor), "Constraint Based Reasoning", Proc. of 13th European Conference on Artificial Intelligence, pp.207-266 (1998)

[Rao & Lu, 93] Rao, R. B. and Lu, S. C-Y., "Building Models to Support Synthesis in Early Stage Product Design", AAAI'93, pp.277-282 (1993)

[Rigopoulos & Oppenheim, 90] Rigopoulos, D. R. and Oppenheim, I. J., "Design Objects and their Representation: Buildings and Structures", Intelligent CAD II, Elsevier Science Publishers (North Holland), pp.251-264 (1990)

[Rinderle, 91] Rinderle, J. R., "Grammatical Approaches to Engineering Design, Part II: Melding Configuration and Parametric Design Using Attribute Grammars", Research in Engineering Design, 2, 3, pp.137-146 (1991)

[Sekiguchi et al., 86] 関口博，今村聡，小島俊雄，井上久仁子，"回転機能部品の部品展開の手法に関する研究(第２報)－組み立て分解手順の生成と規則化"，精密工学会誌，53，8，pp.1183-1188 (1986)

[Sata et al., 85] Sata, T., Kimura, F., Suzuki, H. and Fujita, T., "Designing Machine Assembly Structure Using Geometric Constraints in Product Modelling", Annals of the CIRP, 24, 1, pp.169-172 (1985)

[Sawada, 95] 澤田浩之、"数式処理的手法に基づく過少代数制約問題の数値解法"、情報処理学会論文誌，36，12，pp.2761-2770 (1995)

[Sawada, 99] 澤田浩之、"新たな条件式の導入による多変数連立代数方程式の解法"、情報処理学会論文誌，40，5，pp.2314-2324 (1999)

[Shimada et al., 89] Shimada, K., Numao, M., Masuda, H. and Kawabe, S."Constraint-based Object Oriented Description for Product Model", Proceedings of CAPE'89, North Holland, pp.95-106 (1989)

[Skalak et al., 98] Carlson-Slalak, S., White, D. and Teng, Y., "Using an Evolutionary Algorithm for Catalog Design", Research in Engineering Design, 10, pp.63-83 (1998)

[Sriram, 87] Sriram, D., "ALL-RISE: A Case Study in Constraint-Based Design", Artificail Intelligence in Engineering, 2, 4, pp.186-203 (1987)

[Steele Jr, 90] Steele Jr, G. L., "Common LISP Object System", Common LISP Second Edition, Digital Press, pp.770-864 (1990)

[Struss, 87] Struss, P., "Multiple Representation of Structure and Function", Working Conference on Expert Systems in CAD, IFIP WG5.2 (1987)

[Suh, 88] Suh, N. P., "The Principles of Design", Oxford University Press, Oxford, UK (1988)

[Sussman & Steel, 80] Sussman, G. J. and Steel Jr, G. L., "CONSTRAINT - A Language for Expressing Almost Hierarchical Descriptions", Artificial Intelligence, 14, 1, pp.1-39 (1980)

[Tomiyama et al., 89] Tomiyama, T., Kiriyama, T., Takeda, H. and Xue, D., "Metamodel: A key to Intelligent CAD Systems, Research in Engineering Design", 1, 1, pp.19-34 (1989)

[Tomiyama et al., 93] Tomiyama, T., Umeda, Y. and Yoshikawa, H., "A CAD for Functional Design", Annals of the CIRP, 42, 1, pp.143-146 (1993)

[Tweed & Bijl, 90] Tweed, C. and Bijl, A., "MOLE; A Reasonable Logic for Design ?", Intelligent CAD System 2; Implementation Issues, Springer-Verlag, Berlin (1990)
[Uchida, 87] Uchida, S., "ESP Guide", ICOT Technical Memorandum 0388 (1987)

[Ulrich & Seering, 89] Ulrich, K. T. and Seering, W. P., "Synthesis of Schematic Descriptions in Mechanical Design", Research in Engineering Design, 1, 1, pp. 3-18 (1989)

[Umeda et al., 90] Umeda, Y., Takeda, H., Tomiyama, T. and Yoshikawa, H., "Function, Behaviour and Structure", Proceedings of AI in Engineering, Boston, pp.177-193 (1990)

[Ward & Seering, 89] Ward, A. C. and Seering, W., "Quantitative Inference in a Mechanical Design Compiler", MIT AI Memo No.1062 (1989)

[Welch & Dixon, 92] Welch, R. V. and Dixon, J. R., "Representing Function, Behavior and Structure during Conceptual Design", ASME DTM'92, pp.11-18 (1992)

[Winsor & MacCallum, 94] Winsor, J. and MacCallum, K., " A Review of Functionality Modelling in Design", The Knowledge Engineering Review, 9, 2, pp.163-199 (1994)

[Wolfram, 92] Wolfram S., "Mathematica", Addison Wesley, Japan (1992)

[Yokoyama, 89] Yokoyama T., "An Object-Oriented and Constraint-Based Knowledge Representation System for Design Object Modeling", ICOT Technical Memorandum 809 (1989)

[Yokoyama et. al., 97] 横山正明, 遠藤亨, 車周憲, "知識ベースＣＡＤ総論", 知識ベースCAD (第１章), コロナ社 (1997)

[Yoshikawa, 79] 吉川弘之, "一般設計学序説", 精密機械, 43, 1, pp.906-912 (1979)

[Yoshikawa, 81] 吉川弘之, "一般設計過程", 精密機械, 47, 4, pp.906-912 (1981)

発表目録

（1）学会誌

Imamura, S., Kojima, T., Inoue, K. and Sekiguchi, H., "A Knowledge Based System to Handle Assembly Drawing Data", Bulletin of JSPE, 21, 2, pp.146-147 (1986)

関口博, 今村聡, 小島俊雄, 井上久仁子, "回転機能部品の部品展開の手法に関する研究(第2報)－組み立て分解手順の生成と規則化", 精密工学会誌, 53, 8, pp.1183-1188 (1986)

今村聡, 小島俊雄, 関口博, 井上久仁子, "自動寸法指定問題に関する研究", 精密工学会誌, 53, 11, pp.1713-1718 (1986)

関口博, 今村聡, 小島俊雄, 井上久仁子, 小池, "回転機能部品の部品展開手法に関する研究(第3報) －組立構造に着目した部品公差の設定方法－", 精密工学会誌, 54, 9, pp.1782-1787 (1988)

Imamura, S., Horie, Y., Sakakibara, H. and Enomoto, S., "Goal-Oriented Machine Assembly/Disassembly Planning by Cooperative Agents", International Journal of The Japan Society for Precision Engineering, 27, 4, pp.387-388 (1993)

今村聡, "工業設計支援用制約対象指向言語の開発（第1報）-パラメトリック設計支援について-", 精密工学会誌, 60, 9, pp.1242-1246 (1994)

Imamura, S., "Structure Merging as a Key Operation to Assemble Parametric Design Models", International Journal of the Japan Society for Precision Engineering, pp.271-272 (1995)

今村聡, "エージェント群の協調作業による機械の組立分解プランニング", 日本機械学会論文集C編, 63, 612, pp.375-381 (1997)

今村聡, "工業設計支援用制約対象指向言語の開発（第2報）-パラメトリック設計とコンフィギュレーション設計の同時支援について-", 精密工学会誌, pp.1400-1404 (1997)

Imamura, S., "Machine Assembly/Disassembly Planning by Cooperatove Agents", International Journal of the Japan Society of Mechanical Engineers, Vol.41, No.4, pp.947-952 (1998)

（2）国際会議

Imamura, S., Kojima, T., Sekiguchi, H. and Inoue, K., "A Study on the Object Oriented Product Model -Representation of Geometry and Dimension", Annals of the CIRP, 37, 1, pp.127-130 (1988)

Imamura, S. and Kojima, T., "Public Knowledge Base for Intelligent Machine Design Systems", Preprints of the Third IFIP WG5.2 Workshop on Intelligent CAD, pp.174-179 (1989)

Masaki, H., Nomura, N., Imamura, S. and Kojima, T., "The Object Oriented Modeling to Grasp Form Features", Proceedings of CAPE'89, North Holland, pp.87-94 (1989)

Nomura, N., Kojima, T., Masaki, H. and Imamura, S., "A Method to Exactly Determine Part Geometry from a Dimensioned Rough Drawing", Proceedings of CAPE'89, North Holland, pp.123-130 (1989)

Imamura, S., Arimoto, M. and Enomoto, S., "Adaptive Assembly/ Disassembly Planning by A Society of Agents", Proceedings of Japan-U.S.A. Symposium on Flexible Automation, pp.617-624 (1994)

Kojima, T., Imamura, S., Sekiguchi, H. and Inoue, K., "Use of Object Oriented Concept on Product Modeling System", Proc. of ICPE, pp.801-806 (1987)

Imamura, S., Nomura, N., Masaki, H. and Kojima, T., "An Attempt to Maintain the Consistency of Product Model by Introducing Constraint Programming", Proceedings of the Third International Conference on Computer Aided Production Engineering, pp.446-453 (1988)

Imamura, S., "FDL: A Constraint based Object Oriented Language for Functional Design", Human Aspects in Computer Integrated Manufacturing, IFIP, Elsevier Science Publishers, pp.227-236 (1992)

Imamura, S., "Towards More Flexible Parametric Design Support by Introducing a Transformable Object Oriented Language", Proceedings of CAPE'95, CHAPMAN & HALL, pp.548-555 (1995)

Imamura, S. and Sawada, H., "An Object Oriented Language Introducing Biogenetic Algorithms", Proceedings of International Symposium on System Life, JSPE, pp.191-198 (1997)

Masaki, H. and Imamura, S., "CAD based Multi Agent Machine Assembly/Disassembly Planning", Proceedings of the 11th International Conference of Applications of Prolog, pp.130-135 (1998)

Imamura, S., Masaki, H., Tokunaga, H. and Sawada, H., "Design for Product Evolution", Proceedings of 2000 Japan-USA Symposium on Flexible Automation (CDROM) (2000)

（3）解説記事など

小島俊雄，今村聡，"ＦＡにおけるＣＡＤシステム技術"，計測と制御，26，7，pp.569-574（1987）

今村聡, "機械設計における AI 技術の活用と実用化の課題", 機械設計, 33, 1, pp.23-32, 日刊工業新聞社（1989）

今村聡，"対象指向とインテリジェントＣＡＤ"，インテリジェントＣＡＤ(下)第３章, pp.67-87, 朝倉書店（1991）

今村聡, "機械設計", ｂｉｔ別冊 協調プログラミング例題集, 第７章, pp.61-69, 共立出版（1996）

井上克己, 永井保夫, 藤井裕一, 今村聡, 小島俊雄, "工作機械の設計手法の解析 －旋盤の回転機能部品の設計", ICOT Technical Memorandum 494（1988）.

今村聡, 榊原寿, 堀江優美子, 榎本進, "ロボットハンドの最適操作列を導くエージェント群の協調動作方式", 機械技術研究所所報, 47, 6, pp.227-236（1993）

今村聡, 波潟誠, 清水透, 佐野利夫, "軸対称製品を対象とした冷間鍛造工程設計エキスパートシステム", 機械技術研究所所報, 51, 4, pp.126-131（1997）

今村聡,”環境負荷低減を目指す設計手法”, 高圧ガス, 36, 2 , pp.148 -149（1999）

今村聡, 徳永仁史,”機械設計におけるモジュール化設計と再構成”, 精密工学会誌, 66, 7, pp.1039-1042（2000）

今村聡,”進化型設計技術　－製品の再生を目指して－”，産業と環境，29，10，pp.31-34, オートメレビュー社（2000）

謝辞

本文でも述べた通り、本研究は産業科学技術研究開発制度「新ソフトウエア構造化モデル」プロジェクトの中で行った。本プロジェクトは情報科学のプロジェクトであり、機械技術研究所からの参加は茨の道であったが、小島俊雄部長（当時、情報工学課長）、大山尚武所長（当時、企画室長）のご尽力で参加への道が開け、本研究を遂行する研究環境を得ることができた。まず、両氏に感謝の意を表したい。研究プロジェクト遂行中は、大見孝吉国際研究協力官、小鍛治繁極限技術部長、松田浄史前物理情報部長からプロジェクト遂行のための直接的な支援を受けた。また中澤克紀前所長からは、本研究への強い関心を示していただき、励みとなった。井上久仁子元情報工学課長からは、時折に論文執筆へ向けて励ましを受けた。また、システム工学研究室、数理工学研究室の同僚にも多々お世話になった。以上の方々に、感謝の意を表したい。

「新ソフトウエア構造化モデル」プロジェクトの遂行に当たっては、電子技術総合研究所、情報処理振興事業協会の多くの関係者に大変お世話になった。評価委員会での事前準備、海外調査派遣など、懐の深い対応に感謝する。Prof. Kenneth MacCallum（現英国グラスゴー・ベルカレッジ学長、前ストラスクライド大学 DMEM 学科長）とは、良好な国際研究協力関係を持つことができ、本研究の視野の拡大に大変役立った。

木村文彦教授からは、本論文執筆に対して懇切丁寧なご指導をいただき、完成にこぎつけることができた。心から感謝申し上げる。

最後に、機械技術研究所へ就職することができ、本研究に従事できたのも、妻と両親の理解、支援の賜であり、この紙面を借りて感謝する。