特　集　「知識の共有と再利用」

知識ベース再利用へのアプローチ
—KADSを中心とした欧州における動向—

Approaches to Knowledge Reuse
—KADS-Related Activities in Europe—

元田　浩*
Hiroshi Motoda

*　(株)日立製作所 基礎研究所
Advanced Research Laboratory, Hitachi, Ltd.

1993年8月15日　受理

Keywords: knowledge reuse, conceptual modeling, knowledge modeling.

1. はじめに

知識工学が提唱された頃は，知識と推論は明確に分離すべきであり，推論部は独立にプログラム化することができるので，専門家は知識の記述だけに専心すればよく，少量の知識でもシステムはすぐに動き，知識の量や質を向上させればそれにつれ性能も向上する，というラピッドプロトタイピングのメリットが強調されていた．しかし，研究が進むにつれ，知識表現はその知識が問題解決過程でどのように使われるかに依存すること，また知識レベルの表現[Newell 82]と具体的な計算機で動かす実装レベルの表現とは区別[*1]すべきであることが認識されてきた．

今日では，問題解決にはさまざまな違った役割を果たす知識が関与し，対象ドメインの知識と問題の性質に関する知識（タスクの知識）を明確に区別し，知識を構造化するのは当然のこととして受け止められている．特に，専門知識（エキスパティーズ）のモデル化を実装技術とは独立に実施することが重要で，これを通常，知識モデリングと呼んでいる[*2]．

本稿では，このような視点に立つ知識ベース構築手法の代表的なものの一つであり，ESPRITプロジェクトの一環として，1982年よりヨーロッパで開発されてきたKADSならびにその周辺の技術を，知識の再利用という観点から紹介する．

2. KADSの概要

KADS[*3]は，KADS-IとKADS-IIの二つのフェーズに分かれており，専門家モデルを中心とするKADSの基本方針が確立された1990年までの成果を総称してKADS-I，それ以降の商業化を目指した研究(Common KADSともいわれている）をKADS-IIと呼んでいる．

KADSでは，知識ベースシステムの開発に次節以降に紹介する五つの原則を採用している．このうち，専門家モデルを表す4層モデルと，知識の再利用が最大の特徴である[Breuker 89, Wielinga 92a, Wielinga 92b].

2・1　複数モデルによる複雑さの回避

知識ベース開発は複雑なプロセスであるので，すべてを一つのモデルで扱うのではなく，上流から下流にかけて複数のモデルを採用する．具体的には組織モデル（導入分析），応用モデル（問題，システムの機能の定義，外部の制約），タスクモデル(機能達成のためのタスクの組合せの同定，さらにタスク分解，タスク分配，タスク環境に細分され分析），協調モデル(ユーザと知識ベースシステムの間で必要な協調作業の同定），専門家モデル（タスク達成に必要な専門知識の種類の

*1　Newellは，この二つをKnowledge levelとSymbol levelと区別した．
*2　KADSでは，このモデルのことを概念モデルと呼ぶ．
*3　KADSは，最初はKnowledge Analysis and Documentation Systemの略称であったが，後にはKnowledge Analysis and Design Support，あるいはKnowledge Analysis and Design Structuringの略称とも解釈されるようになった．

同定)，概念モデル(実装に独立な前2者を合わせたもの)，設計モデル（概念モデルを実装するための設計)に区別している．

これを見てわかるように，単にソフトウェアとしての知識ベースシステムを作るだけでなく，なぜこのようなシステムを開発する必要があるのか，ユーザとの関わり合いはどうあるべきかなど,知識ベースシステムの開発の全工程を支援している．最大の特徴は概念モデルを実装に独立なものとし，実装上の制限に煩わされることなく専門家モデルを構築できるようにしたことである．したがって，概念モデル自身は実世界の現象を素直に表現する言葉で書くことができる．このことはもちろんKADSの大きな利点ではあるが,同時に概念モデルができたからといって,即,動くシステムができたことにはならないという問題点を含んでいる．

2・2 専門家モデル

KADSの枠組みのうち，最も中心的なものが概念モデルである．概念モデルは専門家の問題解決の方法を抽象的に記述したもので，知識の表現レベルでいえばNewellの知識レベルに相当する．KADSでは知識の役割を全部でドメイン，推論，タスク，戦略の4層に分割している．

〔1〕 ドメイン層

ドメイン固有の概念，その特性(属性)，概念間の関係，概念の特性間の関係(因果関係や時間関係)，概念の構造を静的（つまり宣言的に）に記述する．これらはドメインのオントロジーを規定するもので，これらを用いてドメインで成立する事実や，ドメインモデル(例えば，デバイスのプロセス）を表現する．この際，ドメインの見立を規定するドメインモデルオントロジーと，そのオントロジーに従って実際にドメインの実体を記述するドメインオントロジーを区別する．つまり，モデルオントロジーはドメインオントロジーのメタ記述であり，これが間接的に使われ方を規定することになる．一つのドメインに対して，複数のモデルオントロジーがある.タスクと分離して記述することが,ドメイン知識の再利用可能性を与える第一歩となる．

〔2〕 推論層

ドメイン知識で，どのような推論が可能かを記述する．ここで定義される推論が文字どおり推論の単位となる.実際にどのようにしてこの推論を実施するかは,概念モデルを構築するうえでは知る必要はない．したがって，推論の実体を示す名前（例えば，抽象化，照合，分解)，入出力，ドメイン知識のどの部分を使うかのみを規定する．これらはそれぞれ知識源，メタクラス，ドメインビューと呼ばれている．メタクラスには，二つの目的があり，一つはドメインオブジェクトが問題解析過程で果たす役割（例えば仮説，証拠などでドメインオブジェクトのメタ記述となっている）を表明すること，他の一つはこの役割を果たすドメインオブジェクトのタイプを示す（つまり，使えるドメイン知識を絞る）ことである．一つのドメインオブジェクトは，複数のメタクラスのインスタンスになり得る．

〔3〕 タスク層

具体的に，知識源をどのように組み合わせて推論をすればゴールが達成できるかを規定する．つまり，問題解決のフローを表す．ここでは解くべきタスクを分解してゴール，サブゴールの関係を明確にし，これらをメタクラスに対応させ（ゴールを達成することは出力のメタクラスの実体を求めることに相当する),タスク構造を明確にする．推論層で定義した知識源は，これ以上分解できないタスクに相当する．

サブタスク間のフロー制御を記述するために，制御項目(Control-terms)を定義する．制御項目はメタクラスに対するラベルにほかならない(例えば,「仮説が零でない間は…をせよ」という制御フローでは，まだふるい落とされていない仮説の集合に対して「差分(differential)」なるラベルを貼る)．制御構造そのものを説述するためにRepeat-until, Do_for_each_untilなどの記述子が，協調モデルで規定される外部との情報授受のタスクを実行するために，獲得(obtain)，提示（present)，受領（receive)，提供（provide）の四つの語彙*4が準備されている．

〔4〕 戦略層

ユーザの要求，データの性質，計算資源などの変化に対応し，必要なゴールの修正，タスク構造の修正，再立案を実施する．人間が環境の変化に順応し，戦略を変えることに対応する．今までに開発されたシステムでは，この層を本格的に記述したものはない．

〔5〕 4層の関係

これらの4層の関係の解釈の仕方はいろいろある．一つの解釈の仕方は，各層をどのように解釈するかを一つ上の層が規定していると解釈するものである（図1)．この解釈に従えば，各層はドメインレベルで何をなし得るかに対し，新たな制約を付加している．さらに，各層をメタ記述でつないだ実行可能なものと考えれば，非常に遅い（効率の悪い）KBSになっていると見ることもできる．実際，KADSの方法論で開発されたかなりの数のエキスパートシステムで,最初の3層の

＊4 システム側から見た表現であるが，最初の二つはシステム主導，後の二つはユーザ主導である．

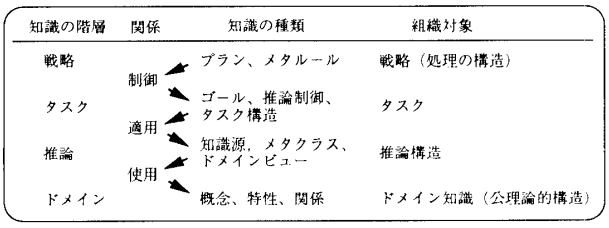


図1 各知識層の間の関係

階層化(大半のものは戦略層を省略)が認められている．

2・3 推論構造・タスク構造のライブラリ化と再利用

KADSでは，この4層モデルが知識の共有と再利用の基本になる．まず，知識源やメタクラス自身が再利用可能な語彙となる．さらに，ドメイン層とそれ以外の層とを明確に分離することによって，再利用可能な問題解決方法をライブラリの形で蓄積することが可能となる．KADSではドメイン層を除いた問題解決法，つまりタスク層，推論層を解釈モデル(interpretation model)と称している*5．したがって，解釈モデルにドメイン層の知識を付加したものがKADSの概念モデルとなる．新たな問題（タスク）に対する概念モデルを作るには，類似の解釈モデルを選択し，これをテンプレートとして利用し，問題に固有の修正を推論層，タスク層，戦略層に施し，最後にドメイン層の知識を補充すればよい．ドメイン独立，タスク依存の知識記述方針がこれを可能にしている．

そこで，重要となるのが知識源やメタクラスに用いる語彙（オントロジー）の選択である．知識源は，ドメインに対する可能な推論操作を規定するものとして定義したので，基本的にはタスクに独立な語彙である．どれだけのタイプの知識源を準備すれば十分であるかは難しいが，KADSでは推論操作による入力と出力のメタクラスの性質の差異に基づき，三つのカテゴリー(概念の中身を変更，概念から新しい概念を生成，概念間の関係を生成)に分類し，これに知識源の構造を操作するものを加え，図2に示す13種類の知識源を提案している．

一方，メタクラスは知識源の引数であり，知識源に対する役割を表す．役割は問題解決過程の各段階で変わるので，これを系統的に整理するのも難しい．KADSでは問題解決過程を時系列的，因果的な観点から眺め，入力（問題），ドメイン固有の知識（システムモデル），中間的なデータ，中間の解，最終解に整理して，おのおのの役割を分類している．

知識源やメタクラスは専門知識を表現するための語彙であるが，解釈モデルはそれらを用いて専門知識の部分モデルを構成するものである．ドメイン知識なしの専門知識とみなすことができる．特定のタスクに対して，どのように知識源を使えばよいかを記述する．

推論操作のタイプ	知識源	メタクラス(入力 → 出力)
概念、構造、事例の生成 これらの知識源は概念や事例から新しい概念や事例を成。	具体化(instantiate)	概念 → 事例
	分類(classify)	事例 → 概念
	一般化(generalize)	事例の集合 → 概念
	抽象化(abstract)	概念 → (新しい)概念
	特殊化(specify)	概念 → 概念
	選択(select)	集合 → 概念
概念の変換 属性に値を付与。これらの知識源は概念の属性値を操作。	値の付与(assign-value)	属性 → 属性の値
	計算(compute)	構造 → 属性の値
値や構造の差分（区別） 共通部あるいは差を表明する。ゴール達成されたかどうかなど推論の制御によく使用される。	比較(compare)	値＋値 → 値
	照合(match)	構造＋構造 → 構造
構造の操作 構造を操作して別の構造に変換する。構造の種類だけ変換法がありえるが、ここではよく使われるもののみを示す。	組立(assemble)	事例の集合（構成要素） → 構造
	分解(decompose)	構造 → 事例の集合
	変換(transform)	構造 → 構造

図2 KADSにおける知識源の分類

- メタクラス(metaclass)
  - 問題(problem)
    - 質問(question)
  - データ(data)
    - データ構造(data_structure)
      - 事例記述(case_description)
      - システム記述(system_description)
    - 個別データ(individual_data)
      - 制約(constraint)
      - 変量(variable)
      - 兆候(symptom)
        - 不具合(complaint)
  - 中間データ、解の役割(intermediary_role_of_data/problem)
    - パラメータ(parameter)
    - 要因(factor)
    - 所見(finding)
      - 証拠(evidence)
  - ドメイン知識の役割(intermediary_of_domain_knowledge)
    - システムモデル(system_model)
    - 仮説(hypothesis)
    - 規範(norm)
    - 項目(term)
  - 解(solution)
    - 診断(diagnosis)
    - 分類(decision_class)
    - 計画(plan)
    - 設計(design)

図3 KADSにおけるメタクラスの分類

- システム分析(system_analysis)
  - 同定(identify)
    - 分類(classify)
      - 単純分類(simple_classify)
      - 診断(diagnosis)
        - 単一故障診断(single_fault_diagnosis)
          - 経験的分類(heuristic_classification)
          - 系統的診断(systematic_diagnosis)
            - 因果追跡(causal_tracing)
            - 局在化(localization)
        - 多重故障診断(multiple_fault_diagnosis)
    - 査定(assessment)
  - モニタ(monior)
  - 予測(predicion)
    - 挙動予測(prediction_of_behavior)
    - 値の予測(prediction_of_values)
- システム修正(system_modification)
  - 修繕(repair)
  - 対症療法(remedy)
  - 制御(control)
    - 維持(maintain)
- システム合成(system_synthesis)
  - 変換(transformation)
  - 設計(design)
    - 単純設計(transformational_design)
    - 修正設計(refinement_design)
      - 単一修正設計(single_stream_refinement_design)
      - 多重修正設計(multiple_stream_refinement_design)
    - 配置・構成(configuration)
  - 計画(planning)
  - モデリング(modelling)

図4 KADSにおけるタスクの分類

＊5 専門家がインタビューで答える知識を解釈する指針となるので，解釈モデルと呼ばれている．

ドメインに独立であるので，他のドメインに対する同じタスクに再利用可能である．

解釈モデルはタスクの性質に従って，決定木の形に分類されライブラリ化されている．図4に，KADSのタスクの分類を，図3と同じ形式で表示して示す．この決定木の枝が，一つ（あるいは二つ以上のこともある）の解釈モデルに対応している．

2・4 個別問題に対処するためのモデルの修正

解釈モデルをテンプレートとして利用することにより，新しいタスクの概念モデルの構築が容易になる．しかし，ライブラリから持ってきた解釈モデルがそのままタスクモデルになることはまれで，多くの場合修正が必要となる．例えば，図5の左上に示す診断モデルでは診断の対象となるシステムについて，すでにどのような部品から構成されているかがわかっていることを想定している．もしわかっていない場合には，必要な知識を入れてモデルを組み立てなければならない．その場合でも，部品に着目して分解することは決まっている．しかし，機能について分解するほうがよい場合もある．このように，何に着目してモデルを組み立てるかを決めることも含める場合には，新たな視点の選択が必要になる．同図はこのような修正を順に加えて解釈モデルの一部を修正していく様子を示している．

この考え方を推し進めると，モデル構築オペレータという考えに達する．図4の枝に示す解釈モデルよりは小さな部分構造で，多くの解釈モデルに共通して現れるものをオペレータとして準備しておき，これを粗いモデルに順次適用し，モデルを実際のタスクに合うように洗練していけばよい．Common KADSでは，このような形で解釈モデルの部品を整理しようとしている．

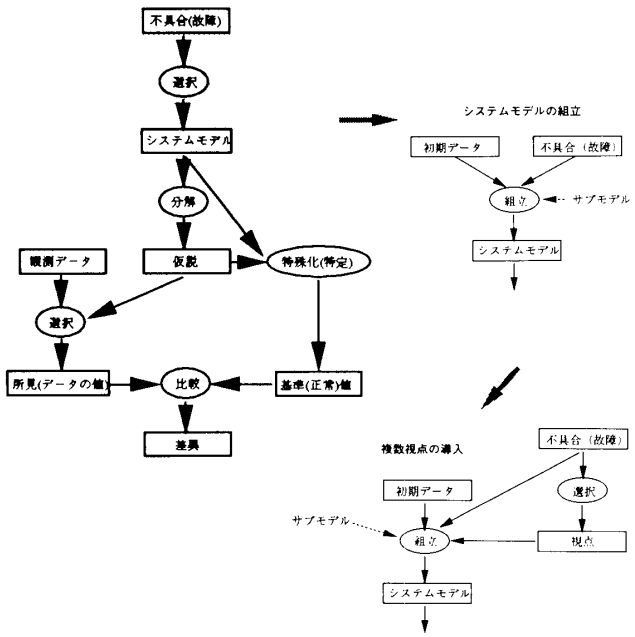


図5 解釈モデルの修正

2・5 知識モデルと実行可能プログラムとの構造同形性

KADSの概念モデルは実行可能でないので，概念モデルから設計モデルを作り実行可能なプログラムを作成しなければならない．知識レベルで表現された概念モデルには実装に必要な情報が不足している．具体的にどの言語を使い，知識をどう表現し，知識源をどうコーディングするかなどを決めなければならない．概念モデルは機能仕様であるので，同一仕様を満足する設計モデルはいくつも存在する．したがって，既存のエキスパートシステム構築用のシェルを使って，概念モデルから設計モデルを作ることも可能であり，実際，多くのシステムがこの方法で構築されている．しかし，このような作り方をすると，でき上がった実行可能プログラムの構成要素と概念モデルの構成要素の間に対応がつけにくい．

KADSでは，知識(概念)モデルと実行可能プログラムとの構造同形性を強く推奨している．構造が同形であることによって，知識ベースシステムの推論プロセスに関し，知識レベルの言葉を使った説明が可能になる．つまり，タスク構造中のどのサブタスクのなかの，どの知識源を実行中で，メタクラスや制御項目の値は何に束縛されているか，などが容易に把握できる．つねに概念モデルに立ち戻って考えられるので，デバッグや保守もやりやすい．また，知識獲得に際しても，概念モデルのどの機能に関する知識がほしいかが明確になるので，適切な知識獲得法を使うことができるようになる．このような利点を生かすため，構造同形性を保証する設計モデル構築手法が提案されている．

3. KADSの周辺

前章までで，KADSの基本的な考え方を説明した．すでに述べたように，KADSの弱点は，実行可能なプログラムを自動生成してくれないことである．これを克服するための研究が，KADSプロジェクトの内外で勢力的に行われている．本章では，KADSの方法論をサポートする研究のいくつかを紹介する．

3・1 統合ワークベンチShelly

Shelly［Anjewierden 92］は，KADSの概念モデルと設計モデルをサポートすることを目標に開発された統合ワークベンチである．知識エンジニア（ドメイン

の専門家ではない)を対象としており，AST(Activity Support Tool) と呼ばれる多くのツール（現在は9種類）から構成されている．ライフサイクル中のどのフェーズにいるかに従って，使用すべきASTの種類と順番が異なるが，どのASTをどういう順番で使うかはユーザに任されている（つまりShelly主導ではない)．各ASTは，プロトコルなどのデータと他のASTで作られたモデルを入力として受け取り，そのASTがサポートするモデルとドキュメントを出力する．AST間でデータの変換が必要とならないよう，データ構造はShelly内で統一されている．すべての経過は画面上にダイヤグラムなどを使って視覚的に表示される．

各フェーズの機能を達成するためのアルゴリズムはライブラリとして持っており，これを選択し使用するようになっている．したがって，ライブラリにないものは対処できない．また，設計モデルは実装の一歩手前のモデルであって，実行可能プログラムではない．この後，設計モデルから実行可能プログラムを作る仕事は知識エンジニアに任されている．

3・2 実行可能モデルへの変換言語

〔1〕 MODEL-K/OMOS/MOMO

KADSの概念モデルから設計モデルを経由して，実行可能プログラムに至る手間があまりにも面倒であることから，Linster [Linster 92]らは，概念モデルから直接実行可能なプログラムを生成する言語MODEL-Kを開発した．MODEL-Kは一種の高級言語であり，同じ研究グループが開発したPROLOGベースのオブジェクト指向言語BABYLON上に構築されている．概念モデルをMODEL-Kで記述すれば，コンパイラが実行可能プログラムを生成してくれる．

したがって，概念モデルの記述にはアルゴリズムなど，KADSの概念モデル記述には必要なかった詳細な情報も埋め込む必要がある．そのため，タスク記述や知識源の記述にはボディ部(Task-body, Knowledge-source-body)と称するものを別に設け，そこに処理の実体をBABYLONやLISPを使って直接記述する．この段階で，メタクラスのデータ構造（例えば配列を使用）なども指定する．この考え方はドメイン知識にも適用され，例えば概念間の関係もボディ部(Relation-body)に，具体的に規則などの形で記述する．KADSとの関係を示すと，表1のようになる．MODEL-Kでは，KADSの主張する構造同形性が保証される．MODEL-Kを用いて，オフィスレイアウトシステムが開発されているが，タスク構造や知識源のライブラリ化には至っていない．

表1 KADSとMODEL-Kの関係

KADS概念モデル		実行可能化の方策	問題解決の動的制御
タスク層	タスク	制御命令	タスクアジェンダ
推論層	知識源	コードボディ	
	メタクラス	データ構造	メタクラスの現在値
ドメイン層	概念	概念実体	
	関係	関係タップル	

〔2〕 Components of Expertise

Steelsも，Components of Expertise (CoE) [Steels 90, Steels 92]と呼ぶ手法を開発し，KANというシステムを試作し，これを発展させたCOMMETという統合ワークベンチシステムを実現している．KADSの各知識源に対して，実行可能なモジュールをビルディングブロックとして準備しておき，これらを組み合わせて実行可能なシステムをオンラインで作り上げるものである．溝口らのMULTIS [Mizoguchi 92]に非常に近い．

対象とするユーザは，ドメインの専門家（非プログラマ）で，ユーザとのインタラクションはすべて知識レベルで実施する．したがって，知識レベルでモデルを編集するだけで，結果が実行可能プログラムに反映される．ライブラリ（アプリケーションキットと呼んでいる)を整備して知識の再利用を図る点では，一見，KADSの自動化を進めたものであるように思えるが，KADSがトップダウンのアプローチを採用し，知識レベルでのライブラリの標準化を狙っているのに対し，CoEではアプリケーションキットにさまざまな再利用可能な部品が徐々にボトムアップ的に蓄積されていくことを指向している*6．メタクラスのような推論に関する明示的な概念がなく，多くの方法論の寄せ集めの感があるが，KADS-IIプロジェクトでKADSの成果に組み込まれれば，さらに洗練されたものに発展するであろう．

4．他の研究との関連

以上，KADSならびにその周辺の主要な技術を概観してきたが，知識モデリングの考え方はESPRITプロジェクトだけの成果ではない．表現は違うが，世界各国で類似のシステムが研究されている．本章では，そのうちの代表的ないくつかを簡単に紹介する．

Clanceyは，経験的分類の記述に推論構造を導入している[Clancey 85]．早くから，タスク知識は専門家の知識の重要な要素であると指摘しており，彼のNEOMYCINの専門家モデルもその一例である．Clanceyのこれらの初期の研究は，KADSに大きな影響を与えている．

Chandrasekaranのgeneric task (GT) [Chandrasekaran 86, Chandrasekaran 88]もよく知られて

*6 Steelsは，これをevolutional ptocessと呼んでいる．

いる．GTは問題と解法を組み合わせたもので，各GTに適した知識表現と推論法を規定している．比較的小さなGTをうまく組み合わせれば，多くの問題を解くことができるという仮説に基づいている．GTの解法は，KADSの解釈モデルよりは粒度が小さい．粒度的には構築オペレータが近い．しかし，知識レベルで記述したものではなく，GTは実行可能なビルディングブロックである．

McDermottら[Fahelman 86, McDermott 88]もhalf weak methodと呼ばれているタスクに依存する制御知識のみを組み込んだ問題解法（例えば，propose & revise, cover & differentiateなど）を前もって準備しておき，解法固有の言葉で知識獲得をガイドする方法を実践してきた．知識獲得が終了した段階で動くシステムが構築される．これらの解法はKADSの解釈モデルと同じ粒度である．彼らは，その後，Spark/Burn/FireFighter [Klinker 91, Marques.92]と呼ばれる．専門家が直接知識ベースシステムを開発するための統合環境を開発している．メカニズムと呼ばれる実行可能な解法をあらかじめビルディングブロックとして準備しておき，これらを組み合わせて特定のタスク向けの実行可能なシステムを構築する．メカニズムの粒度は知識源とほぼ同じであるが，実行可能である点が異なる．

MusenのPROTEGE [Musen 89]は，知識獲得システムを作るメタ知識獲得システムである．タスク構造を用いて知識獲得をガイドする点ではKADSと同じであるが，作成された知識獲得システムはドメインの用語を知っており，専門家の知っているドメインの言葉で知識獲得をガイドする点が違う．

溝口らのMULTIS [Mizoguchi 92]は，タスク分析に力を入れた問題解決エンジン生成コンパイラである．タスクオントロジーとドメインオントロジーを分離した点，問題解決過程を記述する知識レベルと実行可能なビルディングブロックを分離した点など，KADSを含めて上に紹介した手法と共通点は多い．MULTISでは，タスクオントロジーに基づき専門家が理解できる汎化語彙（名詞，動詞）を定義し，問題解決過程をドメインに独立に記述する．KADSの知識源が汎化動詞に，メタクラスが汎化名詞に対応する．KADSではこれらはタスクオントロジーに基づいて定められてはおらず，むしろ，タスクに依存しないものとしており，この点に相違がある．

堀らのCAKE [Hori 92]も，類似の考えに基づくシステムである．プリミティブな問題解決方法を準備し，これを組み合わせて具体的なタスクの問題解決方法を構築する方法論を開発している．CAKEでは，問題の仕様を記述するタスクオントロジーと，問題解決過程を記述する問題解決オントロジーを区別する．問題解決モデルの要素がKADSの知識源に対応するが，タスクが限定されているため，タスクに依存したものになっている．

上に紹介した研究では，それぞれ違った表現が使われているが，共通していえることは，問題解決にはそれぞれ果たす役目の違う複数の知識が関与しており，これらを明示的に区別することが重要であるということである．特に，ドメイン知識とタスク知識（制御知識）を分離することは，すべての研究で実現されている．どのような知識が必要かを，システム自身が知っていることが大切である．

5. おわりに

KADSプロジェクトは現在もKADS-IIとして進行中であり，KADSを用いて開発された知識ベースシステムは50を超える．これらのシステムの開発経験からKADSの4層モデル，概念モデルと設計モデルの分離，解釈モデルのライブラリ化などは知識ベースシステム開発のための明確な方法論を与えたものとして評価されている．しかし，本文中にも記した，以下に要約するいくつかの問題点も指摘されている．Common KADSはこれらの問題点を解決することを目標としてはいるが，これらを完全に解決することは難しいであろう．

- 4層モデルの語彙の表現力が不十分．
- 知識源の分類が一般的すぎる．
- 解釈モデルの蓄積が不十分．
- 概念モデルを実装するまでの過程の支援が不十分．
- 概念モデルの検証の方法がない．
- モデル化の過程の支援機能がない．

知識ベースシステムを構築することはモデリングそのものであり，知識獲得とは結局のところモデル獲得である．したがって，モデリングをいかにサポートするかが，かぎになってくる．モデリングには当然モデルを記述するための言語が必要である．言語を統一することができれば，すでに提案されている多くの研究成果を共有できるし，相互の比較も可能になる．言語を統一するということはオントロジーを統一するということである．現状では，類似の研究がいくらあっても，アイディアは利用できるが，時間をかけて作ったモデルはそれぞれのオントロジーに変換して，作りなおさなければ利用できないため，共有できない．各研

究機関の利害が絡んで難しいとは思うが，解決策を真剣に考えていかねばならない問題である．入れ物を作るのは容易であるが，中身を充実させるのは難しいし，時間もかかる．中身の共有に向けて，真の協力体制が確立されつつあることを望みたい．

◇参 考 文 献◇

[Anjewierden 92] Anjewierden, A., Wielemaker, J. and Toussaint, C.: Shelly — Computer - aided Knowledge Engineering, *Knowledge Acquisition*, Vol. 4, No. 1,pp. 109-125 (1992).

[Breuker 89] Breuker, J. A. and Wielinga, B. J.: Model Driven Knowledge Acquisition, Guida, P. and Tasso, G. (eds.), *Topics in the Design of Expert Systems*, pp. 265-296, Elsevier Science Pub. (1989).

[Chandrasekaran 86] Chandrasekaran, B. Generic Tasks in Knowledge - based Reasoning: High - Level Building Blocks for Expert System Design, *IEEE Expert*, Fall, pp. 23-30 (1986).

[Chandrasekaran 88] Chandrasekaran, B.: Generic Tasks as Building Blocks for Knowledge - based Systemrs: The Diagnosis and Routin Design Examples., *The Knowledy Engineering Review*, Vol. 3, No. 3, pp. 183-210 (1988).

[Clancey 85] Clancey, W.: Heuristic Classification, *Artif. Intell.*, Vol. 27, No. 3, pp. 289-350 (1985).

[Eshelman 86] Eshelman, L. and McDermott, J.: MOLE: A Knowledge Acquisition Tool that Uses its Head, *AAAI-86*, pp. 950-955 (1986).

[Hori 92] Hori, M., Nakamura, Y. and Hama, T.: Methodology for Configuring Scheduling Engines with Task-Specific Component, *Proc. 2nd Japanese Knowledge Acquistion Workshop for Knawledge-Based Systems*, pp. 215-229 (1992).

[Klinker 91] Klinker, G., Bhola, C., Dallemagne, G., Marques, D. and McDermott, J.: Usable and Reusable Programming Constructs, *Knowledge Acquisition*, Vol. 3, No. 2, pp. 117-135 (1991).

[Linster 92] Linster, M., Karbach, W., Voß, A. and Walther, J.: An Analysis of the Role of Operational Modeling Languages in the Development of Knowledge-based Systems, *Proc. 2nd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 65-87 (1992).

[Marques 92] Marques, D., Dallemagne, G., Klinker, G., McDermott, J. and Tung, D.: Easy Programming—Empowering People to Build Their Own Applications, *IEEE Expert*, June, pp. 16-29 (1992).

[McDermott 88] McDermott, J.: Using Problem-Solving Methods to Impose Structure on Knowledge, *Proc. Int. Workshop on Artif. Intell. for Industrial Applications*, pp. 7-11 (1988).

[Mizoguchi 92] Mizoguchi, R., Tijerio, Y. and Ikeda, M.: Task Ontology and Its Use in a Task Analysis Interview System — Two - level Mediating Representation in MULTIS—, *Proc. 2nd Japanese Knowledge Acquistition for Knowledge-Based Systems Workshop*, pp. 185-198 (1992).

[Musen 89] Musen, M. A.: Automated Support for Building and Extending Expert Models, *Machine Learning*, Vol. 4, No. 3/4, pp. 347-375 (1989).

[Newell 82] Newell, A.: The Knowledge Level, *Artif. Intell.*, Vol. 19, No. 2, pp. 87-127 (1982).

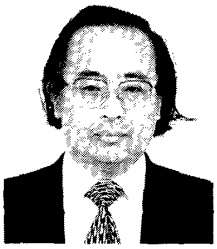
[Steels 90] Steels, L.: Components of Expertise, *AI Magazine*, Vol. 11, No. 2, pp. 28-49 (1990).

[Steels 92] Steels, L.: End-user Configuration of Applications. *Proc. 2nd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 43-64 (1992).

[Wielinga 92a] Wielinga, B. J., Shcreiber, A. Th. and Breuker, J. A.: KADS: A Modelling Approach to Knowledge Engineering, *Knowledge Acquisition*, Vol. 4, No. 1, pp. 5-53 (1992).

[Wielinga 92b] Wielinga, B. J., Van de Velde, W., Schereiber, G. and Akkermans, H.: The KADS Knowledge Modelling Approach, *Proc. 2nd Japanese Knoweldge Acquisition for Knowledge-Based Systems Workshop*,pp. 23-42 (1992).

著 者 紹 介



元田　浩(正会員)

1965年東京大学工学部原子力工学科卒業．1967年同大学院原子力工学専攻修士課程修了．同年，(株)日立製作所に入社．同社中央研究所，原子力研究所，エネルギー研究所を経て，現在，基礎研究所主管研究員．原子力システムの設計，運用，制御に関する研究，診断型エキスパートシステムの研究を経て，現在は人工知能の基礎研究，特に機械学習，知識獲得，視覚推論などの研究に従事．工学博士．日本ソフトウェア科学会理事，人工知能学会理事，同編集委員を歴任．Knowledge Acquisition (Academic Press) 編集委員，IEEE Expert 編集委員．Artificial Intelligence in Engineering (Elsevier Applied Science)編集委員，日本認知科学会編集委員会委員，1975年日本原子力学会奨励賞，1977，1984年日本原子力学会論文賞，1989，1992年人工知能学会論文賞受賞．情報処理学会，日本ソフトウェア科学会，日本認知科学会，AAAI，IEEE Computer Society 各会員．