TM-0591

設計対象記述のための知識表現システム FREEDOMの提案

横山孝典

August, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

設計対象記述のための知識表現システム FREEDOMの提案

横山 孝典

新世代コンピュータ技術開発機構

設計型問題における対象モデル記述のための知識表現システム FREEDOM を提案する。ここでは設計を一定の制約条件のもとで要求仕様を満足する対象モデルを生成する問題ととらえ，動的機能を重視した知識表現を実現する。

FREEDOM はオブジェクト指向を基本に，対象の属性や構造に関する制約の記述を可能にし，動的変更可能なクラス・インスタンス関係，"is-a" と "includes" 関係によるクラス階層等，設計対象の記述に適した知識表現機能を提供する。

そしてインスタンスの属性値を常に制約充足状態に保つとともに，制約充足が不可能な場合それが可能なクラスへ変更する機能や，オブジェクト間の関係を利用した詳細化支援機能を提供することにより，効率よい解の生成を可能としている。

FREEDOM : A Knowledge Representation System for Design Object Modeling

Takanori YOKOYAMA

Institute for New Generation Computer Technology

1-4-28, Mita, Minatoku, Tokyo, 108, Japan

This paper presents a system for representing design object models, FREEDOM. FREEDOM provides knowledge representation facillities based on object-oriented paradigm, descriptions of constraints on attributes and structures of objects, dynamically-changeable class-instance relation, class hierarchy with "is-a" and "includes" relations.

It is possible to solve the design problems effectively by the constraint satisfaction mechanism, which determines the values of attributes and the classes of instances, and the refinement support functions using the relations between objects.

1．はじめに

現在我々は設計型問題向きの知識システム構築技術に関する研究を行っている。ここではその一環として開発中の対象モデル表現システム FREEDOM (a Framework for REpresEnting Design Object Model) について報告する。

最近の知識システムでは問題の対象に関する知識を問題の解決法に関する知識と分離し，対象モデルとして表現することが重要視されている。この方式は，対象モデルを知識ベースの中心とするもので，知識のモジュール化，汎用化を図るとともに，システムの能力向上に有力だと言われている[上野85][Ohsuga85]。

従来，対象モデルの表現には，対象の属性や構造を表現しやすいフレームやオブジェクト指向言語が多く用いられてきた。両者はよく似た概念であり，設計対象をその構成要素ごとにフレーム，あるいはオブジェクトとしてモジュール化して扱いやすい形で表現できる。

ところで設計型問題は，様々な制約条件のもとで与えられた要求仕様を満足する対象モデルを生成することと見なせる。従って，単に設計対象に関する知識を静的に表現するだけでなく，モデルの取捨選択，修正，詳細化等の処理が容易な動的機能が重視される。

従来の設計システムでは対象モデルは単なる静的なデータ構造に過ぎず，その解釈や一切の操作は外部の設計知識を用いて実行されることが多かった。しかしこの方式は設計方法に関する知識のみに重点が置かれ，設計対象自身が本来持っている性質に関する知識が有効に利用されない可能性がある。

FREEDOM における知識表現はオブジェクト指向を基本に，制約の記述を可能とし，さらにこの制約充足のためにオブジェクトが動的に変化する機能を付加することにより，詳細化に適した形式となっている[横山88]。これにより対象モデルは単なるデータ構造ではなく，問題解決を積極的に支援する機能を提供でき，知識システムの高度化に役立つものとなる。

以下ではまず設計型問題における対象モデル表現に必要な基本機能と FREEDOM の位置づけについて述べる。次に FREEDOM における知識表現と設計過程支援機能について論じ，現在開発中のプロトタイプ FREEDOM/S0 (FREEDOM with Sequential inference mechanism, version 0) の機能と実現方式について説明する。そして関連する研究について触れ，最後に今後の課題について述べる。

2．対象モデルの表現

2.1 設計型問題における対象モデル

一般に対象モデルは解析型問題では固定的であるのに対して，合成型問題では問題解決過程で動的に変化することがある。特に設計は，要求仕様を満足する対象モデルを生成する問題と定式化でき，モデルの取捨選択，修正，詳細化等が設計過程での処理の中心である。

ところで設計型問題における対象モデルではあらかじめ知られている設計対象に関する基本的，一般的な知識を表現するものと，設計過程で生成される具体的な設計解（またはその中間結果）を表現するものとを区別する必要がある。前者は設計対象の構成や，対象が持つべき属性や構成要素の存在，また，それらが満たすべき制約などであり，物理法則や実験式等も含まれる。一方後者は具体的な属性値を持つデータの集合である。ここでは両者を区別するため，前者をテンプレートモデル，後者をインスタンスモデルと呼ぶことにする。

例えば機械の基本的な構成や属性値が満たすべき制約条件の記述は前者で，材質や寸法等が決定された具体的なデータが後者である。電子回路では回路の基本的構成やそこに存在する制約を記述したものが前者，具体的な部品の特性や回路定数を表現するのが後者である。

この観点で見ると，設計過程とは設計方法に関する知識を持つ設計手続きが，テンプレートモデルに記述された知識を利用して，要求仕様を満足するインスタンスモデルを生成することと見なせる。あるいは，テンプレートモデルに記述された制約と要求仕様という形で与えられた制約をともに満足するインスタンスモデルを設計手続きに従って生成することであると言ってもよい。

ところが従来の設計エキスパートシステムではインスタンスモデルのみを扱い，テンプレートモデルの表現をおろそかにしがちであった。これは設計方法に関する知識のみを重視し，設計対象に関する知識を軽視していたり，テンプレートモデルに記述すべき設計対象に関する知識がモデラのモデル操作手続きや，設計手続き中に埋め込まれてしまっていたためである。しかしこれでは対象に関する知識と設計方法に関する知識が未分化になり，システムの柔軟性を損ない，設計システムの高度化や，一般性のあるシステム構築法の確立を阻害する可能性がある。

そこでここでは効率的な問題解決を実現するためには設計対象に関する知識を有効に利用する必要があると考え，対象モデルの表現能力の向上と，それに基づく問題解決支援機能の強化を図ることにする。

2.2 FREEDOM の位置づけ

FREEDOM はテンプレートモデルの記述と要求仕様を満足するインスタンスモデルの生成を効率よく支援するためのシステムである。図1に FREEDOM を用いた設計システムの

基本的な構成を示す。対象モデルすなわち設計対象に関する知識は FREEDOM 上に記述される。これに対し設計解を求めるための設計手続きを実行する部分はその外部に存在する。一般に知識システムでは設計手続きは設計方法に関する知識ベースとなる。

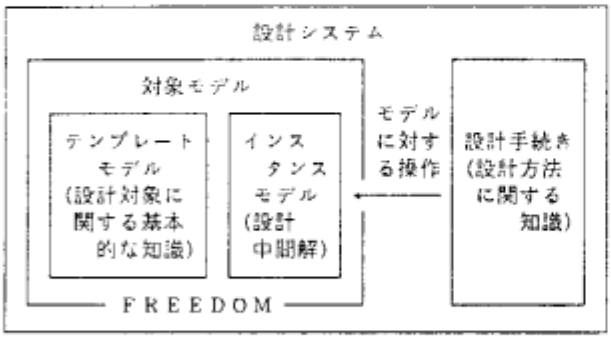

図1 FREEDOMを用いた設計システムの基本構成

設計過程では，設計手続きが FREEDOM 上のインスタンスモデルに対し，構成要素や属性値の変更，構造の修正等の操作や，制約の追加削除等の処理を実行する。ここで，FREEDOM はテンプレートモデルに記述された知識や，設計手続きがインスタンスモデルに付加した制約を利用して，インスタンスモデルに対する制約充足処理や詳細化処理を実行し，効率のよい設計解の生成を支援する。

FREEDOM は制約充足機構や詳細化支援機構等の機能を有するものの，それ自身が設計過程を実行するわけではない。あるいは FREEDOM の中心的な機能は対象モデルが常に制約を満足する状態を保つことであると言ってもよい。従って設計方法に関する知識の記述や，実際に設計手順を実行する機構は外部に用意する必要がある。この点で FREEDOM は TMS [Doyle79]に似た形態のシステムである。

しかし，要求仕様と設計対象に関する知識から直接設計解を導くことが可能な簡単な問題では，ほぼ FREEDOM の機能のみで設計を行うことができる。また，FREEDOM に問題に合ったユーザインタフェースを追加することにより，スケッチパッド[Sutherland63]のような対話的な設計支援システムを容易に実現できる。

3．FREEDOM における知識表現

3.1 オブジェクト指向への制約の導入

対象モデルは全体の構造やそれを構成する部品の属性，属性間の関係等，問題解決上必要な対象に関する知識をわかりやすい形で表現する必要がある。そこで対象を部品毎にオブジェクトとして自然な形で記述できるオブジェクト指向による表現を採用する。

しかし一般のオブジェクト指向言語ではオブジェクトの属性や動作の記述は容易だが，属性間，オブジェクト間の関係をうまく表現する手段がない。このため従来，関係をメソッドを用いて手続き的に記述したり，関係を表現するためのオブジェクトを導入する等の方法が用いられてきたが，必ずしも適切な表現とは言い難い。

そこでここでは対象モデルが満たすべき属性値間の関係や，構造的関係を制約条件として宣言的に記述できる機能を追加する。また，数値的制約と記号的制約を同じ形式で表現するため，述語形式の記述を基本とする。

ところでオブジェクトの構造の表現においては部分・全体関係，いわゆる“part-of”関係が重要であるが，この関係には部分が全体にとって必要不可欠な場合と，必ずしも必要でない場合がある。前者の例は四角形における４つの辺で，後者の例は本棚におけるその中の本である。ここでは前者を全体に対する「構成要素」，その関係を“consists-of”と呼んで，オブジェクトの構造上の制約として扱うことにする。

なお，制約条件は静的に与えられるものとは限らず，問題解決の過程で動的に生成されるものもある。このためインスタンスに動的に制約を追加，削除する機能も提供する。

3.2 柔軟なクラス・インスタンス関係

設計型問題における対象モデルをテンプレートモデルとインスタンスモデルに分けることは既に述べたが，前者はオブジェクト指向言語における「クラス」，後者は「インスタンス」に対応させることができる。

設計過程では要求仕様を満足する部品の種類を見つけ，その属性値を決定する必要がある。このことは制約を満足するクラスを探索し，インスタンスの属性値を決定することである。ところが，実際の設計では両者は並行して行われるため，インスタンスの属性値を決定した後，別のクラスへ変更したいということが頻繁に発生する。

しかし一般のオブジェクト指向言語ではインスタンスが属するクラスが固定化されているため，修正の度に古いインスタンスを消去し，新しいクラスのインスタンスを生成し，両者に共通な構成要素や属性値をコピーする必要がある。しかしこのような操作を手続き的に記述するのは煩わしいうえ，一般に修正箇所はインスタンスの一部に限られるから非効率でもある。

そこでここでは，クラス・インスタンス関係，いわゆる“instance-of”関係を従来のように固定的なものとせず，問題解決過程で動的に変更可能とする方式を提案する。こ

れにより，インスタンスの属性値を決定しながら，要求仕様を満足するクラスを探索することが可能になる。

ただし，クラスの動的変更を無制限に許すことはシステムの効率を低下させる可能性があるうえ，動的変更が有効なのはインスタンスの一部を修正する場合と考えられるからその必要性も少ない。

一般に設計はトップダウンになされ，このことは対象モデルを抽象的なレベルから具体的なレベルへと詳細化（具体化）していくことと見なせる。そこで効率的な詳細化処理を実現するには，「抽象－具体」を表わすクラス階層，いわゆる“is-a”関係を利用することが考えられ，クラスの動的変更はこの詳細化処理において有効になる。

従ってここでは“is-a”の階層に沿った形でのみクラスの動的変更を可能とする。

3.3 “is-a”と“includes”関係によるクラス階層

前節ではクラスの探索に“is-a”関係を利用することを述べた。ところが従来の多重継承を許すオブジェクト指向言語で書かれたプログラムを見ると，機能の包含関係を表わすためにクラスの継承機能を利用することが多く，必ずしも意味的な“is-a”関係の表現とはなっていないため，そのまま詳細化処理に利用することはできない。

例えばある製品の筐体に用いる金属板を決定する問題を考える。この時クラス「鉄板」を，クラス「鉄」とクラスと「板」というクラスを多重継承して定義したとする。この場合，「鉄板」is-a「板」，は“is-a”階層として意味があるが，「鉄板」is-a 「鉄」はあまり意味をなさない。

そこで本方式では“is a”関係を意味的に同一次元に属するクラス間のみに定義可能とし，多重継承は許さないことにした。これにより“is-a”関係は単純な木構造となる。従ってクラスの動的変更は“is-a”の同一木構造に属するクラス間でのみ許すことになる。

ただし機能の包含関係という意味での多重継承機能も知識のモジュール化という点で重要である。そこで“is-a”関係とは別に，クラス間の機能の包含関係を表わすものとして“includes”関係を定義可能とした。“is-a”関係と“includes”関係は詳細化処理における解釈が異なるものの，継承機能の点では特に差はない。ただし，継承の優先順位は“is-a”関係の方が高い。

ところで一般に設計で扱う対象は様々な形状，材料，機能を有するものがあり，部品の種類は膨大になる。このためクラスの継承機能を利用したとしても相当数のクラスを定義する必要がある。そこでここでは“includes”関係を利用して，効率的なクラス構成を実現する。すなわち，包含するクラスは指定したクラスのみに限定されるのではなく，そのサブクラスをも許すことにする。そして特定の包含するクラスのみに着目して詳細化を行うことができるようにする。これによりクラス構成をすっきりしたものにできる。

本方式による金属板のクラス階層の例を図2に示す。ここで金属板に関するクラス構成は，「金属板」is-a「板」，「金属板」includes「金属」であり，金属については「鉄」is-a「金属」，「アルミ」is-a「金属」が定義されている。金属板に使用する金属の種類を決定する問題では「金属板」の包含するクラス「金属」にのみ着目し，これを“is-a”階層を下にたどって詳細化し「鉄」あるいは「アルミ」とすればよい。また“includes”関係を用いれば，「鉄板」，「アルミ板」等金属の種類全ての金属板のクラスを定義する必要がなくなる。

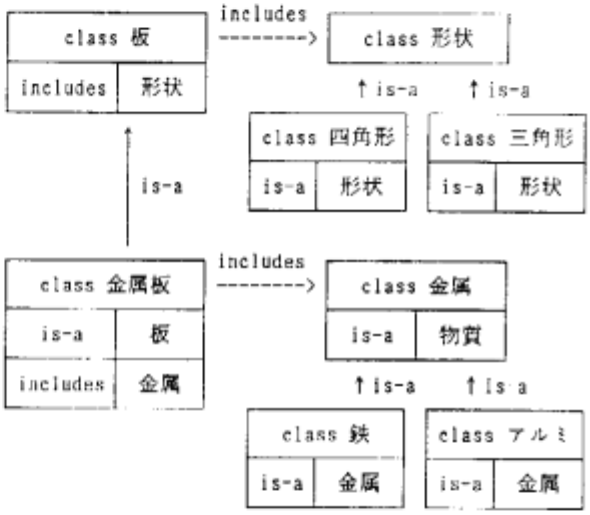

図2 “is-a”と“includes”関係によるクラス階層

ところで従来複数の機能を持つオブジェクトを表現する場合，多重継承を利用する方法と“part-of”（部分－全体）関係を用いる方法とが考えられ，いずれがよいかの議論があるが[Stefik86]，“includes”関係を用いれば自然な表現が可能となる場合が少なくない。

なおクラス間の“is-a”関係の適，不適は対象とする問題によって異なることがある。例えば先程の筐体用の金属板を決定する問題では，「鉄板」is-a「板」が意味をもち，「鉄板」is-a「鉄」は意味をもたなかったが，「鉄」という材料の用途を考える問題では「鉄板」is-a「鉄」が意味をもつ可能性がある。そこで“is-a”のリンクにラベルを付け，ラベルの異なる複数の“is-a”関係を許し，問題によって最適な“is-a”リンクを選択する方式を検討中であ

るが，その詳細についてはここでは触れない。

4．FREEDOM の設計支援機能

4.1 制約充足状態の保持

前述のように FREEDOM は，クラスで定義されたり外部から付加された制約をインスタンスが常に満足する状態を保つ機能を提供する。すなわち，インスタンスの属性値の設定や変更を行った場合には制約充足処理を実行し，関連する属性値を修正するなどの制約充足処理を実行する。

設計型問題では与えられた要求仕様をも制約と見なすことができ，これと対象自身の持つ制約をともに満足する解を求めることが設計の目的である。従ってインスタンスが与えられた制約を常に満足する状態を保つことは設計過程の強力な支援となり，特に属性値の決定において非常に有力である。

一般に対象モデルの属性値は数値または記号で表現され，これらは制約式を解くことにより求めることができる。例えば重量，密度，面積，厚さという４つの属性間に

重量 ＝ 密度 ＊ 面積 ＊ 厚さ

という制約が存在する場合，これらのうちいづれか３つが与えられれば残りのひとつを決定できる。

4.2 制約充足によるクラスの変更

クラスの“is-a”の階層を利用した詳細化（具体化）については既に触れたが，これは要求仕様を満足する部品の種類，すなわちクラスの探索処理である。FREEDOM ではインスタンスの属するクラスを動的に変更できるため，自然な詳細化処理が可能である。

設計過程では抽象的なクラスのインスタンスから出発し，現実に存在する対象（部品）を表わすクラスに達するまで詳細化が繰り返される。ただし，要求仕様を満足しないクラスに達した場合にはバックトラックする等，試行錯誤的な探索がなされるのが普通である。

ところが，クラスを特定の種類のインスタンスに関する制約を記述したものととらえれば，インスタンスの属するクラスの変更を制約充足処理のひとつとして実現することができる。すなわち，詳細化処理の過程で，インスタンスの属性値の変更や，構成要素の追加，削除を行った場合，それが属するクラス内での制約充足処理が不可能であれば，自動的に制約充足が可能なクラスに変更する機能を提供する。この機能により，インスタンスに対する操作のみで，制約を満たすクラスの探索が可能となる。

図３はクラスの“is-a”階層を示したものである。ここでクラス０，１，２は抽象的なオブジェクトを，クラス３，４，５，６は実際に世の中に存在するオブジェクトを表わすとする。また Dn はインスタンスの持つべき構成要素と属性の宣言で，Cn はインスタンスに関する制約の集合であるとする。一般にひとつのインスタンスが持つ性質はそれが属するクラス及びその上位のクラスに記述されたものの和集合となる。例えばクラス３に属するインスタンスは，D0 ＋ D1 ＋ D3 に記述された構成要素と属性を持ち，その制約は C0 ＋ C1 ＋ C3 となる。

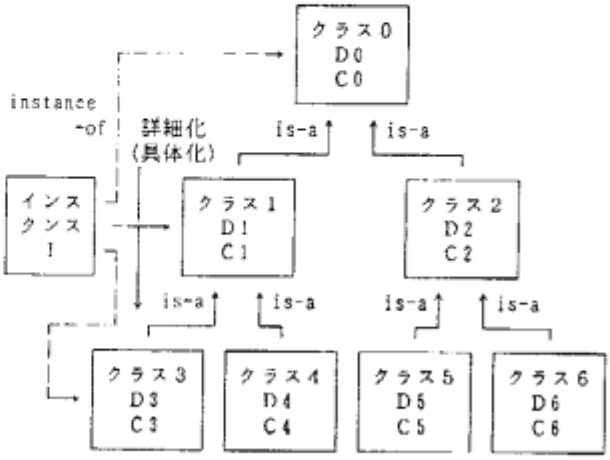

図３ クラスの“is-a”階層を利用した詳細化

今，最上位のクラス０のインスタンスＩを生成し，その構成要素や属性値を決定しながら，クラス１まで詳細化が進んだとする。この時インスタンスＩは制約 C0 ＋ C1 を満足しているが，ここでさらに詳細化を進める。まず，Ｉの属するクラスをクラス３に変更し，制約 C0 ＋ C1 ＋ C3 の充足を試みる。もしこれが可能であればＩのクラスはクラス３となり，詳細化は終了する。もし失敗すればバックトラックが発生し，次のクラス４に変更し，制約 C0 ＋ C1 ＋ C4 の充足を試みる。成功すればＩはクラス４のインスタンスとして終了し，再び失敗であれば詳細化処理自体が失敗したものとみなし，Ｉは最初の状態，すなわちクラス１のままである。

例えば金属板を決定する問題で，クラス階層が図４であったとする。ここで，最初インスタンス「金属板Ａ」は鉄板を使用することを想定していたが，面積と厚さを決定した時点で，重量を一定値以下とする制約により金属板Ａの属するクラスは「鉄板」から「アルミ板」に変更される。

このように FREEDOM では単に属性値のみでなく，要求仕様を満足する部品の種類の決定をも，処理の手続きを明示的に記述することなく，制約充足処理により行うことを可能としている点に大きな特徴がある。

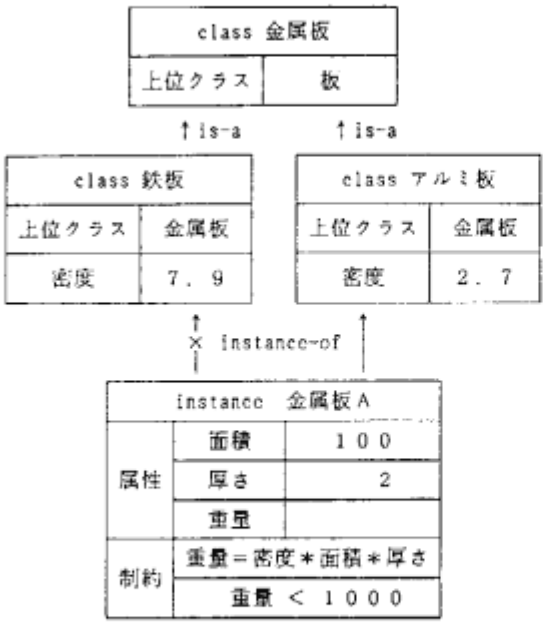

図4 制約充足によるクラスの変更

ところでクラスがオブジェクトの制約を記述したものであれば，クラス階層は制約の優先順位の階層を表わすものととらえることができる。優先順位は上位クラスほど高く，図3の例では

C0 ＞ C1 ＞ C3

である。従って詳細化処理は，優先順位の高い制約から順に充足を図りながら段階的に制約のきつい状態に向かってインスタンスを遷移させていく作業であると言うことができる。

4.3 トップダウン設計の支援

設計過程では，「抽象」から「具体」へという詳細化（具体化）の他，「全体」から「部分」へという形の詳細化（細分化）も非常に重要である。FREEDOM では前者は "is-a" 関係を利用して実行することは既に述べたが，後者については "includes" 及び "consists-of" 関係を利用する。

"is-a" の階層はシステムの構造や，部品の種類を選択する場合に，"includes" の階層はひとつの対象を複数の異なる視点で眺めて，それぞれ別々に詳細化を進める場合に，"consists-of" は対象をその構成要素に分解し，それぞれ独立に詳細化を進める場合に利用する。

ところで "includes" で宣言されたクラスはそのサブクラスをも許すことは既に述べたが，FREEDOM では "consists-of" で宣言された構成要素に関しても同様に指定されたクラスのみでなくそのサブクラスも許す。従って FREEDOM における詳細化の手順は，まず，抽象的なクラスのインスタンスを生成し，これを "is-a" 階層に沿って具体化するとともに "includes" と "consists-of" 関係を利用して部分に分解し，分解したものをさらに具体化するという処理の繰り返しとなる。

図5にギアシステムを例に3種の関係を利用したトップダウン設計のイメージを示す。最上位のクラス「ギアシステム」のインスタンスを生成し，クラスの "is-a" 階層に沿って要求仕様を満足する構造をもつギアシステムに具体化する。この場合「2段ギア」を選択したものとする。ところが2段ギアシステムは，「シャフト」，「ギア1」，「ギア2」等の部品から成立することが "consists-of" 関係で定義されているため，次にこれらの部品毎に詳細化を実行する。そしてさらに例えばギア1を "includes" 関係を利用して「形状・大きさ」と「材料」のそれぞれの観点から具体化する。材料の場合，金属の種類を "is-a" の階層を利用して決定する。

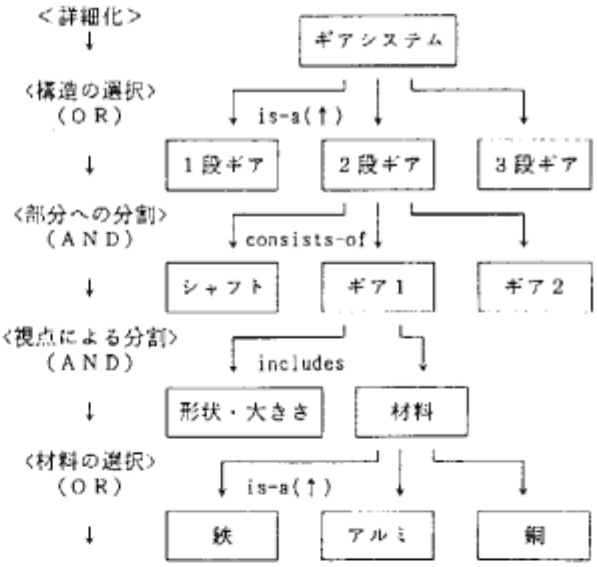

図5 対象モデルの構造を利用したトップダウン設計支援

ただし実際の設計過程では，これら3種の詳細化が同時並列的に実行される。また "includes" や "consists-of" 関係により対象を分割する場合，一般にそれらの間には何らかの依存関係が存在するため，詳細化の順序はそれによって実行時に決定される。

ところで "is-a"，"includes"，"consists-of" の3種の関係はそれぞれ木構造をなすが，詳細化処理での働き

を見ると，“is-a”では下位のクラスのいづれかを選択することになるが，“includes”及び“consists-of”では，包含するクラスや，部分として持っている構成要素は全て不可欠の要素である。従って詳細化の観点からは，前者はＯＲ関係の，後者はＡＮＤ関係の木構造として見え，それぞれＯＲ並列，ＡＮＤ並列的に詳細化を実行することになる。

5．ＦＲＥＥＤＯＭ／Ｓ０

5.1 基本機能

FREEDOM の最終目標は並列推論マシン上での統合的な設計支援システムにおける，汎用的な設計対象表現機構の実現であるが，ここではその第一段階として，逐次推論マシンＰＳ-Ｉ上に開発中のプロトタイプ FREEDOM/S0 について述べる。

FREEDOM/S0 は比較的小規模な組み合わせ設計，パラメトリック設計を対象とし，設計対象は記号的，数値的に取り扱うことが可能なものとする。すなわち FREEDOM/S0 は FREEDOM の必要最小限の機能を有する実験的なシステムである。

FREEDOM/S0 はオブジェクト指向言語ＥＳＰ[Chikayama84]で記述され，数式に関する制約充足メカニズムとして制約論理型言語ＣＡＬ[坂井88]の機能を利用することを考えている。

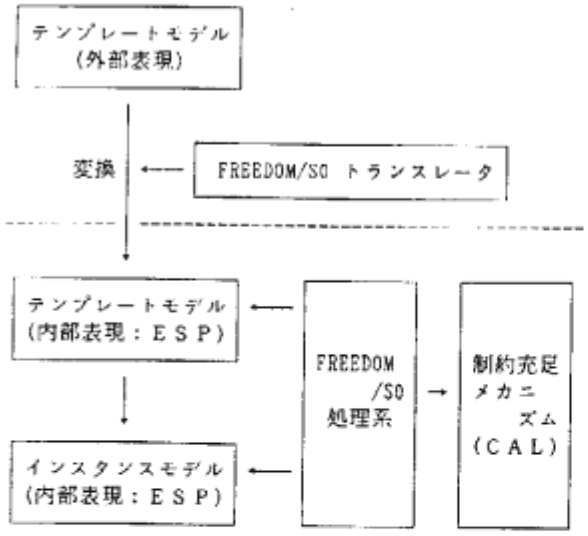

図６ FREEDOM/S0 の構成

FREEDOM/S0 の構成を図６に示す。FREEDOM/S0 におけるオブジェクトは外部表現と内部表現のふたつの表現形態をもつ。外部表現はいわばテンプレートモデルのソースプログラムであって，次節で述べるようなユーザに理解しやすい文法で記述される。外部表現はトランスレータによりＥＳＰ形式の内部表現に変換されて実行可能となる。

5.2 外部表現の記述形式

FREEDOM/S0 の記述形式はＥＳＰユーザにとって違和感のないことを重視し，ＥＳＰに近い文法とする。

また本来はインスタンスの静的な記述も可能としたいが，FREEDOM/S0 ではＥＳＰと同様にクラス定義（テンプレートモデル）のみを記述し，インスタンス（インスタンスモデル）は動的に生成することとした。

現在検討中のクラス定義の基本形式を図７に示す。ただし必ずしも全項目を記述する必要はない。また，各項目で［ ］で囲まれた部分は省略できる。

```
fr_class  クラス名  has
    is_a  上位クラス名 ;
    includes
        [キー名１ :: ] 包含クラス名１,
                  ・・・
        [キー名ｉ :: ] 包含クラス名ｉ;
    common attributes
        共有属性名１［ ::=  デフォルト値１］,
                  ・・・
        共有属性名ｊ［ ::=  デフォルト値ｊ］;
    consists_of
        構成要素名１［ ::=  構成要素クラス名１］,
                  ・・・
        構成要素名ｋ［ ::=  構成要素クラス名ｋ］;
    attributes
        属性名１［ ::=  デフォルト値１］,
                  ・・・
        属性名ｍ［ ::=  デフォルト値ｍ］;
    constraints
        制約式１,
          ・・・
        制約式ｎ;
  instance
    :meth(Obj, メッセージ) :- メソッド本体
                  ・・・
  local
                  ・・・
fr_end.
```

図７ FREEDOM/S0 のクラス定義形式

ここで，包含するクラス名にはキーを付けることができ，キー名を指定することによりオブジェクトの特定の側面の

みに着目して処理を実行することができる。制約式の記述は述語表現を基本とする。またＥＳＰと異なり，"instance" 以下にはメソッドのみしか記述できず，その形式もＥＳＰとは違っている。

メッセージ送信の記述形式は

:send(送信先のオブジェクト，メッセージ)

で，メッセージの形式は

メッセージ名(引数１，・・・，引数ｎ)

である。"local" 以下にはＥＳＰと同様のローカル述語を定義できる。

なおメソッドは対象モデルを記述するためというより，外部の設計手続きが対象モデルを操作する場合のインタフェースを与えるためのものである。

また，インスタンスの生成，クラスの明示的変更，詳細化，属性値の変更，構成要素の変更，制約の追加及び削除等の処理を行うメソッドをシステム提供する。

5.3 実現方式

FREEDOM ではインスタンスの属するクラスの動的変更が可能なため，これを効率よく行える実現方式が要求される。

そこでＥＳＰによる内部表現では，ひとつのインスタンスを，継承するクラス数だけのＥＳＰのインスタンス（以下部分インスタンスと呼ぶ）の組み合わせとする。すなわちクラスの "is-a" 関係，"includes" 関係によるクラスの階層構造がそのままインスタンスの構造に反映されることになる。従ってクラス定義が上位クラスとの差分のみの記述となっているのと同様に，部分インスタンスは上位クラスとの差分のみを表現する。

これによりクラスの動的変更処理をクラス間の差分を表わす部分インスタンスの追加，削除で実現できる。また，性質の継承機能はいわゆる "delegation" [Liberman86] によって実現する。

図３のクラス構成における「四角形の形状を持つ鉄板」を表わす，クラス「金属板」のインスタンスの内部構成を図８に示す。

図８の例で，インスタンスの属するクラスを「金属板」から「板」に変更する場合には参照先を「金属板」の部分インスタンスからその上位の「板」の部分インスタンスへ変更すればよい（このとき「金属板」の部分インスタンスは不要になる）。また金属板の形状を四角形から三角形に変更する場合は，「板」の部分インスタンスの "includes" のポインタが指している「四角形」の部分インスタンスを「三角形」の部分インスタンスに交換すればよい。

ところで処理のオーバヘッドを考慮すると，大規模な対象モデルを扱う場合には常時制約充足を行うのではなく，明示的に指定された時のみ実行するのが実用的と思われる。

またオブジェクトの属するクラスの自動変更機能をどの程度支援するかについては現在ふたつの案がある。ひとつは決定可能な最も詳細な（制約のきつい）クラスを常に維持する方法である。この方法はできる限りの詳細化処理をシステムが自動的に実行してくれる点では便利である。しかし，そのためには常にクラスの探索処理を実行していなければならず，オーバヘッドが非常に大きくなるという問題がある。

もうひとつの案は詳細化の方向，すなわちクラス階層を下にたどる操作は明示的に指定された時のみ実行し，制約充足が不可能な場合に，より抽象的なクラスに変更することのみを自動的に行うものである。このことは詳細化におけるバックトラック処理を自動化することを表わしている。処理速度を考慮するとこちらの方が実用的である。

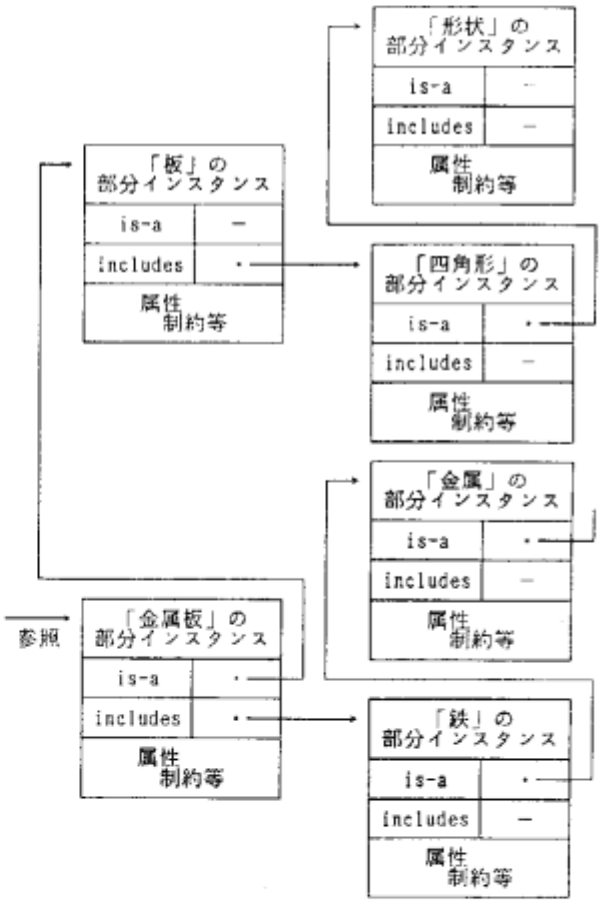

図８ インスタンスの内部構成

６．関連する研究

ミンスキーのフレーム理論[Minsky75]以来，対象をフレー

ムで表現することは知識工学における標準的な手法になっている。そして最近ではフレームとよく似た概念であり，動的機能を重視したオブジェクト指向による表現が主流になっており，設計型問題へも多く適用されている[木村88]。

しかし，従来のオブジェクト指向言語は制約を利用する機能が乏しく，付加手続きやメソッドにより手続き的に記述する必要があり，そのままでは知識表現として優れているとは言い難い。

一方，対象の持つ制約に着目し，制約記述による対象の表現も数多くなされている[Stallman77][Sussman80][Heintze87]。これらは構造的な記述がなされていないため，大規模な対象の記述は難しいが，対象における属性間の関係を宣言的に記述できる点が優れている。

そこで，オブジェクト指向に制約を導入する方式も試みられているが[Borning 81,86][Harris86][Struss87]，そこで扱われている制約は属性（インスタンス変数）間の数値上の制約に限られている。このため制約が属性値の決定にしか利用されない。

これに対し，FREEDOM ではオブジェクト指向のクラス定義自体を特定の種類のオブジェクトに関する制約記述ととらえ，クラスの探索にも制約充足を利用する等，より積極的に制約を利用している点に大きな特徴がある。この点で本方式は単なるオブジェクト指向への制約の導入というより，オブジェクト指向パラダイムを制約の観点で新たにとらえ直そうという試みである。

7．今後の課題

FREEDOM/S0 では対象モデルのクラス定義が固定的であり，自由な構造を持つ対象を設計することはできない。従って FREEDOM/S0 が対応できる設計はいわゆる組み合わせ設計やパラメトリック設計等のルーチン設計に限られる。これを開発設計に対処できるように発展させるには，クラスで定義されていない構造を持つインスタンスの生成や，クラスの動的生成を可能とする等の機能拡張が必要である。

また実際の設計はマクロ的にはトップダウンになされるが，ミクロ的にはボトムアップ的な手法が用いられることも多い。しかし FREEDOM/S0 はトップダウン的な詳細化の支援に重点を置き，ボトムアップ的な設計向けの機能は提供していない。従って今後ボトムアップ的な設計支援を可能とするとともに，両者の協調処理を実現したいと考えている。

ところで大規模なシステムの設計では全体を部分問題に分割して並列に解くことにより効率のよい問題解決を実現することが期待される。そこで現在，自然な形の分割法として，対象モデルの構造を利用する方式を検討中である。

一般に対象モデルは複数の部品を階層的に組み合わせたものである。そこでまず部品（オブジェクト）単位に部分問題に分割することが考えられる。しかしこの分割のみではそれほど多くの並列度は望めない。そこでさらにひとつのオブジェクトを複数の視点から眺め，各視点単位に部分問題に分割することが考えられる。

この点については既に 4.3 で述べたように，FREEDOM ではトップダウン設計における詳細化の過程で図5に示したような形の並列性を自然に生み出せるのではないかと考えている。また FREEDOM/S0 のインスタンスは複数の部分インスタンスより形成されるため，部分インスタンス単位に分割して処理することにより，さらに大きな並列度が得られる可能性がある。

しかし一般に部分問題間には何らかの依存関係が存在し，その扱いが問題となる。そこで現在，部分問題間の依存関係を制約として表現し，制約伝播により部分問題間の協調を図る方式[Stefik81]を検討中である。今後，より検討を深め，FREEDOM を並行処理機能を持つシステムへと発展させたい。

8．おわりに

以上設計型問題における対象モデル記述のための知識表現システム FREEDOM を提案し，現在開発中のプロトタイプ FREEDOM/S0 の機能と実現方式について述べた。

FREEDOM における知識表現はオブジェクト指向と制約指向の融合を図ったもので，制約の宣言的記述，柔軟なクラス・インスタンス関係，“is-a” と “includes” 関係に基づくクラス階層の表現，制約充足による詳細化支援等の特徴を有する。そして，設計過程で生成される対象モデルを，常に制約を満足する状態に保つ機能を提供することにより，効率のよい設計過程の支援を可能としている。

FREEDOM は設計対象の記述を目的に開発を進めているものであるが，その基本機能は汎用的であり，広範囲の問題に適用可能と考えている。

参考文献

[Borning81] Borning, A. "The Programming Language aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM Trans. on Prog. Lang. and Syst. vol.3, no.4, pp353-387, (1981)

[Borning86] Borning, A. and Duisberg, R. "Constraint-Based Tools for BuildingUser Interfaces", ACM Trans. on Graphics, vol.5, no.4, pp345-374, (1986)

[Chikayama84] Chikayama, T. "ESP Reference Manual", ICOT Technical Report, TR-044, (1984)

[Doyle79] Doyle, J. "A Truth Maintenance System", Artificial Intelligence, vol.12, pp231-272, (1979)

[Harris86] Harris, D.R. "A Hybrid Structured Object and Constraint Representation Language", Proc. of AAAI-86, pp986-990, (1986)
[Heintze87] Heintze, N., Michaylov, S. and Stuckey, P. "CLP(R) and Some Engineering Problems", in Lassez, J.L. (ed.) "Logic Programming, Proc. of 4th Int. Conf.", MIT Press, 675-703, (1987)
[木村88] 木村文彦 "オブジェクト指向によるＣＡＤ／ＣＡＭのためのモデリングとデータベース", 情報処理, vol.29, no.7, pp368-373, (1988)
[Lieberman86] Lieberman, H. "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems", OOPSLA'86 Proc. pp214-223, (1986)
[Minsky75] Minsky, M. "A Framework for Representing Knowledge", in Winston, P.H. (ed.) "The Psycology of Computer Vision", McGraw-Hill, (1975)
[Ohsuga85] Ohsuga, S. "Conceptual Design of CAD Systems Involving KnowledgeBases", in Gero, J. (ed.) "Knowledge Engineering inComputer-Aided Design", pp29-88, North-Holland, (1985)
[坂井88] 坂井公, 相場亮 "ＣＡＬ：制約論理プログラミングの理論と実例", 電子情報通信学会研究会資料, SS87-22, (1988)
[Stallman77] Stallman, R.M. and Sussman, G.J. "Forward Reasoning andDependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", Artificial Intelligence, vol.9, pp135-196, (1977)
[Stefik81] Stefik, M. "Planning with Constraints (Molgen:Part1)", Artificial Intelligence, vol.16, pp111-139, (1981)
[Stefik86] Stefik, M. and Bobrow, D.G. "Object-Oriented Programming: Themes and Variations", AI Magazine, vol.6, no.4, pp40-62, (1986)
[Struss87] Struss, P. "Multiple Representation of Structure and Function", in Gero, J. (ed.) "Expert Systems in Computer-Aided Design", North-Holland, (1987)
[Sussman80] Sussman, G.J. and Steel G.L.Jr. "Constraints - A Language for Expressing Almost-Hierarchical Descriptions", Artificial Intelligence, vol.14, pp1-39, (1980)
[Sutherland63] Sutherland, I.E. "SKETCHPAD: A Man-Machine GraphicalCommunication System", Proc. of Spring Joint Computer Conference, (1963)
[上野85] 上野晴樹 「知識工学入門」 6.1 対象モデル, オーム社 (1985)
[横山88] 横山孝典 "設計型問題向き対象モデル表現方式-オブジェクト指向パラダイムの拡張-", 情報処理学会第37回全国大会予稿集, 5H-8, (1988)