知的インタビューシステム
に関する研究

1989年1月

川口敦生

知的インタビューシステム
に関する研究

1989年1月

川口　敦生

内容梗概

本論文は, 筆者が大阪大学大学院工学研究科 (電子工学専攻後期課程) において在学中に行なった「知的インタビューシステムに関する研究」をまとめたもので, 以下の 6 章から構成されている.

第 1 章は序論であり, 本研究の背景および従来の研究を概説すると共に本研究の目的を述べている.

第 2 章では, インタビューの一手法である静的解析を用いたデータベースの論理設計を支援する知的インタビューシステム *I²S-LD* について述べている. *I²S-LD* は 4 種類の質問戦略を用いてデータベースの発注者にインタビューを実施し, データベースの論理構造を設計する. データベースの論理設計に適したドメインモデルの表現形式として, プラン構造を定義し, この上で推論を行ないながらインタビューを実施する機構および必要な知識を示している. また, 質問戦略を用いてインタビューを行なうシステムの一般化を試みた *SIS* の構成についても述べている.

第 3 章では, 動的解析と呼ばれる手法を用いたデータベース構築支援システム *I²S-DB* について述べている. *I²S-DB* は *I²S-LD* の改良版であり, 質問戦略を用いたインタビューに加え, 実際にデータベースを試作して論理設計結果の問題点を発見する動的解析を用いる. ここでは *I²S-LD* に付加したデータベースを試作するための機構および知識について述べている. また試作したデータベースの問題点を検出し, インタビューを通じて修正する方法を述べている.

第 4 章では, インタビューシステムが持つべき学習機能について考察し, *I²S-LD* にドメインの知識の学習機能を付加することを試みている. *I²S-LD* に新しく導入した知識および学習機能について述べ, それらを用いた質問の生成法を示している. 学習機能を付加することにより, *I²S-LD* はよく似た質問の繰り返しを避け, よりインタビューの対象に密接した示唆を受け手に与えることが可能となることを示している.

第 5 章では, 各章での議論を整理し, インタビューシステムの一般アーキテクチャの設計を試みている. インタビュー時の質問生成には, インタビューシステムの問題解決機能および学習機能が重要な役割を担っていることを示し, それらに基づいたインタビューシステムの構成および動作について述べている.

第 6 章は結論であり, 本研究の主な成果をまとめ, あわせて今後に残された問題について述べている.

関連論文

A. 学会誌掲載論文

[1] 川口, 溝口, 山口, 角所 : "インタビュー形式でユーザーと対話するデータベース論理設計支援システム", 電子情報通信学会論文誌, Vol. J70-D, No.11, pp.2243-2249 (1987).

[2] 川口, 溝口, 角所 : "インタビューシステムのためのシェル, SIS", 人工知能学会論文誌 (Vol. 4, No. 4 (1989) に掲載予定).

B. 国際会議発表論文

[1] A. Kawaguchi, N. Taoka, R. Mizoguchi, T. Yamaguchi and O. Kakusho : "An Intelligent Interview System for Conceptual Design of Database", Proceedings of the 7th European Conference on Artificial Intelligence, 2, pp.39-47 (1986).

[2] A. Kawaguchi, R. Mizoguchi, T. Yamaguchi and O. Kakusho : "SIS:A Shell for Interview Systems", Proceedings of the 10th International Joint Conference on Artificial Intelligence, 1, pp.359-361 (1987).

C. 研究会発表論文

[1] 川口, 溝口, 山口, 馬野, 磯本, 角所 : "データベース構築支援エキスパートシステム —基本構成と対話による論理設計支援—", 情報処理学会知識工学と人工知能研究会報告, No.41, pp.25-32 (1985).

[2] 川口, 溝口, 山口, 角所 : "データベースの論理設計を支援する知的インタビューシステム", 情報処理学会知識工学と人工知能研究会報告, No.48, pp.1-8 (1986).

[3] 川口, 沼田, 溝口, 山口, 角所 : "知的インタビューシステム I^2S に基づくデータベース構築・利用支援システム", 情報処理学会知識工学と人工知能研究会報告, No.52, pp.73-80 (1987).

[4] 和田, 川口, 溝口, 山口, 角所 : "知的インタビューシステム I^2S における学習機能について", 情報処理学会知識工学と人工知能研究会報告, No.56, pp.97-104 (1988).

[5] 川口, 高橋, 溝口, 角所 : "インタビューシステムに関する一考察", 人工知能学会ヒューマンインタフェースと認知モデル研究会資料, SIG-HICG-8802, pp.29-38 (1988).

D. 学会口頭発表論文

[1] 川口, 溝口, 山口, 馬野, 磯本, 角所 : "知的インタビューシステムの試み —データベースの論理設計支援への応用—", 情報処理学会第 31 回全国大会, 9N-2 (1985).

[2] 川口, 溝口, 山口, 角所 : "知的インタビューシステム I^2S とその質問戦略—データベースの論理設計に関して—", 情報処理学会第 33 回全国大会, 4L-10 (1986).

[3] 沼田, 川口, 溝口, 山口, 角所 : “知的インタビューシステム I^2S に基づくデータベース構築支援システムの開発”, 情報処理学会第 33 回全国大会, 4L-11 (1986).

[4] 川口, 溝口, 山口, 角所 : “インタビューシステムのためのシェル, SIS とその設計思想”, 情報処理学会第 35 回全国大会, 5P-8 (1987).

[5] 川口, 溝口, 山口, 角所 : “インタビューシステムのためのシェル, SIS の開発”, 人工知能学会第 1 回全国大会, 6-21 (1987).

[6] 山田, 川口, 溝口, 山口, 馬場, 中島, 野村, 角所 : “設計時の仕様獲得を支援する知的インタビューシステム I^2S/D —油圧回路の設計に関して—”, 情報処理学会第 36 回全国大会, 4L-11 (1988).

目 次

図 目 次

表 目 次

第 1 章

序論

DENDRAL[Lederberg64] に始まるエキスパートシステムの研究は, 近年ますます盛んになりつつある. 人工知能の研究は, それまで推論機構を中心になされていた. エキスパートシステムの出現は, 研究の方向を知識中心へと変化させた.

エキスパートシステムの特徴は, 推論機関と知識ベースを分離したことにある. 推論機関として適当なものを用意できれば, あとは知識を用意するだけでエキスパートシステムが構築できる. またシステムの性能を, 知識の追加, 修正を通じて向上させることができる. したがって, この知識獲得の過程がシステムの性能を決定する本質的な役割を担っていると言える. 知識獲得の困難さは, エキスパートシステム研究の初期から認識されていた. そして MYCIN [Shortliffe76] プロジェクトで始めての知識獲得支援システム TEIRESIAS [Davis82] が開発された.

知識獲得には大きく分けて二つの手法, すなわち例題からの帰納的学習と専門家へのインタビューがある. 帰納的学習は, 例題すなわち問題とその解が多数容易に準備できる分野で有効である. 逆にインタビューは, 専門家の経験則が問題解決に重要な役割を果たすような分野に適する.

本論文は, インタビューシステムの試作・検討を通してインタビューを行うのに必要な知識および機構を検討したものである. インタビュー自体は, エキスパートシステム開発のみならず, 様々な分野で重要な役割を担っている. 例えばソフトウェア開発時の仕様獲得に発注者へのインタビューは欠かすことができない. そこで本論文では, エキスパートシステムの知識獲得にとらわれることなく議論を進めた.

一般に人間は自分自身の知識を意識することなく使用している. したがってインタビューを通じて必要な知識を余すところなく, また誤りなく獲得するのは困難である. この状況に対処するには, 誤り, 矛盾の指摘, 示唆などを通してインタビューの受け手の心理的負担を軽減し, よく考えさせることが有効と考えられている.

本論文ではこれらの指摘, 示唆を総称して「良い刺激」と呼んだ. そして良い刺激を生成できることをインタビューシステム試作の目標とした. 良い刺激の生成法として以下の 3 項目を検討した.

1. 静的解析の利用

2. 動的解析の利用

3. 学習の利用

1 はあらかじめインタビューシステムにいくつかの質問戦略を持たせておき, これを用いて良い刺激を生成しようとするものである. 2 は, インタビューを通じて得られた知識を使って見ることを通じてインタビューの受け手によく考えさせることを試みたものである. 3 はインタビューシステムに学習機能を持たせ, インタビューを通じて獲得した知識を後のインタビューで利用することを試みたものである.

さらに, インタビューシステムの試作・検討を通して, インタビューシステムのための汎用アーキテクチャを設計することを試みた. インタビューシステムに関する研究は現在非常に少ない. アーキテクチャの設計を通して, インタビューの聞き手に必要な機構・知識を一般的に検討することを試みた.

以下, まず 第2章 で静的解析 について検討する. ここでは同時にインタビューの例題として取り挙げたデータベースの論理設計についても概説する.

次に 第3章 で動的解析について述べる. 本研究ではデータベースの開発を例題としたので, 獲得した知識から実際にデータベースを試作, 使用することによって, インタビューの受け手に良く考えさせることを試みた.

第4章ではインタビューシステムの学習機能に関して検討する. 実体と関連に注目するデータベースの設計法に着目することによって, インタビューを通じて獲得した知識を有効に再利用できる機構を示す.

最後に 第5章 で汎用のインタビューシステムアーキテクチャを設計, 検討する. 良い刺激の生成が, インタビューシステムに問題解決および学習機能を持たせることによって一般的に実現可能であることを示す.

以上述べたように, 本論文は様々な分野で重要な役割を果たすインタビューに関連してインタビューを行なうのに必要な知識および機構を考察したものである.

第 2 章

静的解析を用いたインタビューシステム

2.1 緒言

本章では, 質問戦略を用いるインタビューシステムについて述べる.

質問戦略の考え方を明確にした初めてのインタビューシステムは, MORE [Kahn85] である. MORE は診断型エキスパートシステムの知識獲得支援システムとして開発されたもので, 8 個の質問戦略を獲得した知識に適用することによってインタビューを進める.

質問戦略の考え方は, インタビューに必要な知識を明確にできるという点で重要である. また知識の明確化は, 機構の明確化にもつながる. すなわち質問戦略とその適用機構の検討は, インタビューシステムに必要な知識および機構の明確化につながる.

本章では, データベースの論理設計を行なうためのインタビューに関する質問戦略を検討する. また, 質問戦略を取り扱う機構を検討し, その一般化を試みる.

2.2 節で, 例題として取り挙げたデータベースの論理設計について概説する. 2.3 節で, 試作したデータベースの論理設計を行なうインタビューシステム *I²S-LD* (Intelligent Interview System for Logical Design of database) について述べる. 2.4 節では, MORE と *I²S-LD* との比較検討結果から開発されたインタビューシステムのためのシェル, *SIS* について述べる.

2.2 データベースの論理設計

本節では, インタビューの題材として取り上げたデータベースの論理設計について概説する. また, 論理設計の困難な点について述べる.

データベースの設計作業は, 次のようにいくつかの段階にわけることができる [穂鷹81].

Step 1 発注者の情報要求 (データベースに対する要求) を分析する.

Step 2 分析をもとに現実世界の様子を決められた形式にしたがって記述する.

Step 3 得られた記述から論理的な (ファイル) 構造を決定する.

Step 4 論理構造から物理的なファイル構造を決定する.

Step 3 までが, 一般にデータベースの論理設計と呼ばれる.

Step 1 は設計しようとしているデータベースがどのような質問に答えられなければならないかを発注者に列挙してもらい, 整理する段階である. これをもとに Step 2 で, 要求に答えられるように現実世界を記述する (記述されたものは現実世界のモデル, あるいはドメインモデルと呼ばれる). この記述にはデータモデル[1]が使われる. データモデルとしては, 関係モデル (Relational Model) [Codd70], 実体-関連モデル (Entity-Relationship Model, 以下, ER モデルと略す) [Chen76] などさまざまなものが提案されている.

Step 3 で実際のデータベースの論理的なファイル構造を決定する. 得られた構造の定義を概念スキーマと呼ぶ. データモデルとして関係モデルを, またデータベース管理システム (Database Management System, 以下 DBMS と略す) として関係型 DBMS を使用する場合は, ドメインモデルと概念スキーマがほぼ同じになる. また ER モデルも関係型 DBMS との親和性が高い.

Step 1, 2 は, 実際には並行して行なわれる. すなわち要求分析結果とドメインモデルを常に比較, 検討しながら作業を行なう. 具体的には要求分析結果をもとに現実世界の概念[2]を抽出していく. またこれらを関連付けしてドメインモデルとしていきながら, 情報要求と照らし合わせてその妥当性を確認する.

論理設計の難しさは, 情報要求の分析およびドメインモデルの構築時に, 現実世界の概念とそれらの間の関連を網羅しなければならないところにある. 一般にデータベースの発注者は, 自分がデータベース化したい現実世界に関して十分に整理することができない. このため発注者から提示される情報要求は, 不完全で必要な概念を網羅していないことが多い. 逆に言えば, データベースの設計者はこの不完全な情報要求から出発して, 必要とされる概念をすべて洗い出す必要がある. これがデータベースの論理設計を難しいものとしている理由である.

実際の論理設計は, 発注者が用意した材料をもとに必要に応じて設計者がインタビューを実施しながら行なわれる. このインタビューでは, 論理設計に必要な現実世界の概念に関する知識が対話を通じて発注者から設計者に移される. 材料としては,

- 実際のデータ
- 現実世界を描写した文章
- データベースに答えさせたい事柄を記述した文の集合

などが使われる. 実際のデータの場合は, 個々のデータから概念や関係を抽出していかねばならないため, 一般に設計者の負担が大きい. また文章の場合は, 自然言語の持つ曖昧性なども問題になってくる.

[1] この言葉は現実世界の記述 (モデル) と非常にまぎらわしい. データモデルそのものはあくまで現実世界を記述するための道具である. 本論文では, 現実世界を記述したものをドメインモデル (Domain Model) と呼ぶ.

[2] データベース化する際に現実世界の中で認識される名前を持った個々のものという意味でこの言葉を使っている.

必要な概念とそれらの間の関係を網羅するために, 設計者は用意された材料を可能な限り分析しなければならない. また自身がそれまでの経験で得た知識の有効利用も必要である. そしてそれらをもとに発注者と質疑応答を行ない, 必要と思われる知識の獲得を試みていく. この際, 発注者が考え易くなるような示唆, 提案を行なっていくことによって有益な知識が得られ, 結果としてより良い論理設計が可能となる.

2.3 I^2S-LD

2.3.1 概要

I^2S-LD は, データベースの論理設計支援を対象とした知的インタビューシステムであり, ユーザへインタビューを行ないながら論理設計を行なう. すなわち, I^2S-LD はデータベース設計の専門家の振舞いをシミュレートするシステムである.

I^2S-LD は, 以下に示す順序でインタビューを実施する.

1. 構築するデータベースの名前を尋ねる.
2. そのデータベースで処理したい検索要求文あるいはドメインにおける事実 (以下要求文と呼ぶ) を尋ねる.
3. 要求文を解析して得られた情報をもとに, そのデータベースのドメインをモデル化する.
4. 3 で生成された attention[3] (対話中にシステムが気付いた疑問点) をもとにインタビューを繰り返して, ドメインモデルを洗練する.
5. ドメインモデルから論理設計を行なって, 概念スキーマを出力する. (関係型 DBMS を使用することを前提としている)

I^2S-LD との対話例を図 2.1 に示す. これは, 製造関係の会社のデータベースを設計する際のインタビューの例である. なお, 図中にみられるように, I^2S-LD との対話は簡単な英語を用いて行なわれる. ">" に続く下線を付された数字あるいは英文が, ユーザの入力である.

(1) で設計しようとしているデータベースの名前を, また (2)(3) でユーザが用意した要求文を入力してもらっている. (4) から (7) にかけて入力された要求文をもとにインタビューを行ない, (8)(9) でデータベースの概念スキーマを設計, 出力している.

対話例中, (4)(5) では入力された要求文についての質疑応答が行なわれている. また, (6)(7) でそれぞれ部品 (part), 会社 (company) に関する質疑応答が行なわれている. これらの詳細は後述する.

[3]attention はインタビュー時にシステムが気付いた問題点, 矛盾, 誤り, 示唆すべき点などを指す. attention は I^2S-LD で重要な役割を果たしており, 本論文ではあえて邦訳を試みなかった.

```
Please enter the name of your database.                          ┐
> makerDB.                                                       │
Do you want to create a new database, makerDB ?                  ├ (1)
> Yes.                                                           ┘
Now, let's start designing conceptual schema of makerDB.         ┐
Please enter an information requirement.                         │
> I want to know the names of parts which are constructed       │
 from other parts.                                               │
Is 'part' an instance of an entity ?                             ├ (2)
> No.                                                            │
Do you want to know the names of part-1's which are              │
constructed from part-2's ?                                      │
> Yes.                                                           ┘
Please enter the next information requirement.                   ┐ (3)
> No more.                                                       ┘
Let's discuss the following fact:                                ┐
F1:Part-1's are constructed from part-2's.                       │
I think you should store the information about following         │
verbs which are possible to precede 'construct'.                 │
  1.buy 2.sell 3.make 4.ship 5.lend 6.stock 7.supply ···         │
Do you agree ?                                                   │
> Yes.                                                           │
Which do you want to store ?                                     ├ (4)
> 7.                                                             │
Please enter the information requirement about 'supply'.         │
> Some parts are supplied from other companies.                  │
Is 'company' an instance of an entity ?                          │
> No.                                                            │
Do you want to store the information about                       │
  "Part-1's are supplied from company-1's" ?                     │
> Yes.                                                           ┘
Let's discuss the following facts:                               ┐
F1:Part-1's are constructed from part-2's.                       │
F2:Part-1's are supplied from company-1's.                       ├ (5)
Is the next fact true ?                                          │
  F:Part-1's in F1 are constructed from part-1's in F2.          │
> Yes.                                                           ┘
Let's discuss the following noun:                                ┐
  N:Part.                                                        │
I think it's one of the physicalObject. ····· ①                  │
Is it right ?                                                    ├ (6)
> Yes.                                                           │
Please choose the attributes of the part from:                   │
  1.size 2.weight 3.cost 4.price 5.(other) ····· ②               │
> 2, 3.                                                          ┘
```

図 2.1 I^2S-*LD* との対話例 (その 1)

```
Let's discuss the following noun:
  N:Company.
I think it's one of the individual or organization.
Is it right ?
> Yes.
Which does the company belong, individual or organization ?
> Organization.
Please choose other attributes of the company from:
  1.address 2.(other)
> 1.
Ok, I start designing conceptual schema.  Wait for
a moment, please.
Designed conceptual schema is as follows:
  construct(part-1,part-2)
  supply(part,company)
  part(name,weight,cost)
  company(name,address)
```

(7) (8) (9)

図 2.1 I^2S-LD との対話例 (その 2)

I^2S-LD の構成を図 2.2 に示す. システムは大きく分けて 4 つのモジュールから構成されている. ユーザインタフェース (interface), 知識ベース (KB), コントローラ (controller) およびメモリー (memory) である. これらのモジュールの内, メモリーが I^2S-LD を特徴付けている.

メモリーは, domain memory, attention list および working memory(WM) からなる. domain memory は, ユーザから聞きだしたドメインに関する知識の格納に用いる. attention list は, 対話中にシステムが気付いた注意事項 (attention) の一時記憶に, また WM は attention を処理する作業時に使われる.

2, 3, 4 の処理手順をフローチャートで示したのが図 2.3 である. 入力処理でドメインに関する知識の内部表現を構成して, domain memory に格納する (Process Query). また同時に attention が生成され (複数個), attention list に格納される (Generate Attention). 要求文の入力が終了すると, attention list 内の attention を順に処理して質疑応答を行なう (Extract Attention, Process Attention). 以上を繰返し行ない, 質疑応答を通じてユーザが別の要求に気付かなければ (Another Query ?), 最後に論理構造の決定 (5) を行なう.

個々の attention が, I^2S-LD が持つ質問戦略の一つ一つに対応する. すなわち, I^2S-LD ではまず attention を生成し, 次にそれを処理するという 2 段階を経て一つの質問戦略に基づく質疑応答が行なわれる.

2.3.2 プラン構造

I^2S-LD では, インタビューによって獲得されるドメインに関する知識は, プラン構造として表現される.

プラン構造では, ドメインにおける個々の活動と実体はそれぞれ動詞フレーム, 名詞フ

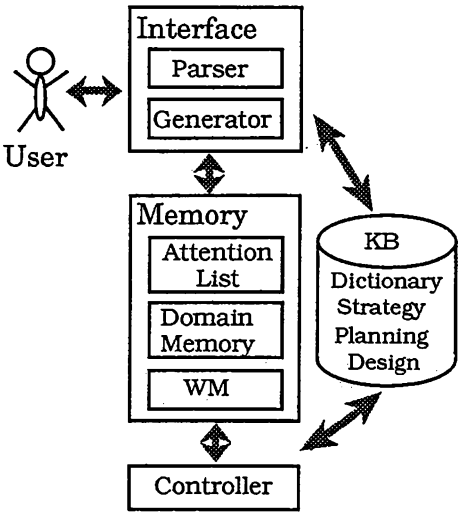


図 2.2 I^2S-*LD* の構成

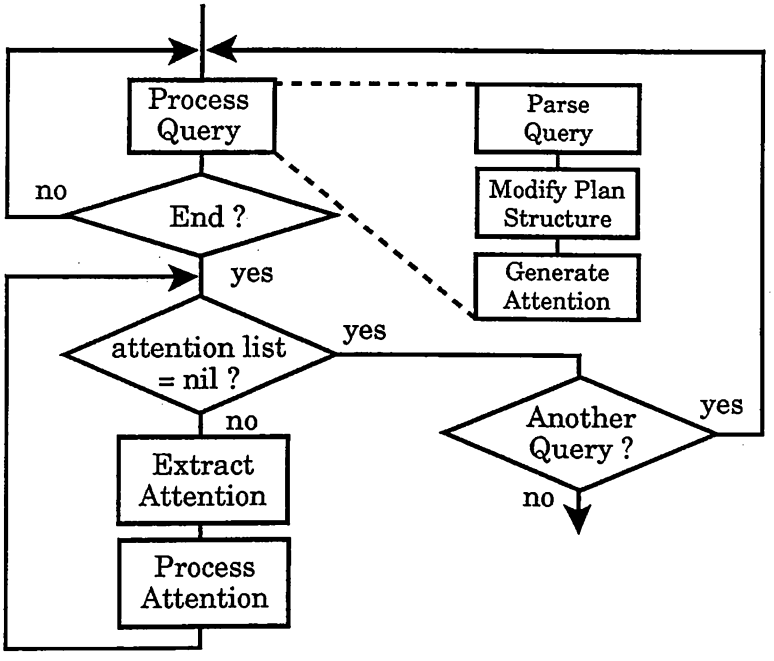


図 2.3 I^2S-*LD* の質疑応答処理

レームで表わされる. 動詞の意味表現にはプランの概念が導入されている. 名詞フレームにはその名詞の属性および上位概念が記述されている. これらのフレームをポインタで関係付けることによって, ドメインのモデル化を行なっている. また, コンテントフレームはドメインに含まれる動詞と名詞のリストを持ち, ドメインモデル全体を要約している.

プラン構造の例を図 2.4 に示す. この例は, 図 2.1 の対話例に対応するものである. 四角で表されたものがフレームである. コンテントフレーム (content) には, 設計しているデータベースの名前 (name) が makerDB であり, 2 つの動詞 (verb) と 2 つの名詞 (noun) が含まれていることが記されている.

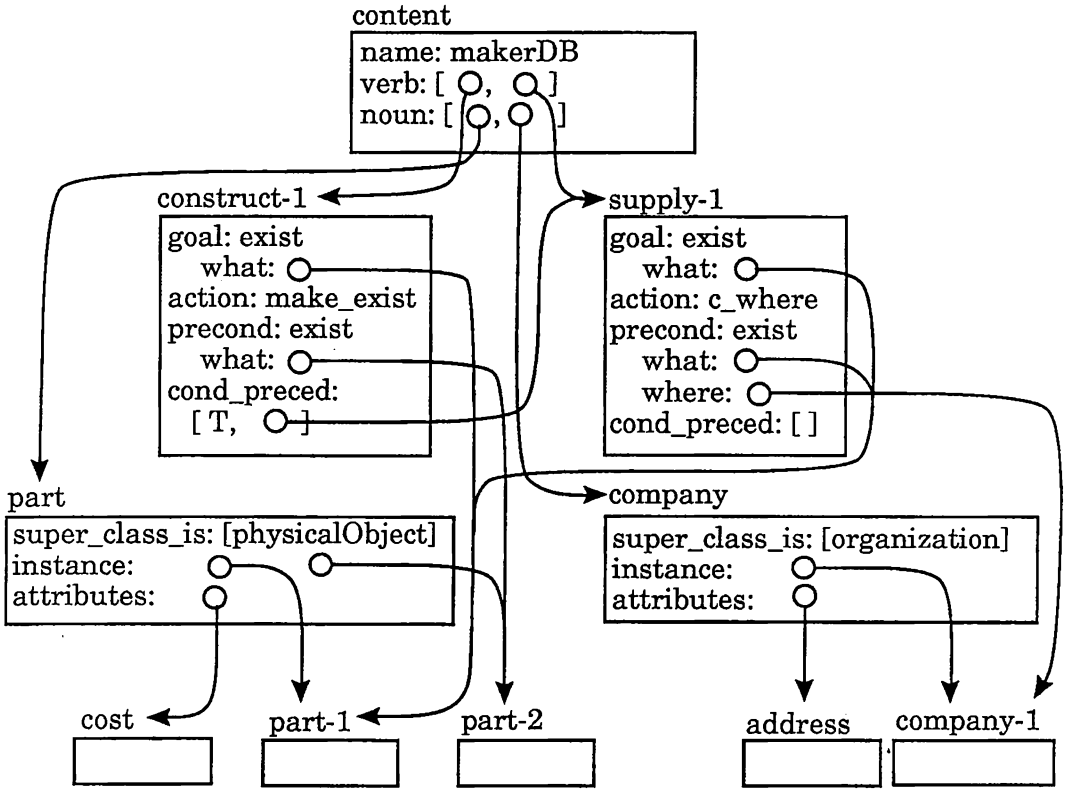


図 2.4 プラン構造の例

「部品は, 会社から納入された部品から組立てられる」という活動は, construct-1 と supply-1 という 2 個のフレームで表されている. 一方,「部品」に関しては part というフレームがその属性 (attribute) を表現している.

動詞フレームは 3 つのスロット goal, action および precondition を持っている. 各スロットはそれぞれ, その動詞が示す行為が達成する状態, その動詞が示す行為, その動詞が示す行為を実行するために達成されていなければならない状態を示している. 各スロットを埋める要素として 10 個の意味素を定義している (表 2.1).

表 2.1 動詞の意味素

primitive	description
for goal and precondition	
exist	someone or something exists
not_exist	someone or something does not exist.
connect	relation such as "contract" exists.
for action	
make_exist	make someone or something exist.
make_not_exist	someone or something not exist.
make_connect	make connection such as "make a contract".
make_not_connect	make connection not exist.
change_how_many	change quantity
change_where	equivalent to "move", etc.
change_state	change state of something.

動詞フレームの例を図 2.5 に示す. これは,「組み立てる (construct) 」という活動に対応する動詞フレームである. goal スロットの exist:what: に part-1 が, precondition スロットの exist:what: に part-2 が記述されている. これは「部品 1 は, 他の部品 2 から組み立てられる」という活動を示している.

construct-1

```
goal: exist:
   what:part-1
action:make_exit
precondition: exist:
   what:part-2
cond_preced:
   [ T, supply-1]
```

図 2.5 動詞フレームの例

ある動詞フレーム A の goal と他の動詞フレーム B の precondition が「A の goal は, B の precondition の一部をなしている」の関係にあるとき,「A と B は接続可能である」という. さらに実世界で「ある行為 A を行為 B より前に行なわなければならない」ことが事実のとき,「A は B に先立つ」あるいは「B は A に従う」という. 一般には, 二つ以上の動詞を順に接続することが可能である. そこでプラン構造では, 一つ以上の (接続された) 動詞フレームの集まりをプランと呼び, プラン間で「先立つ」「従う」といった接続関係を考える. そしてあるプランが与えられたとき, それに先立つプランあるいはそれに従う動詞を探索することをプランニングと呼ぶ.

動詞フレームでは図 2.5 のように, cond_preced スロットに supply-1 を記述することにより, supply-1 が construct-1 に「先立つ」動詞であることを示している.

この様に, 動詞の表現にプランの概念を導入し, プランニングを行なうことによって, I^2S-LD はドメインでの活動を推測することが可能である. インタビューにおけるプランニングの使用については, 2.3.4 で述べる.

名詞フレームはその上位概念を示す super_class_is スロットと属性スロットからなる. 名詞フレームの例を図 2.6 に示す. これは,「部品 (part) 」という実体に対応する名詞フレームである. super_class_is スロットには, 知識ベース中の他の名詞フレームへのポインタが埋められる. すなわち, ある名詞はシステムにとって既知の名詞のサブクラスとして表現される. 属性スロットは, その名詞が示すものが持つ固有の性質を示す.

part

super_class_is:[physicalObject]
attributes:[weight,cost]

図 2.6 名詞フレームの例

例えば, 図 2.6 の部品 (part) フレームの場合, super_class_is スロットに physicalObject[4] (次節で説明する抽象概念の一つ) が記述されているので,「部品」の上位概念は,「物理的な物」であることがわかる. また, 属性 (attributes) スロットに weight と cost が記述されているので,「部品」の属性は,「重さ」と「原価」であることわかる.

2.3.3 知識の分類

インタビューは, 受け手の持っている知識の移行を目的としている. 従って, インタビューシステムにおいてはシステムがあらかじめ持っている知識とインタビューを通じてユーザから獲得する知識との分類は重要な問題である. 本項では, I^2S-LD が取り扱う知識について述べる.

I^2S-LD における知識を表 2.2 に示す. I^2S-LD は, ドメインに依存しない汎用な支援を目標としている. 名詞の意味はドメインに依存するので, 本システムの基本方針として, 名詞を未知概念として扱い, それ以外の知識をシステムが持つことは制限しないとしている.

名詞の辞書を持たない代わりに, システムにはドメインに独立な 17 個の抽象概念に関する知識があらかじめ用意されている (図 2.7). そして, インタビューで獲得した新しい名詞フレームを抽象概念のフレームに下位概念として登録していく. また, 個々の抽象概念フレームには, 候補属性スロット attr_cands があり, その抽象概念フレームの下位概念となっている名詞がもつ属性の候補が記述されている. なおあらかじめ用意しておく抽象概念は, 文献 [国研64] をもとに決定した.

[4]文中, 必要に応じてシステム内で使われているシンボルをそのまま用いている. その場合, 単語の区切りを大文字で示している. この場合は physical object を一つのシンボルとして, physicalObject で参照している.

表 2.2 知識の分類

あらかじめ持っている知識	インタビューで獲得する知識
名詞以外の単語辞書 17 個の抽象概念フレーム 動詞の辞書フレーム 構文知識 質問戦略 プランニングに関する知識 論理設計用の知識	プラン構造 各実体 (名詞) の概念関係 各実体 (名詞) が持つ属性

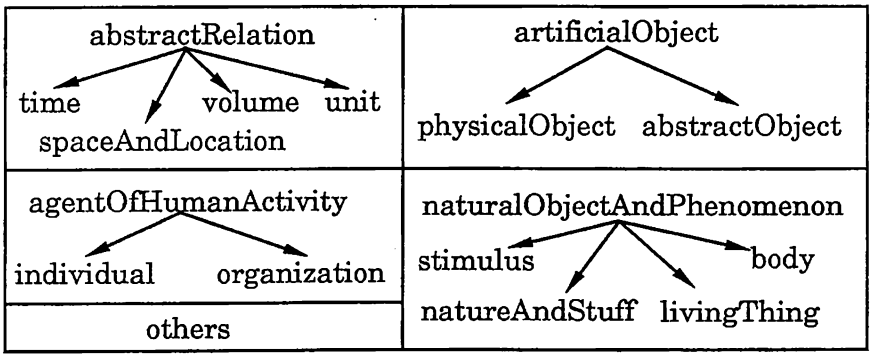


図 2.7 用意されている抽象概念

用意されている抽象概念は, 4 つの概念階層に大きく分けることができる. すなわち,「抽象的な関係 (abstractRelation) 」,「人工のもの (artificialObject) 」,「人間活動の主体 (agentOfHumanActivity) 」,「自然物および自然現象 (naturalObjectAndPhenomenon) 」の 4 つである. また, 以上のいずれにも属さない概念をデータベース化せねばならないときに備えて,「その他 (others) 」という概念を用意している.

抽象概念フレームの例を図 2.8 に示す. これは,「物理的な物」という抽象概念に対応する抽象概念フレームである. 候補属性 (attr_cands) スロットに記述されている size, weight, cost, color は, この抽象概念フレームの下位概念である名詞の属性の候補として,「大きさ」,「重さ」,「原価」,「色」がシステムに用意されていることを示している.

```
physicalObject
super_class_is:[nouns]
attr_cands:[size,weight,cost,price]
```

図 2.8 抽象概念フレームの例

動詞に関しては, システムにはドメインに独立な動詞の辞書フレームがあらかじめ用意されている. 動詞の辞書フレームを図 2.9 に示す. 動詞の辞書フレームには, 候補概念 (value_class) スロットがあり, 動詞フレームの各スロットに入る概念の候補が記述されている. その初期値としてそれぞれ抽象概念のリストが記述されている.

図 2.9 に示した例は,「組み立てる (construct) 」という動詞に対して用意された動詞の辞書フレームである. 候補概念 (value_class) スロットの goal の exist:what: に記述されている physicalObject は,「組み立てる」という動詞フレームの goal スロットの exist:what: に入る名詞が抽象概念「物理的な物 (physicalObject) 」 の下位概念である可能性が高いことを示している.

```
construct
super_class_is:[verbs]
value_class:
  goal:
    exist:
      what:[physicalObject]
  precondition:
    exist:
      what:[physicalObject]
```

図 2.9 動詞辞書フレームの例

インタビュー中に現れた新しい名詞 (例えば図 2.1 の部品 (part) など) は, 関連する動詞フレームの候補概念スロット中の記述を手掛かりにして, その名詞の上位概念をユーザ

にまず尋ねる (図 2.1 (6)①). そしてその名詞のフレームをユーザが示した上位概念のフレームの下位概念として生成し, 上位概念の属性 (上位概念が抽象名詞の場合は候補属性) を提示して新しい名詞の属性をユーザに尋ねる (同②). また, 内部では, 関連する動詞の辞書フレームの候補概念スロットにその名詞を加える.

2.3.4 質問戦略

I^2*S-LD* には 4 種類の質問戦略があらかじめ用意されている. *I*^{2}*S-LD* はこれらの質問戦略に基づいてインタビューを実施する. それぞれの質問戦略は, ドメインモデル内のプラン構造が持つ情報を豊かにする方向へシステムの動作を誘導するように設定されている.

I^2*S-LD* は, ドメインモデル内に不足している情報や曖昧な情報の存在を見つけると, システムの疑問点として attention を生成する. そして生成された attention を処理して行くことによって受け手と質疑応答を行ない, ドメインモデルを洗練していく. このように一つ一つの attention は, 個々の質問戦略に対応している.

表 2.3 に *I*^{2}*S-LD* が質疑応答時に用いる質問戦略を示す. 表中, attention はその戦略に対応する attention を, また格納法はその attention が生成された際の attention list への格納法を示している. この格納法は, attention の処理順, すなわち質問の順序に関与しており, 次のように定義される.

interrupt attention list へ格納せず, ただちに処理する.

push attention list の先頭に格納する.

enque attention list の最後尾に格納する.

すなわち, interrupt, push, enque の順で先に処理 (質疑応答) する.

質問戦略は以下に示すように大きく 5 つに分類される.

1. プランニング関連
2. 詳細度の統一
3. 名詞の取扱い
4. 属性名詞
5. その他

1, 2 は, プラン構造中の動詞フレームおよびそれらの接続に関して質問を行なう戦略である. 3 は名詞フレームおよびそれらの概念関係に関するものである. 4 は属性になりやすい名詞に注目することにより, 動詞の示唆などを行なう戦略である. 5 としては, 時間に関する質問を行なう戦略がある. 通常のデータベース管理システムでは一般に時間に関するデータを取り扱うのが困難なため, この戦略を用意してある.

戦略 1 は, プランニングを用いて, 未入力動詞 (ユーザがまだ言及していない活動) の必要性の示唆あるいは既知の動詞 (ユーザがすでに言及した活動) の接続を図るものである.

表 2.3 質問戦略と attention

質問戦略	attention	格納法
動詞フレーム単体について		
• precondition と goal の詳細さを揃える.	check_detail	enque
• 時間に関する曖昧さを解消する.	ambiguity_period	interrupt
• 再帰性を調べる.	check_recursive	enque
• 接続可能な未入力動詞を探す.	search_unknown	push
プラン構造内の動詞について		
• プラン構造内で接続可能な動詞の組を探す.	search_known	push
• 接続している動詞間の詳細さを揃える.	connect_detail	enque
• 接続している動詞間の接続条件を調べる.	connect_cond	enque
名詞について		
• 未知名詞ならば, 上位の抽象概念に登録し, 属性を検討する.	map_unknown	enque
• 既知名詞ならば, 知識ベース中の名詞 (抽象概念を含む) に instance として登録し, 属性を検討する.	map_known	enque
属性名詞について		
• when_attr スロットの verb ファシットを用いて未知動詞を示唆する.	suggest_attr_v	enque

一般に, ユーザが用意した要求文がデータベース化する必要がある情報すべてについて言及していることは希である. この戦略によってユーザに刺激を与え, ドメインに関する知識をさらに抽出することができる. 図 2.1 の (4) に, 未入力動詞の必要性の示唆の対話例を示した.

戦略 2 は, プラン中の goal と precondition が持つ情報量を揃えようという戦略である. *I*^{2}*S-LD* が行なう論理設計では, 一つの動詞フレームが一つの関係 (テーブルあるいは表) に対応する. そのため, この戦略に従うことによって完成したデータベース中の各テーブル間での結合演算が行ないやすくなる.

戦略 1, 2 の例を図 2.10 を用いて示す. 図の construct-1 フレームは, ユーザが「1 個の部品 A は, 他の部品 B から組立てられる」という要求文を入力した際に作られる. プランニングを用いると, この例では動詞 construct-1 に先立つ動詞として construct[5], buy, make, stock, supply などが見付かる (図中の①). 戦略 1 によって, このユーザが未だ入力していない (データベース化の必要性に気付いていない) これらの動詞に関する情報をデータベース化することをユーザに提案する[6].

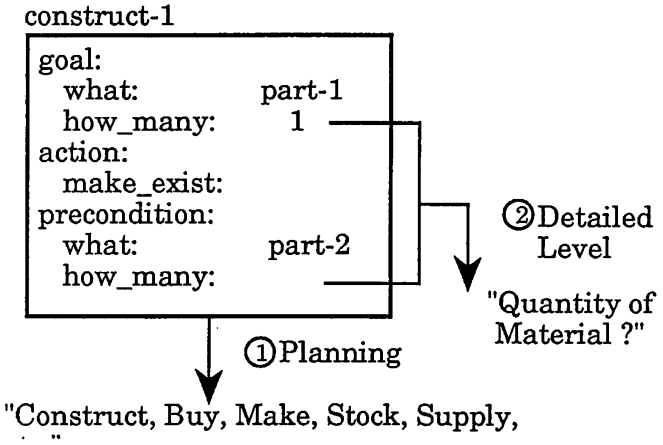


図 2.10 質問戦略の適用例

一方, goal および precondition を比較すると, 図に示すように数量 (how_many) に関して情報量に差があることがわかる. 戦略 2 によって, この例の場合材料である部品 B の数量をデータベース化するかユーザに問う (図 2.10 ②).

戦略 3 は, システムにとって未知の名詞と既知の名詞を取り扱う場合に分けられる. 未知名詞の場合は, まずあらかじめシステム内に用意された抽象概念のどれかに下位概念として登録する. 次にその抽象概念に関する知識を用いて質疑応答を行なう. どの抽象概念に登録するかは, 関連する動詞を手掛りとしてユーザに質問して決定する. 既知名詞の場合は, その名詞フレームを知識ベース中の対応する名詞フレームに実体 (instance) として

[5]プランニングの段階では, construct も自身に接続可能な動詞として求められる. もちろん質疑応答の際は提示されない.

[6]この例では先立つ動詞のみについて述べた. 質疑応答の際は従う動詞も求めるため, 実際の対話例 (図 2.1 (4)) では他の動詞も提示されている.

登録し同様に質疑応答を行なう. 未知名詞の登録に関する対話例を, 図 2.1 の (6)(7) に示した.

次に戦略 4 について述べる. ユーザが検索要求文を実際に入力するとき, 検索主体や実体の属性について入力することが多い. 属性も名詞であるが, ほとんどの場合実体にはならない. また属性は, ほとんどの場合名詞に付属するが, 中には対象ドメイン内での新しい活動を示唆するものもある. また, 名詞に付属する属性は, その名詞の属するべき抽象概念の候補について多くの手掛りを与えてくれる. I^2S-LD では, このような属性の性質に注目してユーザが入力した検索要求文を処理する戦略について検討, 実現した. ここで, 属性であるかないかは検索要求文を構文的に処理することによってほとんどの場合判断できる [Mizoguchi87].

検索要求文の属性の処理において, 人間のデータベースの専門家は次のように判断していると思われる. まず, 色 (color), 住所 (address) 等のように名詞の属性になりやすい属性については, そのまま名詞の属性にしてしまう. 例えば,

```
Find the color of parts which are supplied.
```

という要求文を示された場合, color が実体となるかどうか検討することはない. また, 場所 (place), 量 (quantity) 等は動詞の属性になりやすいので, まず動詞の属性と考えて, それが失敗すると名詞の属性ではないかと考える. 例えば,

```
Find the quantity of parts which are stocked.
```

という要求文に対しては, quantity は stock の属性であることを瞬時に決定する.

このように人間のデータベースの専門家は, 属性を検討するために,

- その名詞が属性になり得るかどうか
- 実体 (名詞) の属性になりやすいか, 関連 (動詞) の属性になりやすいか

という知識を用いていると考えられる. I^2S-LD ではこれを実現するために, 以下に示すような新しい知識を抽象概念及び動詞辞書に追加した.

- 属性になり得る抽象概念フレームには属性時 (when_attr) スロットを作る.
- 関連の属性になりやすい抽象概念フレームには, 属性時スロットの下に verb ファシット (facet[7]) を作る. そしてそこに, 動詞フレームのどのスロットに入るかという知識を持たせる.
- 実体の属性になりやすい抽象概念フレームには, 属性時スロットの下に noun ファシットを作る. そしてそこに, どの抽象概念の属性になりやすいかを記述する.
- 動詞を属性名の候補概念 (value class) によってクラス分け (階層化) を行なう. そして, 候補概念とクラスの間にポインタを張る.

[7] スロットの中のスロットとして働く

以上の知識を用いて I^2S-LD は, ユーザの検索要求文の構文解析の結果属性であると判った名詞 (以下これを属性名詞と呼ぶ) について, 次のような順番で処理をする.

1. その属性名詞が動詞の属性名かどうかを検討する.

   属性時スロットに verb ファシットがあるときは, 動詞の属性名である可能性がある. まず, verb ファシットから動詞に付属する時に入るべきスロット名をとってくる. 次に抽象概念—動詞クラス間のポインタをたどって, 属性名詞を取り得る属性の候補として持っている動詞 (以下候補動詞と呼ぶ) を見つける. 検索要求文に現れる動詞 (以下主体動詞と呼ぶ) がその候補動詞の中にあれば, 属性名詞を主体動詞の属性とする. 候補動詞の中にない場合または検索要求文に動詞が含まれない場合は, この候補動詞をユーザに提示して未入力の動詞の活動がないかどうかを尋ねる.

2. その属性が名詞の属性名かどうかを検討する.

   属性名詞フレームの属性時スロットに noun ファシットがあるときは, 名詞の属性である可能性がある. システムは, noun ファシットからその属性名詞が属性となり得る名詞の候補概念を得る. もし, 検索要求文で属性が付属している名詞 (以下これを主体名詞と呼ぶ) の概念が候補概念中にあるとき, 属性名詞を主体名詞の属性として扱う.

3. 1, 2 以外は名詞の属性として扱う.

このように属性に注目して処理することによって, 以下のように属性名詞を手掛りに新しい活動を示唆するが可能である.

```
Let's discuss the following query:
Q1:  Find the place of parts.
I think you should store the information about
  stock, store, ···
Do yor agree ?  > Yes.
```

この例の場合, 属性名詞である place の属性時スロットの verb ファシットに stock, store などが格納されている. そこでそのような動詞に place が関係しているのではないかと考え, 示唆を行なう. これは place という属性は動詞の対象である場所に関する概念を意味しているが, この検索要求文にはそのような動詞が含まれないため, stock や store などのように場所に関係する活動があるのではないかと示唆することに相当する.

2.4 *SIS*

経験豊富な知識工学者は様々なドメインで専門家にインタビューを実施し, エキスパートシステムを構築することができる. また テレビ番組等でもインタビュアーは色々な分

野の人々にインタビューを行なうことができる. このことから, インタビュアーはドメインに無関係なインタビュー技術 (知識) を持っていることがわかる. *I²S-LD* においてもドメインに独立なインタビュー知識として 4 種類の質問戦略が得られた.

このようなインタビュー知識は, タスク (task[8]) に依存しないプリミティブなインタビューの知識を組み合わせることによって表現できると考えることができる. 例えば *I²S-LD* の 4 種類の質問戦略と MORE [Kahn85] の 8 つの質問戦略には, 多くの共通点が見出される. このことはインタビューに関する, ドメインやタスクに依存しない汎用の枠組みの存在を示唆している. 本節ではこの考えから試作されたインタビューシステムのためのシェル, *SIS* (Shell for Interview Systems) について述べる.

インタビューシステム開発者は *SIS* を用いることにより, ドメインのメタ構造とインタビューのタスクを記述するだけで様々なインタビューシステムを構築することができる. インタビュー知識のプリミティブとして 7 つの質問戦略プリミティブモジュールが用意されており, これらを組み合わせることにより様々な質問戦略 (インタビューのタスクに依存する知識) を実現できる.

以下まず *SIS* の構成を述べる. 次に *I²S-LD* および MORE を *SIS* を用いてインプリメントした例について述べる.

2.4.1 *SIS* の構成

SIS は, インタビューシステム自身も一種のエキスパートシステムであるという考えに基づいて開発されたインタビューシステムを構成するためのシェルである. *SIS* を用いると, 必要な知識を記述するだけでインタビューシステムを構築することができる.

一般にインタビューは図 2.11 に示すような過程をとる.

(1) まず以後の対話のきっかけとなる初期情報 (initial information) を聞く. この時, 聞き手に不完全ではあるがドメインに関する知識 (domain model) が形成される. またそれと同時に様々な疑問が生じる.

(2) 次にそれらの疑問を解くためにインタビューの受け手と質疑応答 (Q&A) を行なう. この時別の疑問も生じ得るが, それらも順に解いていく.

(3) 質疑応答が終了すると, 最後にインタビューの目的物 (output) を出力する.

ここで, (1)(2) においてドメインに関する知識 (domain knowledge) とインタビューの進め方に関する知識 (interview knowledge) をインタビューの聞き手は用いていると考えられる. また, インタビュー中に生じる疑問が *I²S-LD* の attention に対応している.

例えば, 診断型のエキスパートシステムを構築するためのインタビューは, 次の三つのステップからなる.

---

[8]本論文では, そのインタビューが目的としている仕事のことをタスク, 個々のインタビューで対象となっている世界 (分野) のことをドメインと呼んでいる. 例えば *I²S-LD* の場合, タスクはデータベースの論理設計であり, ドメインは個々の発注者がデータベース化しようとしている現実世界である.

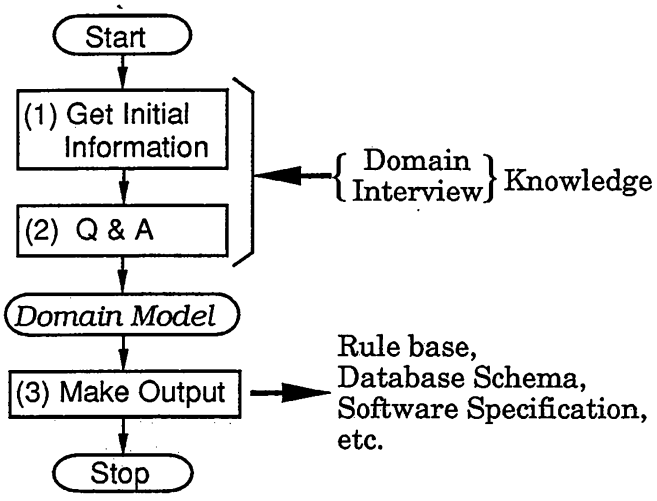


図 2.11 インタビューの流れ

Step 1 まず専門家から (不十分な) 仮説と徴候の集合を聞く (同時に attention が生じる).

Step 2 Step 1 で生じた attention (仮説と徴候の集合に関する疑問) について, 専門家と質疑応答を行なう.

Step 3 最後に (例えば) ルールセットを作成し, インタビューを終了する.

2.4.2 インタビューシステムの概念構造

前節の検討から図 2.12 に示すインタビューシステムの一般形が導かれる. *SIS* を用いて構築されるインタビューシステムはすべてこの図のモジュール構成になる. 各モジュールの機能は以下のとおりである.

(a) Interface

インタビューの受け手と自然言語, 図などを用いて対話するためのモジュールである. 受け手の応答からドメインモデルを構築する機能, ドメインモデルを提示する機能などがある.

(b) Domain model

ドメインモデルを格納しているモジュールである.

(c) Attention manager

attention の生成, 処理待ちの attention の管理などを行なうモジュールである.

(d) Supervisor

システム全体の制御をするモジュールである. attention に関する推論も担当する. 具体的には attention の内容に応じて受け手に質問を行ない, その答えに応じてドメインモデルを修正する.

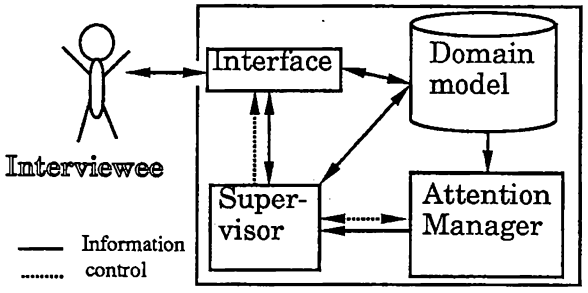


図 2.12 インタビューシステムの概念構造

個々のインタビューシステムは, 上に挙げた各モジュールがそのタスク向けに特化したものと考えられる. また特化の仕方を定義した知識としてタスクに固有のインタビュー知識を捉えることができる.

2.4.3 *SIS* の概要

SIS を用いて構築されたインタビューシステムの概念構造は図 2.12 に示したものと同じ構成になる. 各モジュールの動作の詳細を宣言的に記述することにより, 様々なインタビューシステムを構築できる.

図 2.13 に *SIS* と *SIS* を用いて構築されるインタビューシステムとの関係を示す. *SIS* ではインタビューシステムを概念的に以下の 3 層に分解している.

(1) インタビューシステム層 (Interview system layer)

この層では *SIS* を用いて構築されるインタビューシステムの概念上の構成を取り扱う. 図 2.12 に示した概念構造と同じモジュール構成である.

(2) 内部層 (Internal structure layer)

この層では個々のインタビューシステムは *SIS* 固有のモジュール群によって表現される. インタビューシステム層の各モジュールは, *SIS* に組み込みのモジュールを組み合わせたものとして実現される.

(3) 知識層 (Knowledge layer)

この層では内部層の各モジュールの動作をモジュールとその動作の詳細を宣言的に記述した知識の組で表現する. この層で記述された知識が内部層, そして最終的にはシステム層の各モジュールの動作を定義する. すなわち, 知識層でインタビューシステムの振る舞いが決定される.

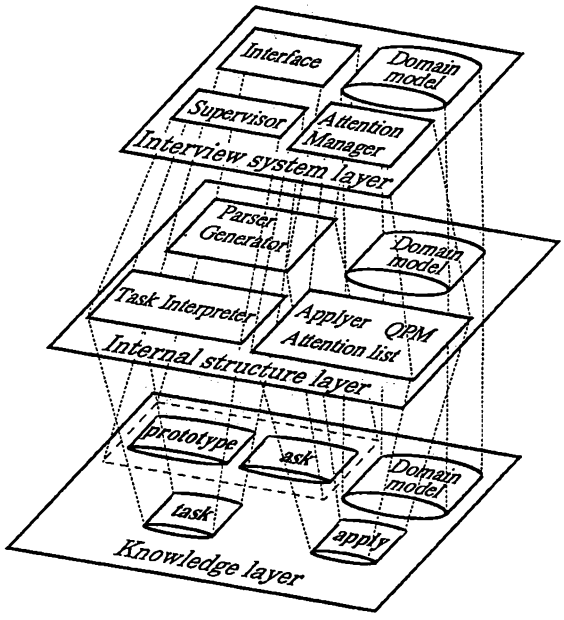


図 2.13 *SIS* とインタビューシステムの関係

内部層のモジュールとして現在以下のものが用意されている.

(a) parser/generator

システムと受け手とのインタフェースであり, 簡単な英文の解析・生成を行なう. どちらもデーモン機構 [Dyer83] に基づいており, 単語単位で解析・生成知識を記述できる.

(b) 質問戦略プリミティブモジュール (QPM (Question strategy Primitive Module))

attention 生成を担当するモジュールである. 現在, 7 つの QPM が用意されており, それぞれがドメインモデル内の特定の状況を検出, attention を生成する.

(c) applyer

QPM をドメインモデルに順に適用していくモジュールである.

(d) attention list

QPM が生成した attention を蓄える一種のキュー (Queue) である.

(e) task interpreter

システム全体の動作を制御するモジュールで, attention についての推論もこのモジュールが行なう.

(f) domain model

SIS はネットワーク形式で表現されるドメインモデルをサポートしている. 各ノード, アークはフレームで表現され, 任意の属性を持つことができる. またインヘリタンスの機能が提供されている.

7 つの質問戦略プリミティブモジュール (以下, QPM と略す) は, 質問生成の第一段階である attention 生成を担当している. QPM は, *SIS* が提供するネットワーク形式で表現されたドメインモデル用に 7 つ用意されており, それぞれはインタビューの対象としているドメインにに依存しない (この意味でプリミティブと呼んでいる).

以上のように各 QPM は, ドメインモデルに関してそれぞれに固有の条件を持っており, この条件が満たされると attention を生成する. 以下に 7 つの QPM とその固有条件を述べる.

(1) planner

あるノードからアークをたどって得られるパスおよびノードについて

- その パス に接続可能な概念が存在する.
- 一つのノードに対して複数のパスが存在する.
- 異なるノードに対して同じパスが存在する.

(2) script_tracer

ノードに対応する概念, あるいはその上位概念に, 受け手に聞くべきことが記述されている.

(3) unknown_object_detector

未知の概念が入力された.

(4) discord_detector

あるノードのスロットと他のノードのスロットの値が一致していない.

(5) ambiguity_detector

あるノードのスロットに値が二つ以上与えられた.

(6) certainty_manager

あるアークに付された信頼度がある範囲内にある.

(7) constraint_checker

あるノードのあるスロットの値が予め定義した制約を満たさない.

QPM はネットワークで表現されたドメインモデル中で, 情報の追加, 修正, 削除の候補となり得る箇所を検出するように用意されている.

SIS の動作, すなわち *SIS* を用いて構築されたインタビューシステムの動作は, 内部層において以下のように行なわれる.

(1) 受け手からの入力は parser によって解析され, 結果はドメインモデルに格納される.

(2) ドメインモデルを質問戦略プリミティブモジュールが走査し, attention を生成する.

(3) 生成された attention について task interpreter が推論し, 受け手に質問を発する.

2.4.4 インタビュー知識

以上に示した *SIS* の動作の内, 個々のインタビューシステムに依存する動作の詳細は, 知識層で記述された各 *SIS* モジュールの動作定義で規定される. このインタビュー知識は, *SIS* では以下の 4 種類に分けられている.

(1) prototype

prototype 記述によってドメインのメタ情報, すなわち,

(a) ノードおよびアークが持ちうる属性

(b) ノード間の is-a 関係

(c) 受け手から得た情報のドメインモデルへの格納法

(d) 英文への変換法

を定義する. インタビュー中に受け手から得た情報は, prototype を実体化したものとしてドメインモデルに格納され, ネットワークを構成する. prototype は parser/generator によって参照される.

(2) apply

apply 記述で各 QPM の適用対象を定義することにより, 構築するインタビューシステムが注目すべき情報を指定することができる. それらの情報について固有条件が満たされると, QPM は attention を生成する. apply は applyer および QPM によって参照される.

(3) task

task 記述によってインタビューの流れ, attention の処理法を定義する. ループ, 条件分岐, sub-task の呼び出しなどを記述できる. また, 質問法を定義した ask を呼び出すことにより, 受け手に質問をすることができる. task interpreter によって参照される.

(4) ask

具体的な質問作成法は ask 記述で定義される. 答えとして以下の 3 種を仮定した質問法を定義できる.

(a) 英文

(b) yes/no

(c) メニュー選択

ask は task interpreter および generator から参照される.

質問戦略は apply による QPM の適用対象指定と task による attention 処理定義の組で定義される. 次節で具体例を通して, これら 4 つの知識と *SIS* の動きについて詳細に述べる.

2.4.5 インタビューシステムの実現例

本節では, タスクおよびドメインがまったく異なる 2 種類のインタビューシステムの *SIS* による実現例について述べる. 例を通して *SIS* が持つ機能の有効性ならびに QPM の一つ, planner の一般性を示す[9].

(1) *I^2S-LD* の実現例

本節では *I^2S-LD* の質問戦略の内, プランニングを取り挙げ *SIS* におけるその定義例および動作を述べる.

SIS でプランニングを定義するにはまず, 必要と思われる動詞の意味を prototype として図 2.14 (a) に示すように定義する. attributes(①) で prototype が持つ属性が定義され, この例の場合 3 つの属性 (goal(②), action(③) および precondition(④)) が定義されている. また図中, <= および => に続くものは英文の解析および生成をするデーモン [Dyer83] である. これらは, parser/generator に備え付けのものである.

次に apply で, 質問戦略モジュールが作用すべき prototype およびその実体を定義する. プランニングについては図 2.14 (b) に示すように定義する. この定義により, 次の条件が成立するとき, foundCandidate という attention が生成される.

- プラン構造中のある動詞 (domain(verb)) の precondition と辞書中のある動詞 (prototype(verb)) の goal の形が同じである (⑤).

- プラン構造中のある動詞 (domain(verb)) の precondition と別のある動詞 (domain(verb)) の goal が同じ構造で値も同じである (⑥).

[9]機能的には planner が最も重要であるが, 実現例全体における使用頻度の点では script_tracer がもっともよく使われている.

```
prototype construct
  super_class verb
  attributes [ ..................................... ①
     [goal, [exist [what <= (searObjAft), ............. ②
                   how_many <= (searNumAft),
                   when <= (searTimeAft),
                   where <= (searPlaceAft)]],
     [action, [make_exist]], ......................... ③
     [precondition,[exit, [what <= (searObjAft), ..... ④
                   how_many <= (searNumAft,of),
                   when <= (searTimeAft),
                   where <= (searPlaceAft)]]]].
```

(a) construct の定義例

```
apply planner
  object
     prototype(verb)->goal &
     domain(verb)->precondition, } ................... ⑤
     domain(verb)->goal &
     domain(verb)->precondition. } ................... ⑥
```

(b) apply の定義例.

```
task planning
  attention foundCandidate, Path, [Cand]
  sub_task
     callAsk exist_link(Verb1, Cand),
     callAsk new_value(Cand).
ask new_value Verb
  showNl('Please enter an information requirement about ', Verb),
  get_answer Query.
```

(c) task と ask の定義例.

図 2.14 プランニングの定義例

そして, task および ask でプランニングに関する attention の処理を図 2.14 (c) に示すように定義する. まず task planning で, foundCandidate という attention が生成されたときはまず候補をユーザに提示し (callAsk exist_link), ユーザが「データベース化する」と答えたときはその候補に関する検索要求文を聞く (callAsk new_value) ように定義している. ask new_value はこの検索要求文を入力してもらうための定義である.

以上がプランニングに関する定義例である. 実際のインタビューでは, 発注者が情報要求文入力時に

```
I want to know the names of parts which are constructed
from other parts.
```

という事実 (図 2.1 (2)) を述べたとすると, それ以後システムは次のように動作する (図 2.15). まず, 各々の入力に対してプラン構造の各ノード (domain constituents) が parser によって作られ, ドメインモデルに格納される (1). これらに対して QPM が apply 定義に従って適用される (2). この例の場合, 検索要求文が先に示した attention 生成の条件 2.4.5 (1) を満たすため, planner の固有条件が満たされ, foundCandidate という attention (3) が作られる.

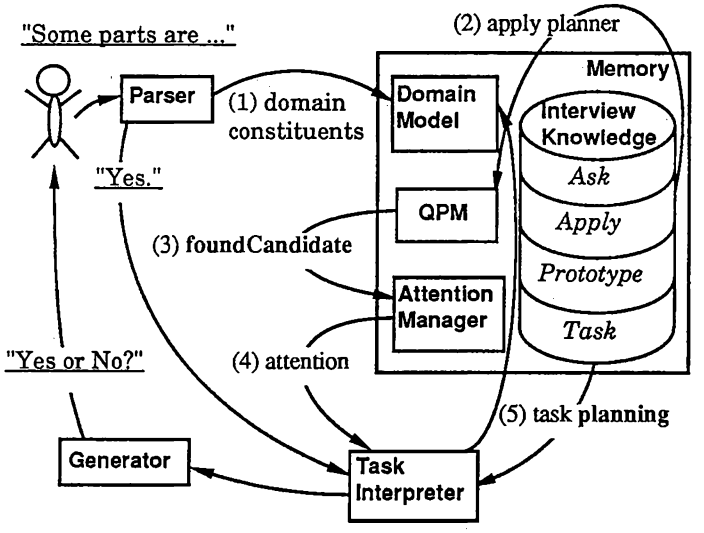


図 2.15 *SIS* の動作の様子

さて, 質疑応答に入ると, attention list から順に attention が取り出され (4), task interpreter に送られる. 例えば foundCandidate (3) については, 図 2.14 (c) の定義に従って (5), 図 2.1 (4) のような対話が行なわれる. ask 定義中の show は文の生成を, get_answer は答えの入力を示しており, *SIS* に組み込みの機能である.

なお, オリジナルの *I*^{2}*S-LD* は C-Prolog を用いて実装されており, ソースコードで約 1 万行になる. *SIS* を用いると約 1 千行で記述できる.

(2) MOREの実現例

MORE[Kahn85] は診断型エキスパートシステムを構築する際の知識獲得を支援するシステムである. MORE はまず, 専門家から仮説, 徴候, その他の条件などを聞き出し, ドメインモデルを構築する. MORE のドメインモデルは, 仮説, 徴候などをノードとするネットワークで, 各々のノードはリンクあるいはパスで結ばれる. 次にこのドメインモデルに 8 つの質問戦略を適用して受け手 (専門家) と質疑応答を行ない, ドメインモデルを洗練する. そして最後にドメインモデルからルールセットを作成する.

8 つの質問戦略の内, 基本となるのは仮説の区別 (differentiation) と呼ばれる戦略である. ある一つの徴候 S がある仮説 H1 を示唆し, 別の仮説 H2 とは関係がないとき, S は H1 を H2 から区別すると呼ぶ. 仮説の区別戦略は, 任意の仮説の組についてそのような徴候がないとき, それを聞くという戦略である.

例えば

"if Increase-In-Viscosity then Shale-Contamination"
"if Increase-In-Viscosity then Water-Influx"

という関係は, 図 2.16 (a) のドメインモデルで表わされる. このモデルでは, 徴候 Increase-In-Viscosity が観測されたとき, それが Shale-Contamination によるものか, Water-Influx によるものか区別できない. そこで仮説の区別戦略によって図 2.16 (b) のような対話を行ない, 新しい徴候 Increase-in-Unemulsified-Water を専門家から獲得する (図 2.16 (c)).

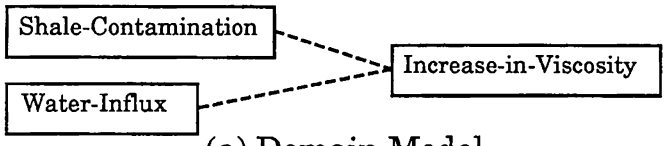
SIS では, planner を用いることによって differentiation 戦略を実現することができる[10]. まず, prototype でドメインモデルに現れる各実体を定義する. 次に仮説の区別戦略を適用すべきときに attention を生成するよう, apply で planner の適用対象を定義する. この戦略の場合, planner がある一つの徴候から複数の異なる仮説へのまったく同じパスを発見したとき, attention を生成するよう apply を定義する. 一方, そうして生成された attention の処理として, そのようなパスが発見されると, 見付かった (複数の) 仮説を区別する徴候がないかメモリーに問い合わせ, ない時は質問を受け手に行なう.

2.5 結言

本章では, インタビューの具体例として,「データベースの論理設計時の対話」を取り上げ, 開発した知的インタビューシステム *I²S-LD* について述べた. また, ネットワークで表現されたドメインモデルに質問戦略を適用するインタビューシステムのためのシェル, *SIS* についても述べた.

I²S-LD は, データベースの発注者にインタビューを実施し, データベース構築に必要なドメインに関する知識を獲得する. 獲得したドメインに関する知識は, プラン構造を用いて表現される. *I²S-LD* は様々な戦略を用いて, ドメインモデルを洗練し, 最後にデータベースの概念スキーマを出力する.

[10] 詳細は付録 A を参照.



(a) Domain Model.

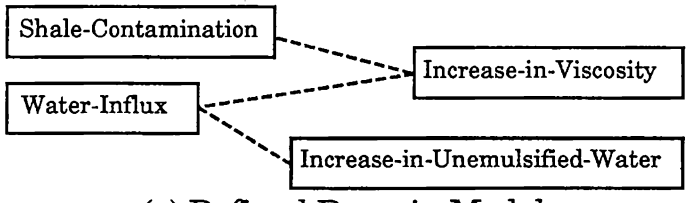
MORE: Increase-in-Viscosity can result from Water-Influx or Shale-Contamination. Can you provide a symptom associated with Water-Influx that cannnot be explained by Shale-Contamination ?

Expert: Yes.

MORE: Enter a symptom associated with Water-Influx.

Expert: Increase-in-Unemulsified-Water.

(b) Interview.



(c) Refined Domain Model.

図 2.16 仮説の区別戦略

I^2S-LDのインタビューの特徴は, インタビューの受け手からより多くの知識を獲得するために, 質問戦略を用いて受け手に「良い刺激」を与えるよう試みることである.

第 3 章

動的解析を用いたインタビューシステム

3.1 緒言

第 2 章では, データベースの論理設計を例に取り上げ, 質問戦略を用いてインタビューを進めるシステム, *I²S-LD* について述べた. またそのような静的解析を行なうインタビューシステムの一般化を試みた.

本章では, 動的解析を用いるインタビューシステムについて述べる.

静的解析は, インタビューの獲得対象となっている知識のメタ構造が整然としたもので, 十分な質問戦略をあらかじめ用意できる時に有効である. しかし, データベース化に必要なモデルやエキスパートシステムが必要とする経験則の獲得には, 次のような問題がある.

(1) 質問戦略を十分に用意することが困難である.

(2) 人間の場合, 質問されてもその場では答えられないことがある.

(3) 獲得した知識を実際に使ってみなければ, 気が付かない問題がある.

特に (3) は, 関係型データベースを用いた本研究では顕著であった. 関係型データベースの大きな特徴として, 結合演算を持っており, 検索操作が柔軟に行なえるということがある. このため, インタビュー後, 作成したデータベースを使っている内にモデルの不備が見つかるということがしばしばあった.

McDermott [Eshelman86] らは診断型エキスパートシステム開発のための知識獲得について同様の議論を行ない, インタビューのみで行なう知識獲得を静的解析 (static analysis), 実際にエキスパートシステムを試作して試用中に問題が生じたときに行なうものを動的解析 (dynamic analysis) と呼んだ. 彼らが開発した知識獲得支援システム MOLE はインタビュー終了後自動的にエキスパートシステムを試作し, 専門家が出した答えと試作システムの答えを比較しながらさらに知識ベースの洗練を試みる.

本章では, 動的解析の考え方に基づいてユーザからドメインに関する知識を獲得し, データベースの構築を支援するシステム, *I²S-DB* (Intelligent Interview System for DataBase design and construction) について述べる. システムは *I²S-LD* を中心に構築されており, まず質問戦略を用いたインタビューを行なってデータベースを設計・試作する. 次にその

試作データベースを実際にユーザに試用してもらう. 試用中に問題が生じると, ユーザにインタビューを実施し, ドメインのモデルを洗練, 試作データベースを改良する.

次節でまず, 動的解析の考え方について整理する. 3.3 節では, 後の節の準備としてデータベースの構築作業について概説する. 3.4 節で, 試作したデータベースの論理設計・構築を行なう動的解析を用いたインタビューシステム I^2S-*DB* について述べる.

3.2 動的解析

多くの場合, いくら良いインタビューを行なってもドメインの専門家が自分の持っている知識を整理できないため, 適確にインタビューに答えられない. この問題を解決するために, 質疑応答による知識獲得だけでなく, 獲得された知識の不十分な点を, その知識を実際に利用しながら動的に検出, 修正する方法が考えられる. 知識獲得時にわからないことは, とりあえず既定値 (default value) を用い, 知識の利用結果と専門家の結果を比較することによって獲得した知識の洗練を試みるのである. 3.2.1 で述べる MOLE [Eshelman86] は, この動的解析を用いた例となっている.

なお, 静的解析と動的解析で得られる知識は, 同じ種類のものであるということにここで注意しておかねばならない. 両者の違いは, インタビューの受け手 (専門家) がどのような状態にあるか, すなわちただ質問に答えようとしている受け身の状態であるか, あるいは問題を解こうとしている能動的な状態であるかという点である. 受け手によっては, 動的解析によらずとも自身の知識を述べることが可能であるという場合も存在する. したがって, 動的解析を行える知識獲得支援システムは, (静的解析を基礎として) 知識を獲得する時点についてより柔軟にしたものと言える.

動的解析を行える知識獲得支援システムでは, エキスパートシステム構築時にドメインの専門家からの知識獲得が不十分でも, 実行結果等をもとに知識ベースを洗練することが可能である. 次節では, 動的解析を行なう知識獲得支援システム MOLE を検討する.

3.2.1 MOLE

MOLE は MORE の欠点を克服するために構築されたシステムである. MORE は, 8 つの戦略を用いてドメインエキスパートにインタビューを行なった. しかし専門家は多くの場合, MORE の質問に答えることや支持値を割りつけることができなかった. MOLE は専門家に対する質問をできるだけ少なくし, そのかわり実際に診断を繰り返しながら専門家と対話することによって知識を引きだすことを試みる. この方法は, 実際に診断を行なう状況に置かれることによって, 専門家はより容易に自身の知識を述べられるだろうという考えに基づいたものである.

MOLE のドメインモデルは, 表 3.1 に示す 4 つの実体からなるネットワークである. このドメインモデルの構築には, 初版ドメインモデルの構築とそのドメインモデルの対話的な洗練プロセスがある. 初版ドメインモデル構築は仮説とそれに関連するイベントをエキスパートに尋ねることによって行われ, その後このドメインモデルの洗練が行なわれる. この洗練には静的解析と動的解析の二つの過程がある. まず, 静的解析により初版ドメイ

ンモデルの明確化を行ない, 弱点を発見し解決策を助言する. 次に, 動的解析により, ドメインモデル中の欠落知識や誤り知識を発見し解決策を助言する.

表 3.1 MOLE のドメインモデルの構成要素

構成要素	意味
H (Hypotheses)	診断される問題の原因・説明
S (Symptoms)	仮説から生じる事象・状態
PC (Prior Conditions)	仮説の生起確率に影響を与える条件
QC (Qualifying Conditions)	観測された徴候や事前条件の支持に影響を与える条件

動的解析は MOLE に実際に診断を実行させ, その結果をエキスパートの診断結果と比較することで行なう. もし, 両者に相違があれば, MOLE は以下にに示す洗練戦略に基づいて, 欠落知識の発見, 獲得と支持値の修正を試みる.

- ネットワーク中の支持値を変化させることによって, 専門家の診断結果に到達できる場合

  (1) 仮説に関する条件 (PC の獲得)
  仮説の支持値を上げる (下げる) 必要があって, その仮説に強い正 (負) の事前条件がない場合, 負 (正) の事前条件を尋ねる.

  (2) 背景条件 (QC の獲得)
  仮説の支持値を上げる必要があって, その支持値が強い負の値をとっているとき, その負の効果を妨げる背景条件を尋ねる.

  (3) 徴候の区別 (QC の獲得)
  ある徴候が支持する仮説が別の仮説であるべきとき, そうさせる識別条件を聞く.

  (4) 支持値の修正とネットワークの再解釈
  専門家が何ひとつ新しい情報を与えられなかった場合は, ネットワークの解釈と支持値の信頼度に応じて支持値を変更あるいはネットワークの一部を再解釈する.

- 支持値を変化させても, 専門家の診断結果に到達できない場合

  (1) 徴候の説明 (H の獲得)
  ある徴候を説明する必要があるために除外されるべき仮説を除外できない時, あるいはある仮説がある徴候の唯一の説明である時は, その徴候に別の仮説がないかを尋ねる. もしそのような仮説が与えられなければ, その徴候の信頼度は低いと見做し, 信頼度を低くする.

(2) 仮説の区別 (S の獲得)

除外されるべきではない仮説が除外されている場合は, その仮説が説明するが, ネットワーク内で関連付けられていない徴候がないか尋ねる.

MOLE の特色は, 診断システムの診断結果をフィードバックすることにより, 知識ベースを繰り返し洗練できることである. MOLE の適用事例としては, 圧延機の故障診断, コンピュータのチューニング問題, 発電機の故障診断等がある.

3.3 データベースの構築

本節では, データベースの一般的な構築法を述べる. 次にデータベースの構築において, データベースの非専門家にとって困難と思われる点を述べる. そして最後に, データベースの構築・利用支援と動的解析について考察する.

3.3.1 データベースの一般的な構築法

データベースの一般的構築・利用の手順を図 3.1 に示す. 以下, 個々の作業について述べる.

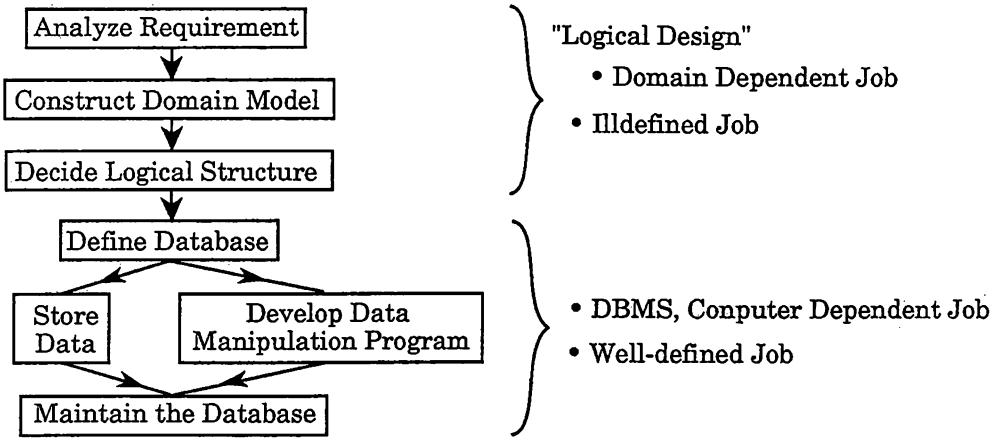


図 3.1 データベース構築作業の一般的手順

まず, 情報要求の分析 (Analyze Requirement) を行なう. これは構築したデータベースで何をしたいのかを整理することである. 次に, 分析した情報要求に答える現実世界のモデルを構築する (Construct Domain Model). そして実際のデータベースの論理構造を設計する (Design Logical Structure). ここでは先に構築したドメインモデルを, 使用する DBMS の表現法にしたがって記述し直す.

ここまでの作業は一般に論理設計と呼ばれている[1].

[1] 2.2 節参照.

次に, DBMS を使用して計算機システム上にデータベースの枠組を作る (データベースの創成, Define Database on Computer). これは実際のデータが入っていないデータベースであり, データを入れるファイルを用意したことになる.

そして, 実際のデータをこのデータベースの枠組に格納する (Store Data). 多くの DBMS では, この作業を行なうためにデータ格納用のプログラムを記述する必要がある.

データの格納を終了した後, データベースにあるデータを操作するためのプログラムを記述する (Develop Data Manipulation Program) ことで, 実際に運用できるようになる. またユーザの利用による問題点の検出とその修正, データのバックアップ, あるいはシステムがダウンした際の回復処理等の保守作業が以後必要とされる (Maintain the Database).

3.3.2 構築・利用の難点

パーソナルコンピュータを用いる時のように, データベースについて専門的知識を持たない一般利用者 (以後, 一般構築者と呼ぶ) が, 自らデータベースの構築とその利用を試みる際に遭遇する困難を以下に列挙する.

1. 情報要求の分析
   - 自分が何をどのようにデータベース化したいのかが, よくわからない
2. データのモデル化
   - データに潜在する論理構造を把握できない
   - 情報要求に適合するモデル化がうまくできない
3. 論理構造の設計
   - 使用する DBMS に関する知識が十分でない
4. データベース創成
   - DBMS の利用法がよくわからない
5. データの格納および操作
   - データ操作言語 (Data Manipulation Language, 以下 DML と略す) によるプログラミングが難しい
6. 保守
   - システムがダウンしたとき, 回復操作ができない
   - データのバックアップ作業ができない
   - 使用しているデータベースの不満点を特定できない. またその修正が出来ない

この他にも, 使用する計算機システムのオペレーティングシステムにまつわる種々の困難 (コマンドの使い方, エディタ, コンパイラなどのプログラミングに必要なツールの利用法) などがある.

1 はデータベース構築の一番最初の作業中の問題である. 基本的には自分がそのデータをどのように利用していたかを解析するのであるが, データベース化することによって新しく生じるそのデータの利用法がそれではわからない. データベース化によって生じる何らかの利益を構築者は期待しているのであるから, このことはユーザの不満につながる.

2 は, データ (現実世界) のモデル化という作業が, データベースの専門家でさえ困難な作業であるため, 一般のユーザにとっては一層難しい問題となっている. この問題の一つの解決法は, データベースの専門家に相談することである. しかし, 専門家の数は限られており一般にはこれは難しい.

3, 4, 5, 6 は, オペレーティングシステムに関連する問題とまったく同じ問題である. すなわちこれらの問題が生ずる段階では, 使用する DBMS に関する詳細な知識が必要とされるわけであるが, その習得が一般ユーザには非常に困難である. DBMS はオペレーティングシステムに匹敵するほどの複雑なソフトウェアである. したがって, 計算機システムそのものに関してあまり知識を持っていないと思われる一般構築者には当然のこととともいえる.

3.3.3 データベース構築・利用支援と動的解析

本節では, データベース関係の構築·利用支援と動的解析の関係について述べる.

2.2 節で述べたようにデータベースの構築作業の中心は論理設計, すなわち構築するデータベースの対象世界のモデル化である. モデル化に, 必要な知識を発注者から獲得しなければならない. もし, データベースのドメインが設計者にとって未知のものならば, この段階で前提とできる知識は非常に少ない. このような場合, 人間のデータベース専門家は発注者に対してインタビューを実施し, ドメインに関する知識を獲得する. そして, この知識をもとにデータベースを構築する.

データベースが出来上がると利用段階に入る. しかし多くの場合, 満足のいくデータベースが一度の作業で完成されることはない. 知識の獲得が不十分であったり, 利用を通して発注者が新しいデータベースへの要求を思いつくことにより, 利用段階での問題点が生じる. そして時間がたつにつれ問題点は蓄積されていく. この問題点をもとに, データベースの専門家はデータベースの再設計, 再構築を行ない, 発注者の満足するデータベースを完成させる必要がある. この作業は一度で終わるとは限らず, 何回かの改良が行なわれるのが普通である. すなわち, データベースの構築は「データベースの構築→問題点の発生→データベースの再構築」という過程をとる. (図 3.2)

すなわち, データベースの構築においては, 試作されたデータベース (Prototype Database) を試用し (Use the Database), 問題点 (Problem) を洗い出す. そして問題点を順に解析し (Analyse the Problem), ドメインモデルを修正する (Correct the Domain Model). 修正されたドメインモデルをもとにデータベースを作り直し (Reconstruct the Database), 再び評価するという過程を繰り返す.

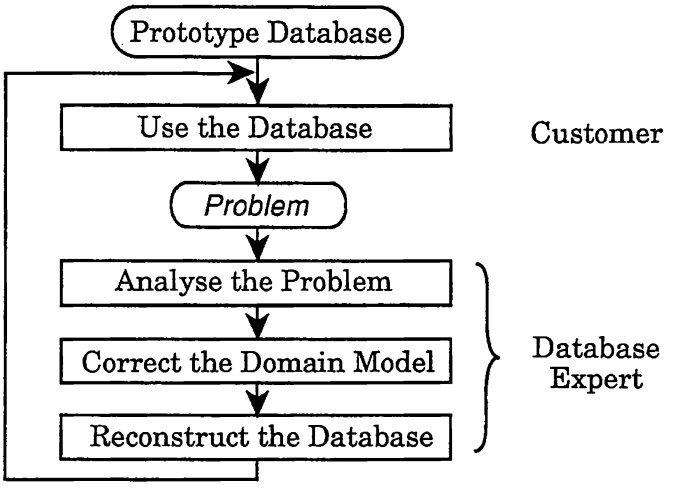


図 3.2 データベース改良の手順

この過程をインタビューの観点から見ると, 初期の知識不足, 誤りをその知識の産物を利用する段階で追加, 修正していることになる. すなわち, この過程は動的解析を行なっている過程である (図 3.3). まず, 初版データベース構築時にデータベース専門家は (不十分な) 知識を発注者から得る (図 3.3 (a)). これをもとにデータベース (先の知識の産物) を構築し, 発注者に提供する (図 3.3 (b)). 利用を始めると様々な問題点が生じる (図 3.3 (c)). これは, 先の知識の不備の発見に当たる. 次に, 専門家はこれをもとに発注者から得た知識を修正し (図 3.3 (d)), データベースを再構築するのである (図 3.3 (e)).

以上のことをデータベースの構築・利用支援に当てはめて考えてみると, 人間の専門家が行なうのと同様に動的解析を用いた支援システムという考え方が可能であろう.

動的解析では, 知識の適用およびその結果の検討が鍵を握っている. 例えば, 診断型エキスパートシステムの知識獲得における動的解析では, 知識の適用は診断の実行を意味する. また, 結果は専門家の出した結論と比較することによって検討される.

もちろん, データベースの構築支援システムとエキスパートシステムの知識獲得支援システムでは様子はかなり異なる. まず獲得される知識は, 構築するデータベースに対して発注者 (システムのユーザ) が潜在的に抱いている対象世界のモデルである. このようなモデルは客観性を伴う必要はなく, そのためこれを一般的に検証する術はない. したがって, 獲得した知識の適用, 結果の検討はシステムが獲得したモデルとユーザが抱いているモデルとの比較に帰結する.

さて, 構築したデータベースに対する検索要求文は, ユーザが抱いている最新のモデルを反映していると考えられる. そこで, データベースの利用時もシステムとの間に入り, 先に述べた比較を検索要求文を手掛りとして行なう. 構築時の知識の不備は検索要求文が前提とするモデルとの不一致という形で現れる. この不一致をもとにユーザに質問を行ない, このようなシステムは持つ知識を修正し, データベースの再構築を行なう. これは, 本質的にはデータベースの専門家のふるまいをシミュレートしているといえる. しかも, 自

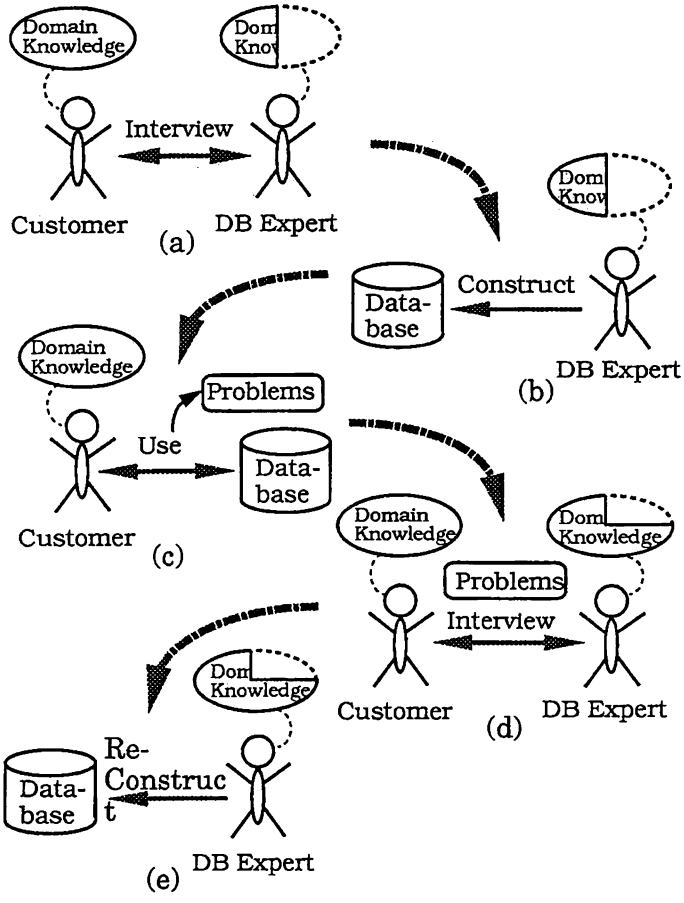


図 3.3 データベースの構築と動的解析

然言語による検索インタフェースをユーザに提供すれば常にユーザの検索要求文を (ユーザに意識させることなく) 検討し続けることができるため, ユーザの負担 (例えば, 苦情をいうなど) が非常に小さいという利点がある.

3.4 I^2S-DB

3.4.1 構成

図 3.4 に I^2S-DB の構成を示す. I^2S-DB は, I^2S-LD を中心として, DBMS とのインターフェース (DBMS Interface) やデータベース構築用知識ベース (KB), Supervisor といったモジュールよりなっている. 各モジュールの動作は Supervisor によって管理され, I^2S-LD と密接な関係を持つ. I^2S-DB で生成した初期データロード用プログラムおよびデータベース修正用プログラムも, 生成後は Supervisor によって管理される.

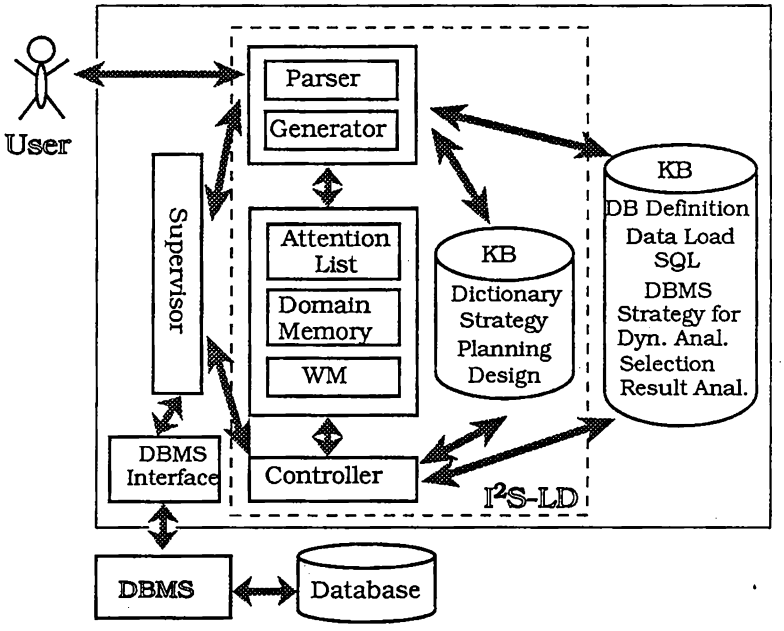


図 3.4 I^2S-DB の構成

DBMS インタフェースは, ユーザの入力した自然言語による検索要求文から I^2S-DB が作成した DML による検索文を DBMS に引き渡し, 検索結果を受け取るモジュールである. また, I^2S-DB が生成する初期データロードプログラム, データベース修正プログラムが使用するデータ操作用の述語パッケージも含む.

データベース構築用に I^2S-DB で追加された知識ベースには, 以下の知識が格納される.

- 論理設計結果から, DDL によるデータベース定義の生成を行なうための知識 (DB Definition).

- 論理設計結果から, 初期データフォーマット仕様書および初期データロード用プログラムの生成を行なうための知識 (Data Load).
- システムの持っているプラン構造と検索要求文からのプラン構造を比較し, attention の生成を行なう知識.
- 各 attention を解決するための知識 (Strategy for Dyn. Anal.).
- プラン構造から DML による検索文を生成するための知識 (SQL).
- 検索結果からユーザへの応答を生成するための知識 (Selection Result Anal).
- ドメインモデルの変更点から追加データフォーマットの仕様書およびデータベース修正用プログラムの生成を行なう知識.

なお, 実際に *I²S-DB* が動作するためには DBMS が必要である. 本システムは, 関係型の DBMS のみを対象としている. *I²S-DB* の試作に当たっては, Prolog で記述した関係型 DBMS シミュレータを使用した. このシミュレータは SQL 型の DML を処理できる.

3.4.2 動作

システムは, 図 3.5 に示すように動作する. まず *I²S-LD* を用いてユーザにインタビューを実施し, ドメインの知識を引き出す. この知識は先に述べたようにプラン構造の形でシステム内に蓄えられる. 次に, プラン構造からデータベースの論理構造を決定し, 使用する DBMS に合わせてデータベース定義ファイル (Create Database Definition), 初期データ格納用プログラム及び初期データファイル仕様書を作成する (Create Data Load Program & Data File Specification). データベース定義ファイルを DBMS に渡すことによってデータの入っていないデータベースの枠組みが計算機上に作成される. また仕様書をユーザに提示してデータファイルを作成してもらう. これを先のプログラムでデータベースに格納する (Data Input) と, 第 1 版の試作データベース (Prototype DB) が完成する. この部分までをシステムのデータベース構築支援機能と呼ぶ.

試作データベースが完成すると, これをユーザの利用に供する. この段階では, システムはデータベースの自然言語による検索インターフェースとして機能する. 構築支援と同じようにユーザの入力した要求文 (Query Input) をプラン構造表現に変換する (Translate into Plan-Representation) (以後このプラン構造表現をプラン表現と呼び, 現在のデータベースの基になったプラン構造表現を単にプラン構造と呼ぶ). システムはプラン表現と試作したデータベースの意味表現であるプラン構造とを比較することによって, ユーザの要求文が現在のデータベースの構造に適合しているかを調べる (Query fits the DB ?).

比較の結果ユーザの要求文が現在のデータベースの構造に適合していないときは, その違いをデータベースに対するユーザの新しい (あるいは潜在していた) 要求とみて, 古いプラン構造に修正を加える (Identify Problems, Correct Logical Design). プラン構造の修正部分より, 修正されたデータベース定義ファイル, 修正用データロードプログラムとその仕様書を作成する. ユーザは, 構築支援時と同様にこの仕様書のフォーマットにしたがっ

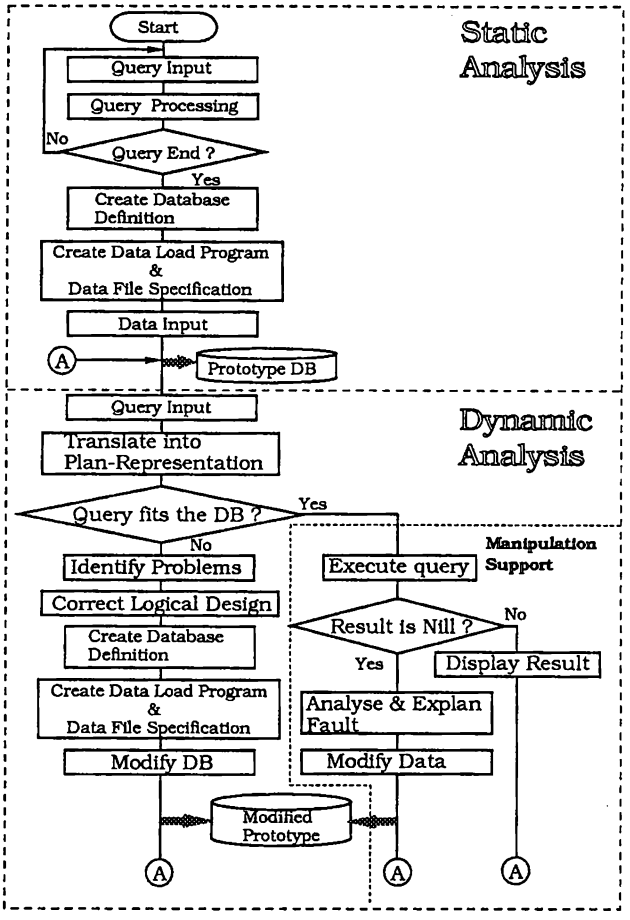


図 3.5 I^2S-DB の動作

てデータファイルを作成する. これを先のロード用プログラムで読み込むことによって修正版のデータベースが作成され (Modify DB), 次の検索からはこのデータベースが使用される. 以上をシステムのデータベース修正支援機能と呼ぶ.

一方, 比較が成功すると (要求文が現在のデータベースの構造に適合すると), プラン表現を DML 表現に変換してデータベースの検索を実行する (Execute query). そして その結果を出力して再びユーザに要求文の入力を促す. 検索結果が「空」の時, システムはその原因を指摘する応答 [Kaplan83] の生成を試みる (これをデータベース利用支援機能と呼ぶ).

構築支援時のインタビューが静的解析に, また修正・利用支援時のインタビューが動的解析に当たる. 静的解析時の質問の手掛りはプラン構造上で 4 種類の質問戦略にしたがって推論を行なうことによって得る. 一方動的解析時の手掛りは, ユーザが行おうとした検索のうち, なんらかの理由で失敗したときの検索要求文である.

図 3.6 にシステムとの対話例を示す. 図中, (1)(2) の部分で論理設計のインタビューが, (3) でデータベースの試作が行なわれている. (4) では試用時の検索が, また (5) ~ (7), (8) ~ (10) では試作データベースの構造に適合しない検索を行なった場合が示されている. なお, 以下の節で示される例はすべてこの対話例に対応するものであり, 文中の番号が対話例中の対応する場所を示している.

3.4.3 特徴

I^2S-DB はインタビューの考え方を中心としてデータベースの構築・利用を支援するものであり, 以下に示す特徴を持っている.

(1) 利用に当たってユーザは構築しようとするデータベースに対して実行したい検索要求文をいくつか用意するだけでよい.

(2) 静的解析, 動的解析を通じてシステムがユーザに刺激を与えるため, ドメインについての知識をユーザがあらかじめ整理する必要がない.

(3) 第 1 版が構築された後では, ドメインの知識がシステムに蓄積されているため, 従来の外付け型データベースインタフェースに比して,

- 自然言語による検索要求文を的確にＤＭＬに変換できる.
- 高次正規形に細かく分解された多数の関係を, ユーザに意識させることなく管理することができる. (ただし現在の試作システムでは高次正規形への分解は実装されていない)
- 利用時の検索失敗に対してドメインに即した原因の解析, 説明が可能である.

```
|?- interview.                                                    ┐
Please enter the name of your database.                          │
> makerDB                                                        │
Let's start designing conceptual schema of maker_db.             │
Please enter an information requirement.                         │
> Find the name and quantity of parts which are stocked. ··· ①   │
Is 'part' an instance of an entity ?                             │
> No.                                                            │
Please enter an information requirement.                         │
> Find the name of parts which are supplied. ····· ②             │
Please enter an information requirement.                         │
> None.                                                          │
Let's discuss the following noun:                                │ (1)
N: part.                                                         │
I think it's one of the physicalObject.                          │
Is it right ?                                                    │
> Yes.                                                           │
Please choose the attributes of the part from:                   │
1. size  2. color  3. price  4. (others)  (.<nl> for none)       │
> 2, 3.                                                          │
Ok, I start designing conceptual schema.                         │
Wait a moment, please.                                           │
Please choose the key attributes of the part from:               │
1. name  2. color  3. price                                      │
Which ?                                                          │
> 1.                                                             ┘
Designed conceptual schema is the following: ····· ③             ┐
  type(part,price,int)                                           │
  type(part,color,char)                                          │
  type(part,name,char)                                           │
  rel(stock,[part],[quantity])                                   │
  key(stock,[part])                                              │ (2)
  hie(stock,part,[],1:1)                                         │
  rel(supply,[part],[])                                          │
  key(supply,[part])                                             │
  hie(supply,part,[],1:1)                                        │
  selEnt(part,[],[name,color,price])                             │
  key(part,[name])                                               ┘
yes                                                              ┐
|?- do_format.                                                  │
Ok !! I start to make the load program. ····· ④                  │
                                                                 │
yes                                                              │
|?- do_load.                                                    │ (3)
Ok ! Load start !!                                               │
Completed to make the first data base ! ····· ⑤                  │
                                                                 │
yes                                                              ┘
```

図 3.6 I^2S-DB との対話例 (その 1)

```
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name and color of parts which are stocked. ····· ⑥
select name, color from  part, stock where
  stock.part=part.name

----------------
name     color
----------------
nail     black
wood     blown
body     white
funiture yellow
```

(4)

```
Please enter a query sentence.<none. for end.>
> Find the name and quantity of parts which are
    supplied from marui. ····· ⑦
Is 'marui' an instance of an entity ?
> Yes.
Please enter the entity name of 'marui'.
> Company. ····· ⑧
Let's discuss the following noun:
N: company.
I think it's one of the organization.
Is it right ?
> Yes.
Please choose the attributes of the company from:
1. address  2. (others)  (.<nl> for none)
> .
I think you should reform the following table:
1: reform 'rel(supply,[part],[])' to
      'rel(supply,[company,part],[quantity])'
Is it right ?
> Yes.
Please choose the hierarchical relation of company and
part of 'supply' from:
1. 1:1  2. 1:n  3. n:1  4. m:n
Which ?
> 3.
Ok !! I start to make the load program. ····· ⑨

yes
|?- do_modify.
Ok !  Load start !!
Completed to modify the data base ! ····· ⑩

yes
```

(5)

(6)

図 3.6 I^2S-*DB* との対話例 (その 2)

```
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name and quantity of parts which are
  supplied from toyota.
Is 'toyota' an instance of a company ?
> Yes.
select quantity, part from supply where company=toyota

----------------
quantity part
----------------
500      body
300      tire

Please enter a query sentence.<none. for end.>
> Find the name of parts which are constructed
  from tire.  ..................................... ⑪
I think you should reform the following table:
1: add 'rel(construct,[part(0),part(1)],[])'
Is it right ?
> Yes.
Please choose the hierarchical relation of part(0) and
 part(1) of 'construct' from:
1. 1:1  2. 1:n  3. n:1  4. m:n
Which ?
> 4.
Ok !! I start to make the load program. ..... ⑫

yes
|?- do_modify
Ok ! Load start !!
Completed to make first data base ! ..... ⑬

yes
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name of parts which are constructed
  from tire.
select part_0 from construct where part_1=tire

--------
part_0
--------
car

Please enter a query sentence.<none. for end.>
> None.
```

(7) (8) (9) (10)

図 3.6 I^2S-DB との対話例 (その 3)

3.4.4 構築支援

(1) 構築手順

図 3.7 に第 1 版試作データベースの構築手順を示す. *I²S-LD* がユーザにインタビューを実施し, データベースの論理設計を行なうまでの動作は, 第 2 章 で詳しく述べたのでここでは省略する. 以下この節で示す例は, 図 3.6 に示した対話例に対応するものである.

静的解析のインタビューを通じて構築されたプラン構造から, *I²S-LD* は, データベースの論理設計結果 (Design Result) を出力する. *I²S-DB* はこれをもとに, データベースを定義する. これは論理設計結果を対象 DBMS の DDL で記述したデータベース定義 (Database Definition) に変換し, DBMS に引き渡すことによって行なう. また, 論理設計結果から, 初期データファイルフォーマット仕様書 (Data File Spec.) と初期データロード用プログラムの生成を行なう. 仕様書にしたがってユーザに初期データファイルを作成してもらい, 初期データロード用プログラムを用いてデータベースに格納する (Load Data). データの格納は初期データロード用プログラム中で対象 DBMS の DML を用いて行なう.

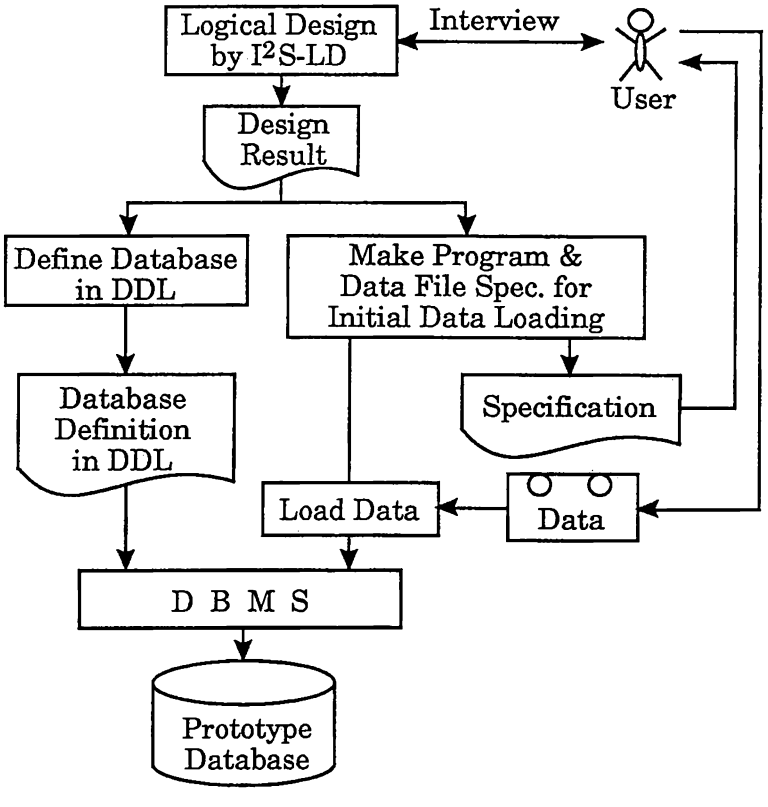


図 3.7 *I²S-DB* による第 1 版データベースの構築手順

このような手順で第 1 版試作データベースが完成する. なお, *I²S-DB* の構築支援機能は, ルール形式で表現された知識と前向き推論を行なうプロダクションシステムで実現さ

れている. 以下, 順に各段階の詳細を述べていく.

(2) データベース定義の作成

図 3.8 は, 図 3.6 の対話例 (1) で構築されたプラン構造である.

I^2S-LD は, このようなプラン構造から次のようにして試作するデータベースの論理構造を得る. ただしここで言う「関係」は, 関係型データベースと同じ意味での関係 (いわゆる「表」) である. 論理構造の決定にあたっては, 実体ー関連モデル [Chen76] の考え方に基づいている.

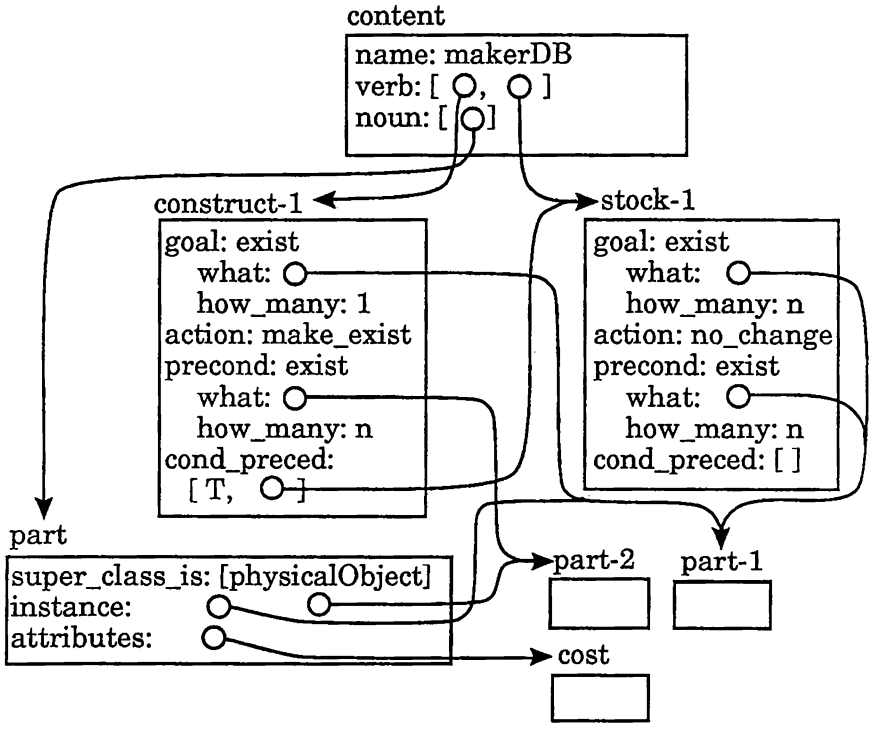


図 3.8 静的解析で構築されたプラン構造の例

(1) 動詞フレームのすべてについて

フレーム名→関係名
各スロットの値→その関係名の属性

として, その動詞に関する関係 (関連) を定義する.

(2) 全ての検索主体 (selectEntity) を表す名詞フレームについて

主体名→関係名
attributes スロットの値→その関係の名前

として, その主体に関する関係 (実態) を定義する.

(3) 各関連について, 関連する実体間の対応関係 (1:1, 1:m, m:n) をユーザに質問して決定する.

(4) 各関係のキー属性を決定する. 動詞に関する関係については change_how_many, change_where, change_state という状態の変化を示す意味素に関係する属性以外のすべての属性をキー属性とする. 主体に関する関係については, ユーザに質問して決定する.

(5) 各属性についてデータ型を決定する. type スロットが存在するときは, その内容をデータ型とする. 存在しないときは文字型とする.

以上のようにして図 3.9 に示す論理設計結果が得られる. 図中, ent および selEnt が実体 (名詞フレーム) に, また rel が関連 (動詞フレーム) に対応する関係を示している. 第 2, 3 引数は, それぞれ関連する実体 (ent, selEnt の場合は [], すなわち存在しない) および属性を示している. また, key が各関係のキー属性を, type が各属性のデータ型を示している. また hie は実体間の対応関係を示しており, 第 1 引数が関連名, 第 2, 3 引数が実体名である.

```
selEnt(part,[],[name,color,price]) ・・・・・・・・・・・・・・・・・・ ①
key(part,[name]) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ②
type(part,name,char) ・・・・・・・・・・・・・・・・・・・・・・・・・・ ③
type(part,color,char) ・・・・・・・・・・・・・・・・・・・・・・・・・・ ④
type(part,price,int) ・・・・・・・・・・・・・・・・・・・・・・・・・・・ ⑤
rel(stock,[part],[quantity])
key(stock,[part])
hie(stock,part,[],1:1)
type(stock,quantity,int)
rel(supply,[part],[])
key(supply,[part])
hie(supply,part,[],1:1)
```

図 3.9 論理設計結果の例

例えば, 部品 (part) については, ① で 3 つの属性, すなわち名前 (name), 色 (color), 価格 (price) を持つことが示されている. また, 部品のキー属性が ② によって名前 (name) であることがわかる. 各属性のデータ型は, ③④⑤によって, それぞれ名前 (name), 色 (color) は文字型 (char), 価格 (price) は整数型 (int) であることが示されている.

なお, 設計結果は関係データベースにおける第 1 正規型で, より高次の正規型へ分解できる余地がある. また対応関係, キー属性の決定などに関しては, システムからの質問が

非常に多い. これらはドメインに関する知識を表したプラン構造の表現力と I^2S-LD が持つ知識による問題でる.

この論理設計結果から各関係の定義を生成する. 実体 (ent, selEnt) および関連 (rel) について, それぞれ以下に示す順で行なう.

- 実体からの生成

  1. 設計結果から実体を 1 つ取り出し, 関係の名前とする.
  2. その実体の属性を設計結果から取り出し, 関係の属性とする.
  3. データ型を設計結果から取り出し, 関係の属性と対応付ける.
  4. その実体のキー属性を設計結果から取り出し, 関係の属性とする.
  5. 取り出した要素から, 関係の定義を書き出す.

- 関連からの生成

  1. 設計結果から関連を 1 つ取り出し, 関係の名前とする.
  2. その関連に関係している実体のキー属性を設計結果から取り出し, 関係の属性とする.
  3. その関連の属性を設計結果から取り出し, 関係の属性とする.
  4. データ型を設計結果から取り出し, 関係の属性と対応づける.
  5. 2 で取り出した関係の属性を, キー属性とする.
  6. 取り出した要素をフォーマットし, 関係の定義を書き出す.

本研究では, 対象 DBMS として Prolog でインプリメントした関係型 DBMS シミュレータを用いた. このシミュレータは, SQL [Date84] 型の DML を処理できる[2].

I^2S-DB が 図 3.9 の設計結果から作成したシミュレータの DDL で記述されたデータベース定義を図 3.10 に示す. このデータベース定義を DBMS に渡すと, データがまだ入っていないデータベース (の枠組み) が計算機上に作られる.

図 3.9 の (1) で関係「部品 (part)」が, また (2)(3) でそれぞれ関係「在庫する (stock)」, 「納入する (supply)」が定義されている. 例えば部品の場合, 三つの属性, すなわち名前 (name), 色 (color), 大きさ (size) が定義されている. それぞれのデータ型は, 空でない文字型 (char_not_null), 文字型 (char), 整数型 (int) である. またキー属性 (key) は名前 (name) であることが定義されている.

なお, このようなデータベース定義の作成は使用する DBMS に大きく依存する作業であり, ユーザが独力で試みる場合の最初の難関となるものである. I^2S-DB は使用する DBMS ごとに必要な知識を用意しておくことにより, この作業を自動化している.

[2]付録 B にシミュレータの仕様を示す.

```
database_definition
    table part
       attributes
         name:char_not_null
         color:char
         size:int
       key name
    table stock
       attributes
         part:char_not_null
         quantity:int
       key part
    table supply
       attributes
         part:char_not_null
       key part
end_database_definition
```

(1)

(2)

(3)

図 3.10 データベース定義の例

(3) 初期データファイル仕様書とデータロードプログラムの作成

データの入っていないデータベースの枠組みができると, 次に初期データを読み込む. まず, 論理設計結果から初期データ格納用プログラムおよびデータファイル仕様書を作成し (図 3.6 ④), ユーザに初期データを入力してもらう (図 3.6 ⑤). なお, プログラムは SQL 文を埋め込んだ Prolog プログラムとして作成される (DBMS にはデータ格納用のユーティリティを持つものも存在する. その場合は, この能力は特に必要ない). また仕様書は簡単な英語で記述される.

この段階では, 格納用プログラムとファイルの仕様書との間の一貫性の維持が課題となる. I^2S-*DB* では, 一貫性を維持するために両者を作成するためのルールは条件部を共有しており, 並行して作成を行なう.

データファイルを作成する際のユーザの負担をできるだけ小さくするため, また, データファイルを効率のよいものにするために, 次に述べる原則 (番号の若いものを優先する) に従ってデータファイルのフォーマットは決定する.

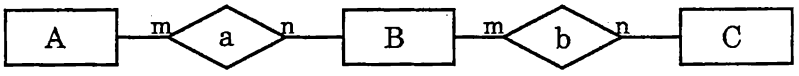
(1) 関係する関連の数が最も多い実体から順にデータを格納する.

(2) ある実体について一組のデータを入力するたびに, 関係する関連のうちで入力可能なデータをすべて格納する.

(3) ある関連に関して実体間に 1:n の対応関係があるときは, n 側の実体から先にデータを格納する.

ここで (1) は, データファイル作成時のユーザの見通しをよくするためである. 図 3.11 に実体と関連が関係している場合のデータ格納の順番に関する簡単な例を示す. ここで (a) は多くの関連が関係している実体から格納していく方法で, (1) に従う方法である. (b) は逆に関係している関連が少ないものから格納していく方法である. (b) のような方法よ

りも (a) の方が実体と関連の関係が明確であり, ユーザのデータファイル作成の作業効率も上がる. また (2) に従うことによって, 関係するデータをデータファイルの中で一まとまりにすることができる.



(a) 実体のデータを B→A→C の順 (関係している関連の数が多い順) で入力する. データファイル内の順は, B→Aa→Cb になる

(b) 実体のデータを A→C→B の順 (関係している関連の数が少ない順) で入力する. データファイル内の順は, A→C→Bab になる

図 3.11 実体の入力順によるデータファイルの違い

(3) は, データファイルの量的効率を高めるためである. 図 3.12 に, 1:n の階層関係があるデータベースでのデータの格納順によるデータファイルの違いの例を示す. データ数が大きくなればなるほど, (3) に従ったほうが, 量的効率が上がる.

以上のような原則にしたがって, 実際には以下に示す格納順になるように初期データロード用プログラムおよびデータファイル仕様書を作成する.

1. 検索主体のうち関係している関連の数の一番多いものを選択する.

2. 選択した検索主体とそれに関係する関連の全ての属性を格納する.

3. 検索主体がすべて選択されるまで 1 から繰り返す.

4. 関係している実体で, その実体以外はすでに選択されている関連を選択する.

5. 実体がすべて選択されるまで 4 を繰り返す.

このようにして, 初期データロード用プログラム, 初期データフォーマット仕様書ができる.

図 3.13 にプログラムの例を示す. プログラムは load 述語 (1) を呼び出すことにより起動される. プログラム中, abolish (①) は引き数の数が第 2 引数の値と等しく, 名前が第 1 引数と等しい述語を消去する述語である. また readIn (②) はデータファイルから 1 行読み込み, 第 1 引数に単一化する述語である. ただし, 改行コードのみからなる行については'[]'(空リスト) を単一化する.

α

A	X
a_1	x_1
a_2	x_2

β

B	Y
b_1	y_1
b_2	y_2
b_3	y_3

γ

A	B	Z
a_1	b_1	z_1
a_1	b_2	z_2
a_2	b_3	z_3

(a) α と β に 1:n の階層関係があるデータベースの例

a_1
x_1
a_2
x_2
[NL]
b_1
y_1
a_1
z_1
[NL]
b_2
y_2
a_1
z_2
[NL]
b_3
y_3
a_2
z_3
[NL]
[NL]

(b) α を先に入力する
データファイル

b_1
y_1
b_2
y_2
b_3
y_3
[NL]
a_1
x_1
b_1
z_1
b_2
z_2
[NL]
a_2
x_2
b_3
z_3
[NL]
[NL]

(c) β を先に入力する
データファイル

図 3.12 データ格納順による量的効率の違い

```
load(DataFile):-                                                    ┐
   see(DataFile),                                                   │
   repeat,                                                          │
      readIn(Part),                                                 │
      partRead(Part),                                               │
   next,                                                            │
   abolish(next,0), ····· ①                                         │ (1)
   repeat,                                                          │
      readIn(Part), ····· ②                                         │
      supplyRead(Part),                                             │
   next,                                                            │
   abolish(next,0),                                                 │
   seen,                                                            │
   !.                                                               ┘
partRead([]):-                                                      ┐
   asserta(next),                                                   │
   !.                                                               │
partRead(Part):-                                                    │
   readIn(Color),                                                   │
   readIn(Price),                                                   │ (2)
   execSQL((insert_into part:(Part,Color,Price))), ····· ③          │
   readIn(Quantity),                                                │
   stockRead(Part,Quantity),                                        │
   !.                                                               ┘
stockRead(_,[]):-                                                   ┐
   !.                                                               │
stockRead(Part,Quantity):-                                          │ (3)
   execSQL((insert_into stock:(Part,Quantity))), ····· ④            │
   !.                                                               ┘
supplyRead([]):-                                                    ┐
   asserta(next),                                                   │
   !.                                                               │ (4)
supplyRead(Part):-                                                  │
   execSQL((insert_into supply:(Part))), ····· ⑤                    │
   !.                                                               ┘
```

図 3.13 初期データ格納用プログラムの例

partRead (2), stockRead (3), supplyRead (4) が, それぞれ三つの関係,「部品 (part)」,「在庫する (stock)」,「納入する (supply)」のためのデータの格納を担当している. execSQL (③④⑤) は SQL 文を呼び出す述語で, 引き数が DBMS に引き渡される.

試作した I^2S-*DB* では, ファイルフォーマットの誤りに関してその検出, 回復を行えるプログラムは, 生成しない. I^2S-*DB* は経験の少ないユーザを対象ため, そのような機能を持ったプログラムを生成するよう改良する必要性が残されている.

```
You have 3 relations, "part", "stock" and "supply".
Please make a datafile in the format as showm bellow:

    name(int)          ; name of a part
    color(char)        ; color of the part
    price(int)         ; price of the part
    quantity(char)     ; quantity of the part which is stocked

 /* Here, 1 block for "part" and "stock" ends.  Put <nl> to show
    the end of blocks, or repeat another block in the same format
    as shown above.
 */                                                              (1)

    part(char)         ; name of a part which is supplied

 /* Here, 1 block for "supply" ends.  Put <nl> to show the end of
    blocks, or repeat another block in the same format as shown
    above.
 */                                                              (2)
```

図 3.14 データファイル仕様書の例

図 3.14 は, データファイル仕様書の例で, 図 3.13 に対応するものである. 二つのブロック (1)(2) からなるデータファイルを作成するよう指示している. ブロック (1) で関係,「部品 (part)」,「在庫する (stock)」のデータを, またブロック (2) で関係,「納入する (supply)」のデータを要求している.

DBMS によってはデータベースの一貫性維持機構が強力なため, あまり経験のないユーザはデータの依存関係に注意しないとデータを格納できない場合がある. システムはデータの依存性に注意してプログラムを作成するため, そのような問題をあらかじめ回避していると言える.

このように, データベース定義ファイルを DBMS に引き渡し, 初期データフォーマット仕様書にしたがって作成したデータファイルを初期データロード用プログラムを使用して格納することにより, ユーザは第 1 版試作データベースを手にすることができる.

3.4.5 修正・利用支援

第 1 版 (あるいは第 2 版以降) の試作データベース (以下, 試作データベースと呼ぶ) が完成すると, 利用段階にはいる. この段階では, 自然言語による検索機能をユーザに提供して, データベースの構造に起因する問題発生の検出とその解消を試みる.

(1) 問題点の検出

試作データベースの問題点検出は, ユーザに特別な負担をかけることなく行えることが望ましい. そこで本システムでは, ユーザに自然言語による検索インターフェースを通して試作データベースを利用してもらい, 同時にその際入力される検索要求文を監視する. このような検索インタフェースは, DML を覚える必要がなくなるのでユーザにとっても有益である.

システムは検索に当たってまず検索要求文をプラン表現に変換する. そして, 検索を実行する前にプラン表現とプラン構造の比較を行う. プラン表現はユーザの所望のデータベースのモデル (の一部) を表しており, プラン構造は, 現在の試作データベースの構造をそのまま表している. 従って比較によってユーザが想定しているデータベースの構造と実際の試作データベースの構造の不適合を, 前もって検出することができる. すなわち比較によってシステムは, 試作データベースがユーザの要求に (少なくとも現時点では) 十分合っているかどうかを判断することができる. また, 同義語の処理もこのとき行なう.

比較の結果, プラン表現がプラン構造に適合しないことがわかると, そのプラン表現に対応する検索要求文をユーザのデータベースに対する新しい, またはユーザ自身が意識していなかった潜在していた要求と見なして, インタビューに入る. 不適合は attention として検出されるので, その attention とそのときのプラン表現を I^2S-*LD* に引き渡すことによって, 修正版のためのインタビューが開始される.

一方, プラン表現がプラン構造に適合する (含まれる), すなわち試作データベースの構造がユーザの要求に充分答えられると思われる時は, このプラン表現を使用している DBMS の DML に変換し, 検索を実行する. 検索結果が「空」でないときは, 検索が成功したと考えその結果をユーザに示す. 検索結果が「空」の時, I^2S-*DB* は検索要求文のどこがおかしいのかを指摘する協調的な応答の生成を試みる. この場合ユーザに二つの選択肢が与えられる. 一つは使用を一旦中断して, データの追加作業を行うというものである. もう一つは検索要求文の入力に戻ることである. これらを利用支援と呼ぶ.

(2) attention とその処理

プラン表現とプラン構造比較時に生成される 5 種類の attention とその処理法を以下に示す.

(1) frame_not_exist

検索要求文のプラン表現内のある動詞フレームがプラン構造内に存在しない場合に生成される. これは, 対象ドメインでいままで知らなかった活動について検索要求がなされたことに対応する (図 3.6 ⑪はそのような検索要求文の例である.).

この場合, 2 つの可能性が考えられる. 1 つは, 同じ意味の動詞フレームがプラン構造中に存在しない場合である. もう 1 つは, 同義語が入力された場合である. そのためシステムは, 存在しないとされた動詞の同義語をプラン構造より捜し, その結果に従って処理をする.

まず, 存在しないとされた動詞フレームの同義語をプラン構造中で探索する. プラン構造では動詞の意味する活動が action の意味素で分類されており, goal および precondition でさらに細分化されている. そこで, そのフレームに相等する可能性があるプラン構造中のフレームを以下の順で探索する.

1. action で比較する.
2. goal で比較する.
3. precondition で比較する.

探索を行なって得られた候補フレームのフレーム名について, 同義語であるかをユーザに質問する. 同義語であるならば, フレーム名を置き換える. そうでない場合や, 探索によって候補が残らなかった場合は, その動詞の同義語もシステムのプラン構造の中に存在しない場合である. このとき, システムはプラン構造にその動詞フレームを付け加える.

(2) slot_not_exist

プラン表現内のある動詞フレームのあるスロットが, プラン構造で対応すると考えられる動詞フレームに存在しない場合に生成される. また, プラン表現内のある動詞フレームと別の動詞フレームの間の接続 (link) がプラン構造内の対応する動詞フレーム間に存在しない場合にも生成される. これは, 対象ドメインで, 既知の関連の新しい属性の存在を示唆している.

この attention については, システムはユーザに動詞フレームに属性を追加するかどうかを尋ねる. ユーザの返答が「追加する」なら, 動詞フレームに新しい属性を追加する.

(3) slot_not_same

ある動詞フレームのあるスロットの値がプラン表現とプラン構造で異なるとき生成される attention である (図 3.6 ⑦はそのような検索要求文の例である.). これには, 二つの場合が存在する. 一つは新しい関連あるいは既存の関連の修正が示唆された場合である. もう一つは, 検索要求文で使われたその値に対応する実体名がプラン構造内のある実体の同義語である, あるいはその実体名がプラン構造内のある実体を包含する (あるいはされる) 場合である (例えば,「工場」は「企業」に含まれると考えることができる).

まず, 同義語あるいは包含の関係があるか検討し, 関係がある場合は検索可能なようにスロット値を置き換える. 関係がない場合はインタビューを実施し, 必要に応じて新しい動詞フレームを作成する.

(4) class_not_exist

検索要求文のプラン表現内のある名詞フレームがプラン構造内に存在しない場合生成される (図 3.6 の compnay ⑧ がそのような名詞 (フレーム) の例である.). これ

には, 二つの場合が存在する. 一つは対象ドメインで未知の実体について入力された場合で, もう一つは既知の名詞の同義語が入力された場合である.

ユーザにインタビューを実施し, 必要に応じて新しい名詞フレームを作成する.

(5) attr_not_same

検索要求文のプラン表現内のある名詞フレームの属性がプラン構造内の対応すると思われるある名詞フレームの属性となっていない場合生成される. これは, ある実体の未知の属性が検索対象として指定された場合である.

システムはその属性が, ある名詞の属性であるかをユーザに尋ね, そうであるならばその名詞フレームに属性を新たに追加する.

以上のようにこの段階では, 問題が生じた検索要求文を手掛りとしてインタビューを実施し, 論理設計結果に修正を加える. 手掛りが得られた後では静的解析と動的解析には本質的な違いはない. 本システムでもインタビューは, 第 1 版のデータベース試作時とまったく同様の戦略, 機構を用いて行なわれる. 結果として得られる修正版の論理設計結果は, 第 1 版試作時に (利用時に) 問題となった検索要求文も手掛りとして同時に入力した場合と同じになる.

図 3.15 にプラン構造が修正されていく様子を示す. 4 角の枠がフレームを, 矢印がポインタを表している. 実線で示された部分が対話例 (図 3.6) の (1) で作られるプラン構造を, 一点鎖線, 点線で示される部分がそれぞれ (5), (8) で加えられたものである.

以下の説明で, 番号は図 3.6 の中の番号を指す. (1) ~ (3) でまず関係 supply, stock, part からなるデータベースが試作される (①②③). (4) の検索要求文 (⑥) はこの時点での試作データベースに適合している. (5) でプラン構造に存在しない名詞 company が (⑦⑧), また (8) では同じく動詞 construct が検索要求文に現われたので, 比較時に検出され新しいフレームが作成される. company に関しては supply フレームにスロット p:exist:where: (precondition の exist:where:) も加えられる.

(3) 試作データベースの修正

図 3.16 にデータベース修正の手順を示す. 修正版に対応する新プラン構造が得られると, これより修正版の論理構造を決定 (Database Definition in DDL), 修正版データベースの枠組みを計算機上に作る. また構築支援時と同様に, ルール形式で記述された知識を用いて追加データファイル仕様書およびデータベース修正プログラム (Program & Data File Spec. for Database Modification) を作成する (図 3.6 の⑨⑫が対応する対話例である). この仕様書 (specification) にしたがって作成されたデータファイルを先のプログラムで格納しながら, 旧版のデータベース (Old Database) の修正を行わなかった部分を新版に移すことによって修正版データベース (New Database) が構築される (同, ⑩⑬).

このように, I^2S-DB は修正するべき関係 (表) をまったく新しく作り直してから, 古いものを消去するという方法をとっている. この方法はデータベースの規模が大きくなると, 大量の記憶領域を必要とするため処理が困難になる. しかし, 本システムの主たる用途は

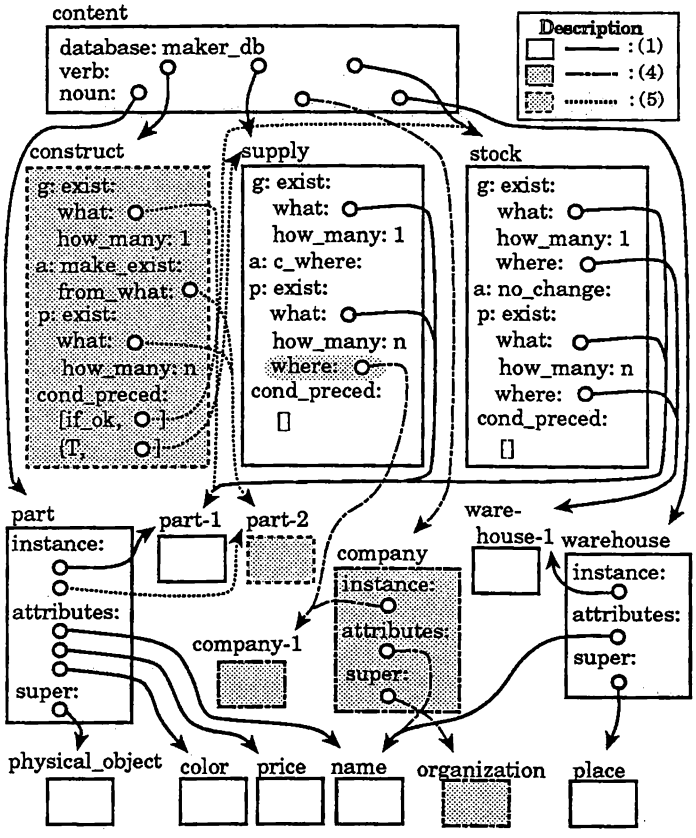


図 3.15 プラン構造の修正

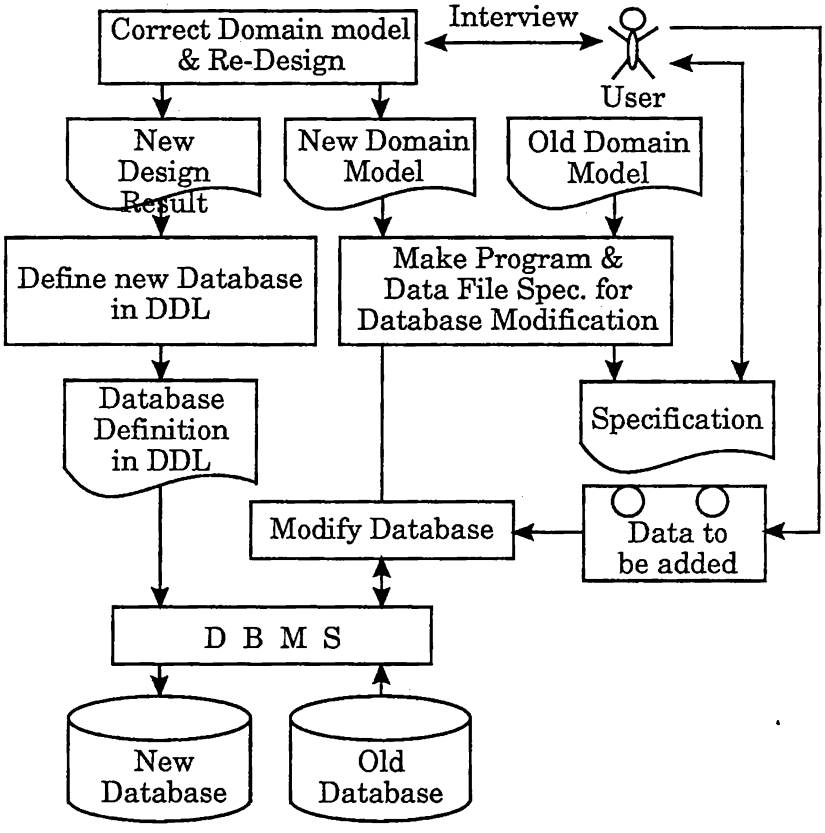


図 3.16 データベース修正の手順

個人レベルでの小規模なデータベースの構築・利用支援であり, 当面問題はないと考えられる. DBMS の中には論理構造を後から変更する機能を持ったものも存在する. しかし, 本システムではデータベース修正プログラムが複雑になるのを避けるため, このような機能は使用していない.

なお, 試作した I^2S-*DB* では修正として, ある関係への属性の追加と新しい関係の追加のみを考慮しており, 大幅な論理構造の修正については実装していない.

(4) 利用時の支援

図 3.17 に検索処理の流れを示す. ユーザの入力した検索要求文 (のプラン表現) が試作データベースの論理構造 (プラン構造) に適合していると, I^2S-*DB* はこのプラン表現 (Plan-Representation) をさらに DML 表現 (DML-Representation) に変換して実際に検索を行なう (Make Selection). I^2S-*DB* では, この変換もプロダクションを用いて行っている. (変換用のルールは, DML に依存するので使用する DML ごとに異なるルールセットが必要となる.)

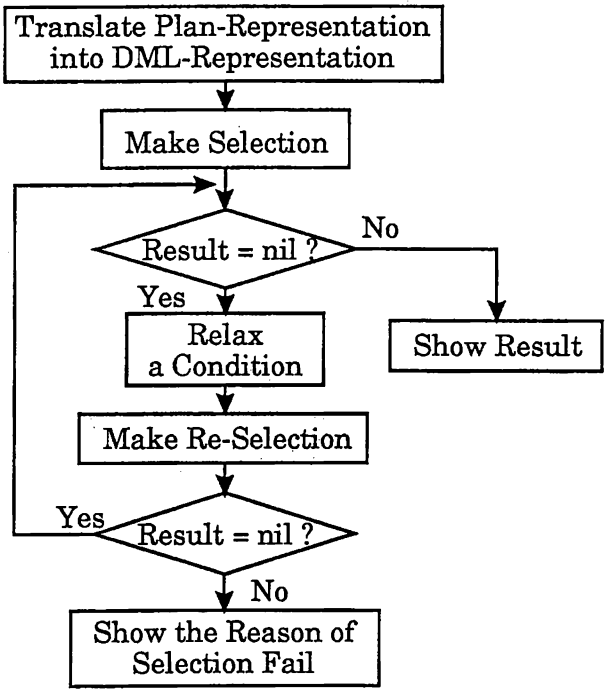


図 3.17 検索処理の手順

検索結果が空 (Result = nil ?) でなければ, I^2S-*DB* はこの検索が成功したものとしてその結果をユーザに提示し (Show Result), 再び検索要求文の入力に戻る. 結果が空になったときは, 検索失敗と考え, CO-OP の協調的応答 (Corrective Indirect Response) [Kaplan83] と同様の応答生成を試みる. この機能はユーザが入力したデータの不備の検出あるいはユーザのデータベースに対する誤解の訂正を目指したものである.

CO-OP は検索インタフェースの移植性を重視していたので, データベースの論理構造と使用する自然言語の表層知識 (構文知識) のみを用いて, MQL と呼ぶ中間表現上で協調的応答の生成を試みていた. I^2S-DB では, インタビューによってユーザから得たドメインに関する知識がすでにプラン構造という形でシステム内に存在しているので, これの活用を試みている.

検索結果が空となるのは, 検索要求文の制約条件を満たすデータが存在していないときである. CO-OP の MQL では, 検索の対象となる集合をノード, 制約をリンクとするグラフを用いて. 検索失敗の原因を, ノードごとに検索を行なうことによって同定する. 例えば結果が空となるノードが存在するときは, そのノードに対応する条件が原因である. またあるリンクに関して結合演算を行なったときに結果が空となるときは, そのリンクに関する条件 (そのリンクで結ばれたノードに対応する条件の論理積) が原因である. I^2S-DB の場合, 検索要求文の制約条件は, プラン表現内の (属性, 値, 属性と値の関係) という 3 つ組の集合で表している. そこでこの集合から 3 つ組を順に取り除くことによって CO-OP と同様の制約条件の緩和を行なう.

CO-OP ではドメインの知識に依存する意味処理を一切行わないため, MQL 上のすべてのノードについて調べ, 協調的応答を生成する. 一方 I^2S-DB の中間表現はプランの概念を含んでいるので, より大きな前提となっている制約条件から順に緩和していき, 原因を同定する. データを生じる活動の流れは常に「 precondition → goal 」の向きなので, 要求するデータに対して, その precondition が成立していなければデータを生じない (したがってデータベースにも存在しない). そこで検索結果が空になったときは, precondition が満たされていないことを疑う. 具体的には以下に示す順で処理を行なう.

(1) precondition が成立していないと仮定し, そこから生じている制約条件をはずして検索を行なう.

(2) この結果が空でなければ, この precondition が成立していないことがわかる. また, 空のままであれば, 原因は他にあることになる.

(3) 他に原因が考えられるときは, goal 側に接続をさかのぼって, 再度原因を探す.

プラン表現内では, 制約条件はあるスロットに与えられた値で表現されている. そこで以下の手順でプラン表現から動詞フレームのスロットを選択し, それを取り除くことによって制約条件をはずしていく.

1. フレーム内では precondition, goal の順で選ぶ.

2. precondition または goal 内では, 内容が制約条件であるスロット, 属性が関係になっているスロット, exist:what: スロットの順で選ぶ.

2 の処理は, スロットの性質の違いに起因している. 名詞には実体になれるもの (たとえば part 等) と実体になれないもの (たとえば tire 等) があるため, それらでは実体になれるもののほうが広い意味をもつ. また, exist:what: スロットは, そのフレームが表している活動の対象を意味する. そこで, この順でスロットを選んでいけば, 緩和する制約の

範囲が順に広くなっていくことになり, 制約条件をはずしていくという意味で妥当と考えられる.

検索失敗の原因が構文解析の段階でシステムが仮に値としたものであれば, 同義語, part_of の確認の質問をユーザに行い, その検索が失敗かどうかの確認を行なう. また, 失敗の原因が確定すれば, *I²S-DB* はその理由をユーザに明示する.

この様な処理の利点は, 制約条件の重要度についての順序付けをできるため, 原因の同定をある程度効率的にできる場合があることである. ただしこのような処理がいつでも意味を持つかについては若干の疑問があり, 議論の余地が残されている.

3.4.6 *I²S-DB* と知識獲得

2.2 節で述べたようにデータベースは, その対象である世界 (ドメイン) のモデルとなっている. そのため, データベースを構築するためにはそのドメインのモデルを行なう必要がある. *I²S-DB* は動的解析を用いたインタビューによってユーザからドメインに関する知識を引き出し, このモデル化を行なう. この意味で *I²S-DB* は, 一種の知識獲得支援システムであると言える. *I²S-DB* では, 獲得した知識をプラン表現で表している. そのため, 実際に獲得する知識は下に示すように, プラン表現に関するものである.

- 実体 (名詞) とその属性
- 関連 (動詞) とその属性
- 実体と関連の結び付き
- 実体間の同義語, part_of の関係
- 2 つの関連間の接続
- 関連の同義語

以下, システムを知識獲得の立場から整理し, 検討する.

第 1 版データベースの試作まで, すなわち構築支援の部分では, *I²S-DB* はまず, ユーザがあらかじめ用意していた構築したいデータベースに対する一連の検索要求文を解析してプラン構造に変換する. 次に, 得られたプラン構造を基にユーザにインタビューを行なう. ここではシステムは, あらかじめ *I²S-LD* が持っている知識 (質問戦略) を用いることによって知識を獲得する. この段階では実際のデータベースは存在せず, ユーザはデータベースを使用することなしに, インタビューに答えなければならない.

次にシステムは, 実際のデータベースを試作し, 修正・利用支援を行う. ユーザに実際に試作データベースを使ってもらうわけである. ユーザはその時点で抱いているデータベースに対するイメージを基に検索要求文を入力する. ユーザのイメージと実際のデータベースの構造が異なると, システムはその異なっている部分を検出し, ユーザにインタビューを行い必要ならデータベースの修正を行なう. この検出は, プラン表現とプラン構造を比較することで行なわれる. 実際にデータベースを使用することは, ユーザにとって良い刺激

となり使ってみて初めて判る要求や, 使っているうちに新しく思い付いた要求, またユーザも気付かなかったような潜在する要求などが生まれる. *I²S-DB* は, これらを検出しデータベースに修正を加えることによって徐々にユーザの要求を満たすようなものに近づけていく事ができる. この様に実際にユーザに目的とするシステムを使用してもらいながら, 新たにでてくる要求, 知識を獲得しているので *I²S-DB* は, 動的解析による知識獲得を行っているといえる.

動的解析を可能にすることによって生じる利点は以下のようにまとめることができる.

1. 静的解析の段階でドメインに関する知識の獲得が完全でなくてもよい.

   静的解析だけでユーザからドメインに関する知識をすべて引き出すことは事実上不可能である. しかし動的解析を導入することによって静的解析での知識の獲得が完全でなくてもよいため, その段階での完全なモデル化を目指した細かい質問は減らすことが可能である. また, ユーザ自身が適切に知識を整理できない場合には, 無理に質問に答えなくともよい. これらにより, ユーザの負担を減らすことができる.

2. データベースを利用することがユーザへの刺激となり, 新しい要求を生み出す. またその要求に対処できる.

   データベースを利用することは, 構築しようとしていたデータベースについてユーザに考えを述べさせる際の良い刺激となる. さらに, 静的解析時には思いもつかなかったような使い方をユーザが考えついた時でさえ, システムは対応が可能である.

このように, *I²S-DB* はデータベースの構築・利用支援システムではあるが, ユーザに試作システムを使ってもらい, その過程で知識を獲得していくという点で動的解析を用いた知識獲得支援システムであるということができる.

3.5 結言

本章では, 動的解析を用いてデータベースの構築を支援する知的インタビューシステム *I²S-DB* について述べた.

I²S-DB は, まず静的解析をもとにデータベースを試作する. そしてユーザに試用してもらい, その過程で生じた問題点を手掛りに, さらに良いドメインモデルを構築することを試みる. このような動的解析を導入したことにより, システムのユーザ (インタビューの受け手) の負担が減少される.

第 4 章

インタビューシステムと学習機能

4.1 緒言

本章ではインタビューシステムの学習機能について述べる.

第 3 章では, 静的解析の問題点を取り上げ, これを補完する動的解析を用いたインタビューシステムについて検討した. 二つの解析法は, 質問の生成に的を絞ったものであり, インタビューを通じて獲得した知識の再利用を考慮していない. そのため,

- 以前のインタビューでしたことがある質問とよく似た質問を繰り返す.
- 細かいことまで逐一, 質問する (質問が多い).

といった問題がある.

本章では, この問題に対処するために, I^2S-*LD* に付加した学習機能について述べる. この学習機能は, インタビューを通じて獲得した様々なデータベースのドメインモデルを蓄積・再利用するためのもので, 質問の省略, 簡略化などの効果をもたらす. また効率良く適切な示唆をユーザに与えることを可能にする.

次節でまず, I^2S-*LD* の知識と学習機能の関係について整理する. 4.3 節で, I^2S-*LD* に付加した学習機能について述べる.

4.2 学習機能

第 2 章で述べたように, I^2S-*LD* がインタビューを通じて獲得するドメインに関する知識は大きく以下の三つに分けられる.

1. ドメインの活動をプラン構造で表現したドメインモデル
2. 動詞フレームの各スロットに入る概念
3. 名詞の概念関係とその固有の属性

本学習機能では, インタビュー時に獲得したこれらの知識をドメインごとに一般化し, 関係のある知識をポインタで接続して記憶する. このような学習を行なうために, 上記の各知識に関して以下の知識表現を学習対象として導入する.

(1) ドメインフレーム

ドメイン内に存在する活動 (動詞) と実体 (名詞) を記述する.

(2) 動詞の辞書フレーム

動詞フレームのスロットに入る概念の候補を記述する.

(3) 名詞の概念フレーム

名詞と上位, 下位, 属性, 同義等の関係にある他の名詞を記述する.

これらの知識の詳細については, 4.3.1 で述べる.

そしてこれらの知識を学習するために, 従来の I^2S-*LD* に三つの学習機能を付加する. 以下に, 学習機能の概要を述べる.

(a) ドメインフレームの階層化

ドメインフレームを階層化して記憶する.

(b) ドメインに依存する動詞の辞書フレームの生成

ドメインごとに動詞の辞書フレームが存在するように拡張し, 各ドメインに適した候補概念を記述する.

(c) 名詞の概念階層の詳細化

名詞の概念フレームをドメインごとに階層化して記憶する. また, 名詞の属性や同義語, part_of などの関係にある名詞に対して, 名詞の概念フレーム間にポインタを張る.

これらの学習機能の詳細については, 4.3.2 で述べる.

学習後のインタビューでは, ドメインモデルに欠けている知識を得るために, まずそのドメインに含まれる動詞や名詞を手掛かりにして類似ドメインを類推する. そしてその類似ドメインの持つ知識を現在のドメインに適用することによって, ユーザの労力を軽減したり, ユーザにより効果的な「良い刺激」を与えることを実現している. このような知識を用いたインタビューに関しては, 4.4 節で述べる.

4.3 I^2S-*LD* への学習機能の付加

本節では, I^2S-*LD* がインタビューを通じて「ドメインに関する知識」を学習するための方法について述べる. まず, 学習の対象となる知識について述べ, 次に, 個々の知識の学習方法について述べる. そして, 学習された知識間の関係について述べる.

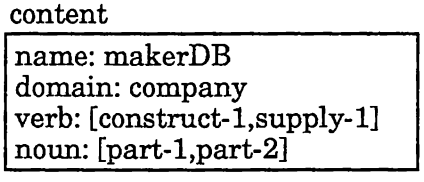
4.3.1 学習対象の知識

本項では, 本学習機能で学習対象とする三種類の知識について, 従来の I^2S-LD の知識と比較しながら詳述する.

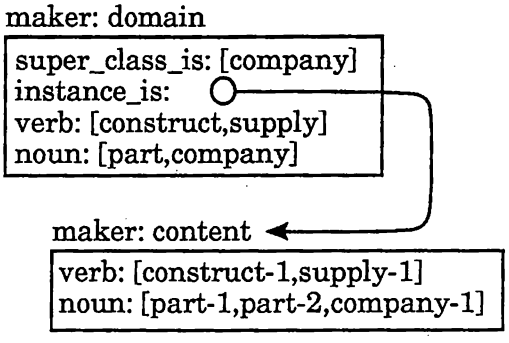
(1) ドメインフレーム

ドメインフレームは, ドメインにおける活動や実体に関する情報を記述する.

第 2 章で述べたように, インタビューの結果として図 4.1 (a) に示すようなコンテント (content) フレームが生成される. コンテントフレームには, ドメインに含まれる動詞と名詞のインスタンスをそれぞれ記述する verb スロットと noun スロットが存在している. ドメインフレームは, このコンテントフレームを一般化したものである. ドメインフレームには, 動詞スロット verb, 名詞スロット noun, そして インスタンススロット instance_is が存在している (図 4.1 (b)). verb スロットにはドメインに含まれる動詞のリストが記述され, noun スロットには名詞のリストが記述されている. すなわち, ドメインフレームから動詞や名詞の知識に対してポインタを張っている. instance_is スロットには, ドメインフレームのインスタンスに当たるコンテントフレームが記述されている. 次項で述べるように, ドメインフレームは階層という形で記憶されるので, 上位ドメインフレームを記述する super_class_is スロットが存在している. また, ドメインフレームの階層では, 上位のドメインフレームの verb スロットと noun スロットの内容が継承される.



(a) コンテントフレームの例



(b) ドメインフレームの例

図 4.1 コンテントフレームとドメインフレーム

以下, ドメインフレームとコンテントフレームをそれぞれ, 【ドメイン名:domain】【ドメイン名:content】と表記する.

(2) 動詞の辞書フレーム

動詞の辞書フレームには, ドメインにおける動詞に関連する情報を記述する. *I²S-LD* では, 図 4.2 (a) に示したドメイン独立の動詞の辞書フレームのみが存在していた. この動詞の辞書フレームには, 候補概念スロット value_class が存在しており, 動詞フレームの各スロットに入る概念の候補のリストが記述されている. 例えば, goal:exist:what: の概念候補は physicalObject, part, building である. このように従来の動詞の辞書フレームには全てのドメインに関する情報が混在していた. 本学習機能ではドメインごとに辞書フレームを持つように拡張する. そして, 全ドメイン共通の情報はドメイン独立の辞書フレームに, ドメインごとの情報はドメイン依存の辞書フレームに記述する (図 4.2 (b)). またドメイン依存の辞書フレームには, そのドメインにおいてその動詞に接続する他の動詞のリストを記述するための接続動詞スロット cond_preced を付け加える.

動詞の辞書フレームのスロットに名詞のリストを記述することは, 動詞の知識から名詞の知識に対してポインタを張ることに相当する. また, 動詞の辞書フレームをドメインごとに持つため, 動詞の名前からドメインフレームを参照することができる. すなわち, 動詞の知識からドメインの知識へのポインタも張られている.

以下, 動詞の辞書フレームに関しては, ドメイン独立の辞書フレームを〈動詞名:com〉, ドメイン依存の辞書フレームを〈動詞名:ドメイン名〉と表記する.

(3) 名詞の概念フレーム

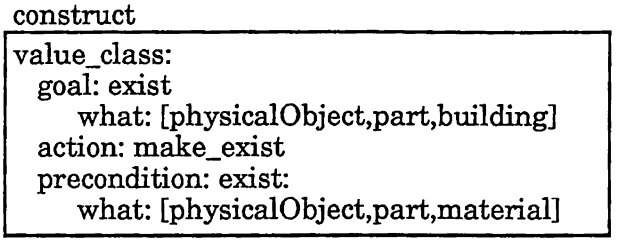
名詞の概念フレームには, ドメインにおける名詞に関連する情報を記述する.

従来の *I²S-LD* では, 図 4.3 (a) に示すような名詞フレームが生成されていた. この名詞フレームは, 上位概念スロット super_class_is と属性スロット attributes を持っている. これらのスロットにはそれぞれ, そのドメインにおいて名詞の上位概念に当たる名詞 (または抽象概念) とその名詞固有の属性が記述されている.

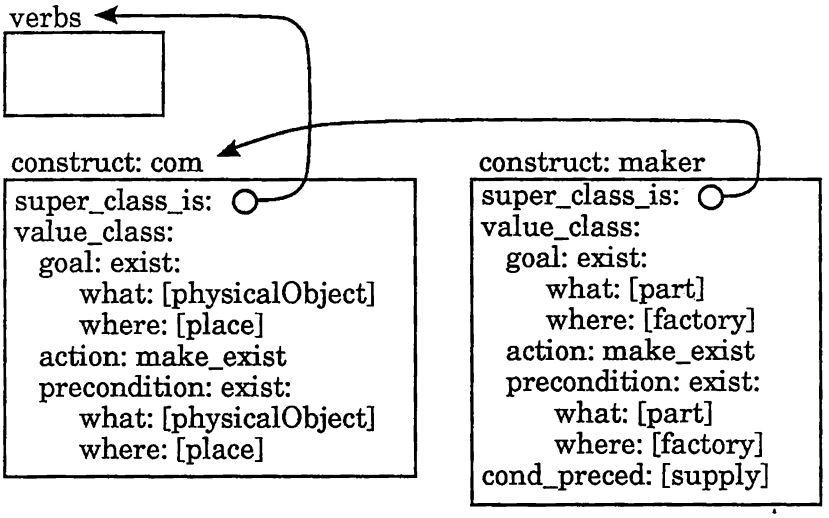
本学習機能では, 名詞に関して名詞概念フレームを導入し学習する. 名詞概念フレームは, 従来の名詞フレーム (図 4.3 (a)) が持っていたスロットの他に, 同義語 (synonym) スロット, 部分構成 (part_of) スロット, 属性時 (when_attr) スロット[1] を持つ (図 4.3 (b)). 同義語スロットには, その名詞と同じ意味を持つ名詞を記述する. 部分構成スロットには, その名詞を「部分として所有する」名詞を記述する. 属性時スロットには, noun ファシットと verb ファシットがあり, その名詞を属性として持つ名詞, 動詞のリストがそれぞれ記述される. 生成された名詞の概念フレームは名詞の概念フレームの階層に統合される. 概念フレームの階層では, 上位概念フレームのもつ attributes スロットが継承される.

名詞の概念フレームの属性時スロットの verb ファシットにその名詞を属性として持つ動詞のリストを記述することは, 名詞の知識から動詞の知識に対するポインタを張ることに相当する. また, 名詞の概念フレームの階層をドメインごとに詳細化することは, その

[1]抽象概念フレームについては既に属性時スロットを使用している (第2章, 17ページ参照).



(a) ドメイン独立の動詞辞書フレームの例



(b) ドメイン依存の動詞辞書フレームの例

図 4.2 動詞の辞書と学習機能

warehouse

super_class_is: [physicalObject] attributes: [address,volume]

(a) 従来の名詞フレームの例

warehouse: maker

super_class_is: [physicalObject] attributes: [address,volume] synonym: [ ] part_of: [company] when_attr: noun: [ ] verb: [stock]

(b) 名詞概念フレームの例

図 4.3 名詞フレームと学習機能

名詞の概念フレームの名前自体が, 名詞の知識からドメインの知識へのポインタに相当することを意味している.

以下, 抽象概念フレームと名詞概念フレームの名前はそれぞれ, 《抽象概念名 : com 》, 《名詞名 : ドメイン名》 と表記する.

4.3.2 学習方法

前項で述べた三種類の知識を学習するために, 三つの学習機能を I^2S-LD に付加した. その学習の方法について以下に述べる.

(1) ドメインフレームの階層化

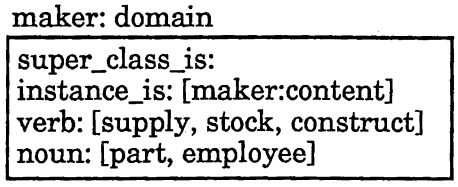
新しく獲得したドメインを以前に獲得したドメインと関係付けて記憶する.

a. 新しいドメインの統合 インタビューを実施したドメインに対応するドメインフレームを生成し, コンテントフレームの verb スロットと noun スロットに記述されている動詞と名詞をそれぞれ, ドメインフレームの verb スロットと noun スロットに記述する. そしてコンテントフレームをこのドメインフレームのインスタンスとする. 次に, ドメインフレームの階層の中から, その活動が新しいドメインフレームの活動に含まれるドメインフレームを探し出す.

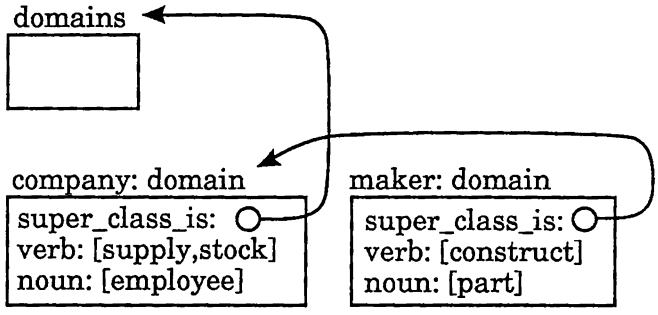
そして, その中から共通する動詞や名詞の数が最も多いドメインフレームを選び, そのドメインがインタビューを実施したドメインの上位クラスであるかユーザに尋ねる. ユーザが同意すれば, そのドメインフレームの下位クラスにコンテントフレームと同じ活動を

持つドメインフレームを作り, コンテントフレームをそのインスタンスとしてドメインフレームの階層に統合する.

例えば, maker ドメインに関してインタビューを行なうと, 【 maker : content 】 が生成される. インタビュー終了後, この 【 maker : content 】 の活動を一般化し, 新たにドメインフレーム 【 maker : domain 】 を生成する (図 4.4 (a)).



(a) 【 maker : domain 】 の生成



(b) 【 maker : domain 】 の統合

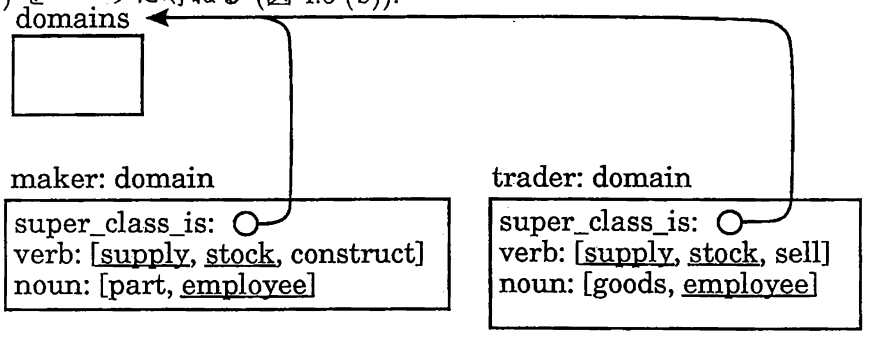
図 4.4 ドメインフレームの階層化

次に, 階層中のドメインフレームと 【 maker : domain 】 の動詞と名詞のリストを比較して, 活動が 【 maker : domain 】 の活動に含まれ, かつ最も似ているドメインフレームとして, 【 company : domain 】 が見つかったとすると, ユーザに company ドメインの内容 (動詞と名詞のリスト) を示し,「company ドメインは maker ドメインの上位クラスですか」と尋ねる. もしそうならば, 【 maker : domain 】 を 【 company : domain 】 の下位クラスとしてドメインフレームの階層に統合する (図 4.4 (b)).

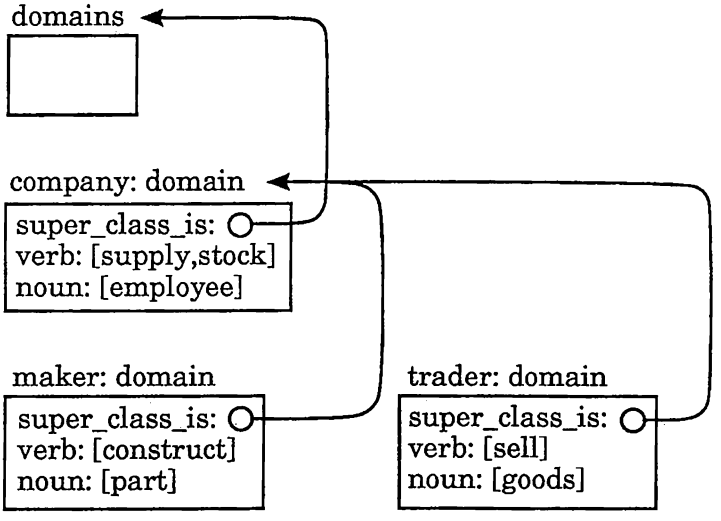
b. ドメインフレームの一般化 新しく階層に統合されたドメインフレームと共通の活動を持つドメインフレーム を探し出す. これは, 新しいドメインの統合と同様に, 新しいドメインフレームの verb スロットや noun スロットに記述されている動詞や名詞が過去に用いられたドメインを参照することで行なう. そし て, 両者がその共通の活動で一般化できるかユーザに尋ねる. ユーザが同意すれば, それらを一般化して, それらに共通の活動を持つドメインフレームを新たに作り, その名前をユーザに尋ねる.

例えば, 新たにドメインフレームの階層に統合された 【 maker : domain 】 と共通の活動

を持つドメインフレーム【 trader : domain 】が階層中に存在した場合 (図 4.5 (a)), ユーザに trader ドメインの内容を示して「 maker ドメインと trader ドメインは, その共通の活動 supply, company, および実体 employee で一般化できますか」とユーザに尋ねる. もしそうならば, それらを一般化し, 新しいドメインフレームの名前 (例えばこの場合は company) をユーザに尋ねる (図 4.5 (b)).



(a) 【 maker : domain 】と【 trader : domain 】の比較



(b) 【 company : domain 】への一般化

図 4.5 ドメインフレームの一般化

(2) ドメインに依存する動詞の辞書フレームの生成

動詞の辞書フレームをドメインごとに生成する.

インタビュー終了後, 構文解析などによって動詞フレームのスロットに入った名詞を, その動詞の辞書フレームの候補概念スロットのリストに加える. ただし, そのドメインに対する動詞辞書フレームが存在しない場合は, 新たに生成する. また, 名詞を候補に加える

際に, その名詞が他の候補 (抽象概念は除く) と概念階層において上位ー下位の関係にあれば, 上位の名詞のみを候補概念スロットのリストに記述する. これはこのリストに記述されている候補概念の全ての下位概念を候補概念としてシステムは取り扱っているからである.

例えば, 動詞 construct に関しては, あらかじめシステムにドメイン独立の辞書フレーム〈construct:com〉が用意されている. maker に関するインタビューで construct が初めて入力されると, インタビュー終了後, maker ドメインに依存する辞書〈construct:maker〉が生成される. そして候補概念スロットの goal:exist:what: と precondition:exist:what: には, 各スロットの候補概念として part が記述される. 接続動詞スロット cond_preced には, supply が記述される (図 4.2 (b)).

(3) 名詞の概念階層の詳細化

システムにあらかじめ与えられた抽象概念からなる名詞の概念階層をドメインごとに詳細化していく.

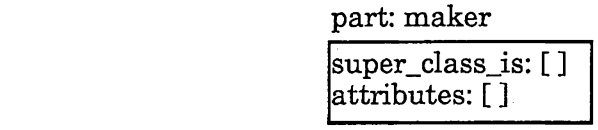
概念階層の詳細化の方法を以下に述べる.

a. 新しい名詞の概念フレームの統合 新しく入力された名詞は, その名詞が入った動詞のスロットに対する候補概念などを参照することによって, 候補を提示してその上位概念と固有の属性をユーザに尋ねる. そして, 新しい名詞の概念フレームをその上位概念の概念フレームの下位クラスとして名詞の概念階層に統合する. そして, 上位概念の属性 (抽象概念の場合は候補属性) を提示して, ユーザにその名詞の属性を尋ねる.

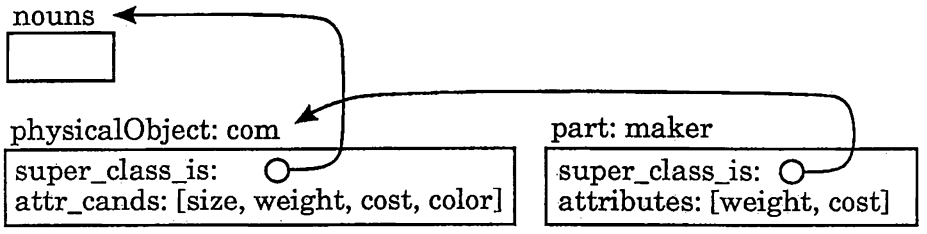
例えば, 名詞 part が maker ドメインに関するインタビューで初めて入力された場合, maker ドメインにおける part の意味を記述する名詞フレーム《part:maker》が生成される (図 4.6 (a)). part が入った動詞フレームのスロットに対する候補概念などを手掛かりにして,「part の上位概念は physicalObject ですか」などとユーザに尋ねる. もしそうならば,《physicalObject:con》の属性を提示しながらユーザに《part:maker》に固有の属性を尋ね,《part:maker》の attributes スロットに記述する. そして,《part:maker》を《physicalObject:con》の下位クラスとして名詞の概念階層に統合する. (図 4.6 (b)).

b. 名詞の概念フレームの一般化 名詞の概念フレームの階層への統合において, 新しく概念階層に統合された名詞と共通の属性を持つ名詞 (上位概念は除く) を探し, それらがその共通する属性で一般化できるかユーザに尋ねる. ユーザが同意すれば, それらの名詞を一般化して, その共通する属性を持つ新しい名詞の概念フレームを生成し, その名前をユーザに尋ねる.

例えば, 新たに階層に統合された名詞《part:maker》と共通する属性を持つ名詞《material:maker》が概念階層中に存在した場合 (図 4.7 (a)),「part と material は, 共通する属性 cost で一般化することができますか」とユーザに尋ねる. 可能ならば, それらを一般化し新しい名詞の概念フレームの名前 (例えば production_goods) をユーザに尋ねる (図 4.7 (b)).



(a) 《part : maker》の生成



(b) 《part : maker》の統合

図 4.6 名詞の概念フレームの階層への統合

c. **名詞間の特別な関係の獲得** 4.3.2 (2) において, 動詞の辞書フレームの候補概念のリストに新たな名詞が加わった場合, その名詞と以前からそのリストに記述されていた候補との間に, 特別な関係 (同じ上位概念を持つ, part_of の関係にある, 同義語であるなど) がないかユーザに尋ね, その答えに従って名詞の概念階層を詳細化する.

例えば, stock の goal, exist, where スロット (在庫する場所) に, warehouse と company が入った場合 (図 4.8 (a)),「warehouse と company にはなにか特別な関係がありますか」とユーザに尋ねる. 答えが「warehouse は company の一部分です」であれば,《warehouse : maker》の part_of スロットに《company : maker》を記述する (図 4.8 (b)).

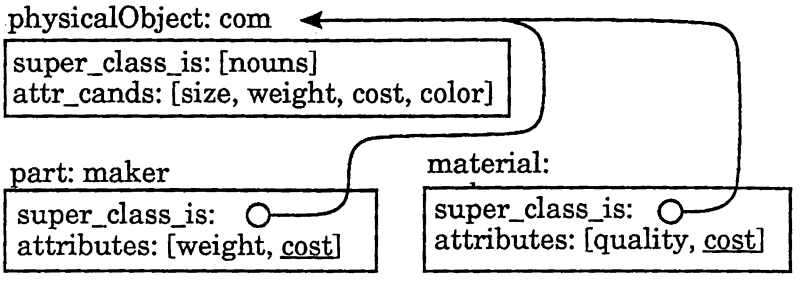
4.3.3 学習対象の知識間の関係

4.3.2 で述べたように, 学習対象の 3 種類の知識であるドメイン, 動詞, 名詞間には相互にポインタが張られている. これらのポインタの様子を図 4.9 に示し, 以下に, 各ポインタの参照の方法とその効果についてまとめる.

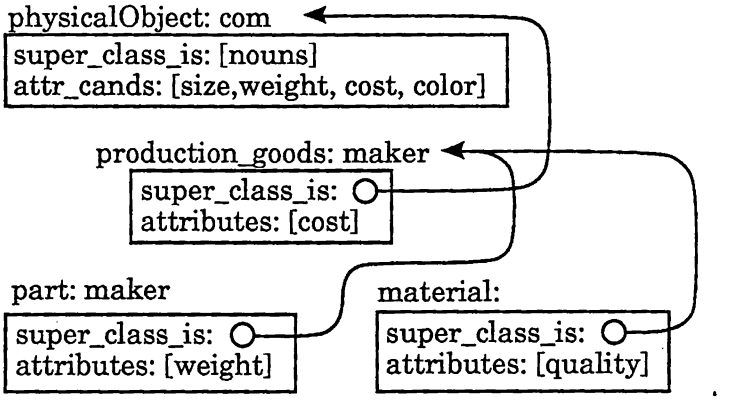
(1) ドメイン → 動詞 ドメインフレームの verb スロットに記述されている動詞のリストを参照することにより, そのドメインの中に存在する動詞を得ることが出来る.

(2) ドメイン → 名詞 ドメインフレームの noun スロットに記述されている名詞のリストを参照することにより, そのドメインの中に存在する名詞を得ることが出来る.

(3) 動詞 → ドメイン 動詞の辞書フレームを参照することにより, その動詞が以前に用いられたドメイン, 即ちその動詞をドメインにおける活動として持っているドメインを得ることが出来る.

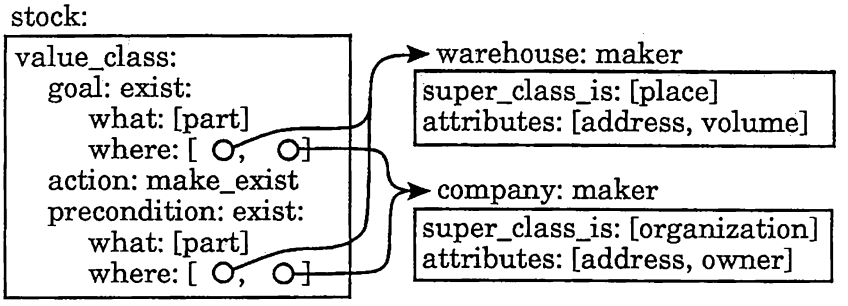


(a) 《part : maker》と《material : maker》の比較

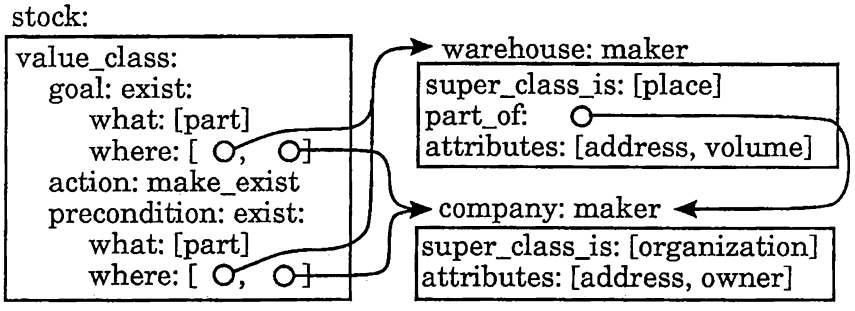


(b) 《production_goods : maker》への一般化

図 4.7 名詞の概念フレームの一般化



(a) warehouse と company



(b) part_of 関係の獲得

図 4.8 名詞間の特別な関係の獲得

(4) 動詞 → 名詞　ドメイン依存の動詞の辞書フレームの各スロットに記述されている候補概念を参照することにより, そのスロットに対してそのドメインにおいて入る概念の候補を得ることが出来る.

(5) 名詞 → ドメイン　名詞の概念フレームを参照することにより, その名詞が以前に用いられたドメイン, 即ちその名詞をドメインにおける実体として持っているドメインを得ることが出来る.

(6) 名詞 → 動詞　名詞の概念フレームの when_attr スロットの verb ファシットを参照することにより, そのドメインにおいてその名詞を属性としてとる動詞を得ることができる.

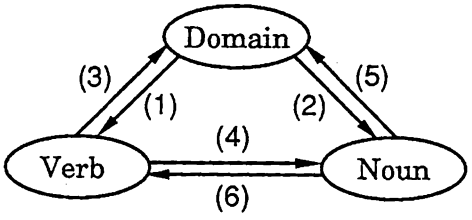


図 4.9 学習対象の知識間の関係

上記の学習の結果得られる知識の例を図 4.10 に示す. これは, maker ドメインにおいて「部品の組み立て」に関して入力された場合の例である.

図中の実線で描かれたフレームはシステムにあらかじめ用意されている知識で, 点線で描かれているフレームはインタビューを通じて獲得された知識である. 【 domains : domain 】, 〈 verbs : com 〉, 《 nouns : com 》は, それぞれドメインの階層, 動詞の辞書フレーム, 名詞の概念階層の最上位のフレームである. 〈 construct : com 〉は, ドメインに独立な辞書フレームであり,「 physicalObject から physicalObject が組み立てられる」という一般的事実を示している.

《 physicalObject : con 》は, 抽象概念の 1 つで, weight という属性を持っている. 【 maker : domain 】と【 maker : content 】は, それぞれ「部品を組み立てる (construct part) 」という活動をもつ maker というドメインのドメインフレームとコンテントフレームである. 【 company : domain 】は, 2 つのドメインフレーム 【 maker : domain 】, 【 trader : domain 】を共通する活動 (supply) と実体 (company, employee) で一般化したドメインフレームである.

〈 construct : maker 〉はドメイン依存の動詞の辞書フレームで,「 部品 (part) から 部品 (part) が組み立てられる (construct) 」いう maker ドメインにおける事実を示している. 《 part : maker 》は, part という名詞が maker ドメインで physicalObject に属し, cost という属性を持っていることを示している. また, 〈 construct-1 : maker 〉, 《 part-1 : maker 》, 《 part-2 : maker 》は, maker ドメインの活動を表現する動詞と名詞のインスタンスである.

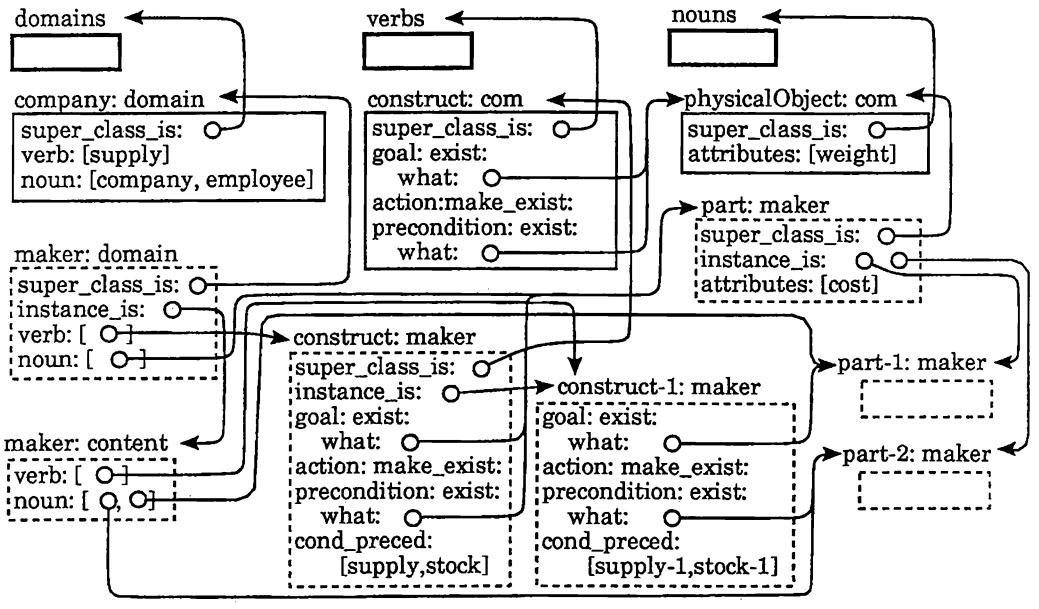


図 4.10 学習された知識の例

4.4 学習した知識の利用

インタビュー中のドメインにおいて獲得した知識や類似するドメインの知識を利用することによって, 次の効果が期待できる.

(1) ユーザの負担を軽減する.

- 冗長な質問をしない.
- 候補を提示して, ユーザに答えを選択してもらうようにする.

(2) ユーザに「良い刺激」を与える.

- ユーザにとって, 曖昧な事柄や気付いていない事柄について示唆する.

本節では, まず類似ドメインの類推方法について述べる. つぎに, 学習された知識が以降のインタビューに及ぼす効果について述べる. そして, 本学習機能の実現にあたり質問戦略を修正及び付加したので, 学習機能付き I^2S-*LD* がインタビュー時に用いる質問戦略について述べる. 最後に学習機能付き I^2S-*LD* による対話例を示し, 従来の I^2S-*LD* による対話と比較する.

4.4.1 類似ドメイン

まず, インタビューを行なっているドメインに最も類似しているドメイン (類似ドメイン) を類推する方法について述べる.

4.3.2 で述べたように, 動詞の辞書フレームや名詞の概念フレームは, それらが以前用いられたドメインへのポインタを持っている. そこで, インタビュー中のドメインに含まれる動詞の辞書フレームや名詞の概念フレームがもつドメインへのポインタをたどることによって, それぞれが以前に用いられたことのあるドメインのリストを得る. そして, ドメインごとにこれらのリストに記述されている回数を集計する. その回数が設定した閾値以上で, かつ最も大きいドメインを類似ドメインとする[2].

類似ドメインの類推は, 4.3.3 で述べた知識 (動詞, 名詞) をインデクスとして用いて, 関係のある他の知識 (ドメイン) を想起することに相当する.

次に, 類似ドメインの類推の具体例を示す (図 4.11). 現在, constructor (建設業) というドメインモデルを構築しているとする. まず, constructor のコンテントフレーム【 constructor : content 】 の verb と noun のリストに記述されている動詞 (supply-1, construct-1) の辞書フレームと名詞 (building-1, material-1) の概念フレームを参照して, それらが以前にどのようなドメインで用いられたか調べる (ドメインに対するポインタをたどる). この例では, 閾値 (2 個) 以上で, かつ最も多い (supply-1, construct-1, material-1 からの) 3 個のポインタ に指されている maker ドメインを類似ドメインとする.

[2]試作システムでは, 閾値を 2 個と設定した.

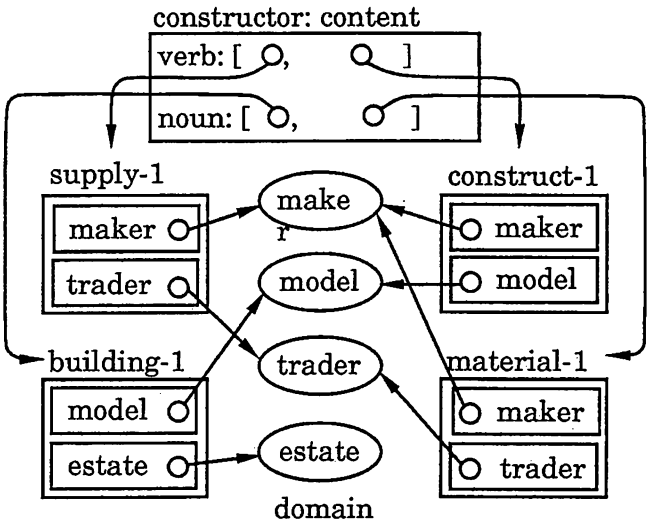


図 4.11 類似ドメインの類推

4.4.2 インタビューにおける効果

インタビューを通じて以前に獲得したドメインに関する知識を現在のインタビューに適用することによって, ユーザに対して以下に示す様々な示唆を与えることができる.

(1) ドメインフレームを用いたインタビュー

a. 未入力の活動と実体の示唆 類似ドメインフレームの動詞や名詞のリストにはあるが, 構築中のドメインには入力されていない動詞や名詞をユーザに提示することによって, 現在のドメインでユーザが入力し忘れているかもしれない動詞や名詞を示唆することが可能になった.

例えば, 現在インタビューを行なっている constructor ドメイン (図 4.12 (a)) の類似ドメインとして maker ドメイン (図 4.12 (b)) が類推されたとする. maker のドメインフレームのリストにはあって constructor にはない動詞や名詞 (図中の下線を付した動詞) に関して,「ship, stock などの動詞や part, warehouse などの名詞をデータベース化しなくてよいですか」とユーザに尋ねることができる. これは, インタビューの聞き手が「こんなことについて言い忘れてはいませんか」と相手に尋ねることに相当する.

(2) ドメインに依存する動詞の辞書フレームを用いたインタビュー

a. 接続する動詞の示唆 従来の I^2S-*LD* では, 動詞フレームに接続する動詞の候補を, プランニングの手法を用いてユーザに示唆している. しかしこの方法には無関係の候補 (動詞) が数多く提示されてしまうという欠点があった. 類似ドメインにおける動詞間の接続関係 (動詞の辞書フレームの cond_precedence スロット) を参照することによって, その動詞フレームに接続する動詞に関してより適切な候補の提示が可能になった.

constructor: content

verb: [supply-1, construct-1]
noun: [building-1, material-1]

(a) コンテントフレーム

maker: domain

verb: [supply, construct, ship, stock]
noun: [part, material, warehouse]

(b) 類似ドメインフレーム

図 4.12 類似ドメインを利用した未入力の活動や実体の示唆

例えば, constructor ドメインにおいて, stock-1 (図 4.13 (a)) に接続する動詞をユーザに尋ねる場合, 類似ドメイン maker における stock の動詞の辞書フレーム (図 4.13 (b)) の cond_preced スロット (supply) を参照することによって,「stock-1 に supply という活動は接続しますか」と尋ねることができる. これはインタビューの聞き手が,「この活動の前にはこんな活動が起こっている必要があるのではないですか」と受け手に尋ねることに相当する.

stock-1:

goal: exist:
    what: material
    where: [ ]
action: no_change
precondition: exist:
    what: material
    what: [ ]
cond_preced: [ ]

(a) constructor ドメインでの stock-1

stock: maker

goal: exist:
    what: part
    where: warehouse
action: no_change
precondition: exist:
    what: part
    where: warehouse
cond_preced: [supply]

(b) maker ドメインでの stock

図 4.13 ドメインに依存する動詞辞書フレームの利用

b. 動詞フレームにおける欠落情報の示唆 従来の *I²S-LD* では, ある質問戦略に基づいて動詞フレームの空スロットを埋めるための質問が為された場合, ユーザがその内容をなんの手掛かりもなしに入力しなければならなかった. 類似ドメインの動詞の辞書フレームの候補概念のリストを提示することによって, ユーザにその内容を示唆することが可能になった.

例えば, constructor ドメインの動詞フレーム stock-1 (図 4.13 (a)) の空スロット (goal: exist:where) の内容をユーザに尋ねる場合, 類似ドメイン maker の動詞の辞書フレーム 〈 stock : maker 〉 (図 4.13 (b)) の中の対応するスロットの候補概念 (warehouse) を参照することによって,「material (資材) は warehouse (倉庫) に在庫されるのですか」と尋ねることができる. これはインタビュアーが「この活動は, こんなものに対して, こんな所で, こんな風に起こっているのではないですか」と相手に尋ねることに相当する.

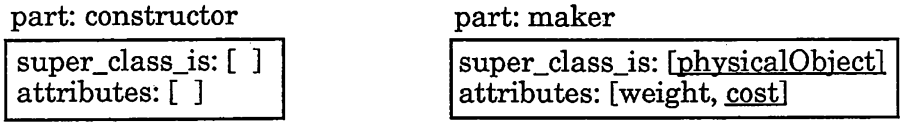
c. **未知名詞の上位概念の示唆** 構文解析時などで, 未知名詞が動詞フレームのスロットに入った場合, 動詞の辞書フレームの候補概念をユーザに提示して, その未知名詞の上位概念を尋ねる. しかし, 従来の I^2S-*LD* では動詞の辞書フレームが全ドメイン共通であったため, 構築中のドメインに全く関係のない候補も提示されていた. 学習機能付き I^2S-*LD* では, 動詞の辞書フレームがドメインごとに存在するように拡張されているので, インタビュー中のドメインに類似のドメインの動詞の辞書フレームの候補概念を参照することによって, 現在のドメインに関係の深い候補だけを提示することが可能になった.

例えば, constructor ドメインにおいて, stock-1 (図 4.13 (a)) の goal:exist:what スロットに未知名詞 steel frame (鉄骨) が入ったとすると, 類似ドメインである maker ドメインの動詞の辞書フレーム〈stock : maker〉の候補概念スロットを参照して,「steel frame の上位概念は material ですか」というように未知名詞の上位概念をユーザに示唆することが可能になった. これは, インタビュアーが「今の話題に類似の話題ではこの動詞は目的語としてこんな種類の名詞を持つので, 初めて現れたこの名詞 (動詞の目的語) も同じ種類の名詞ではないですか」と相手に尋ねることに相当する.

(3) 名詞の概念階層を用いたインタビュー

a. **未知名詞の上位概念の示唆** 現在インタビュー中のドメインにおける未知名詞が, 類似ドメインにおいても存在している場合, その名詞の上位概念を参照して, 現在のドメインにおいても同じ上位概念をとるのではないかとユーザに示唆することが可能になった.

例えば, constructor ドメインにおいて, 新たに part という名詞が入力された場合 (図 4.14 (a)), 類似ドメインの名詞の概念フレーム《part : maker》(図 4.14 (b)) を参照して,「part の上位概念は physicalObject ですか」とユーザに未知名詞の上位概念を示唆することが可能になった. これは, インタビューの聞き手が「この初めて入力された名詞は, 似ている話題の時にこのような種類の名詞であったから, 今の話題においても同じ種類の名詞ではないか」と考え受け手に尋ねることに相当する.



(a) constructor ドメインでの part　　(b) maker ドメインでの part

図 4.14 類似ドメインの名詞の概念フレームの利用

b. **構文解析の効率化** 名詞に関して, 属性, 同義語, part_of といった特別の関係にある名詞がわかっていれば, 構文解析時に冗長な質問を回避することが出来る. しかし, これら名詞の関係はドメインによって異なっているので, ドメインごとに名詞の概念階層を詳細化することによって, 類似ドメインの概念階層を参照することにより, インタビュー中のドメインに適した質問の実施が可能になった.

例えば, 類似ドメインである maker ドメインにおいて cost (原価) が part の属性であることがわかっていれば (図 4.14 (b)),「part の cost が知りたい」という文が入力された場合,「cost と part はどのような関係にありますか」というような余分な質問をせずに構文解析を行なうことが可能になる. これは, インタビュアーが「この名詞とこの名詞の関係はどうなっているのですか」と相手に尋ねなくてすむということに相当する.

4.4.3 学習機能と質問戦略

2.3.4 節で, 従来の I^2S-*LD* が用いていた質問戦略について述べ, 表 2.3 に個々の質問戦略を示した. 学習機能の実現においては, 従来の質問戦略を修正し, 新しい質問戦略を加えた. 学習機能付き I^2S-*LD* がインタビュー時に行なう質問は次の 2 種類に大別できる.

(1) ドメインモデルを洗練するための質問
データベース構築のため, ドメインモデルが持つ情報を豊かにするための質問である. これらは, 基本的には従来の I^2S-*LD* が行なっていた質問と同じ種類の質問である. 主な変更点は, ユーザに質問するときに, 前項で述べたような効果的な示唆をより多く与えることが可能になったことがあげられる.

(2) ドメインモデルをドメインの知識に統合するための質問
ドメインの知識を学習する際に生じる疑問点をユーザに尋ねる質問で, 従来の I^2S-*LD* には存在しなかった. なお, これらの質問は, 基本的にデータベース設計のためのインタビューが終了した後に実施される. また, 現在のデータベース設計には直接関係がないので, 省略 (学習機能の停止) することも可能である.

表 4.1 学習機能付き I^2S-LD の質問戦略

質問戦略	attention	種別
類似ドメインを用いて		
• 未入力の動詞や名詞を示唆する.	suggest_unknown	設計
動詞フレーム単体について		
• precondition と goal の詳細さを揃える.	check_detail	設計
• 時間に関する曖昧さを解消する.	ambiguity_period	設計
• 再帰性を調べる.	check_recursive	設計
• 接続可能な未入力動詞を探す.	search_unknown	設計
プラン構造内の動詞について		
• プラン構造内で接続可能な動詞の組を探す.	search_known	設計
• 接続している動詞間の詳細さを揃える.	connect_detail	設計
• 接続している動詞間の接続条件を調べる.	connect_cond	設計
名詞について		
• 未知名詞ならば, 名詞の概念フレームの階層に統合し, 属性を検討する.	map_unknown	設計, 学習
• 既知名詞ならば, 名詞の概念フレームの階層に instance として登録し, 属性を検討する.	map_known	設計
属性名詞について		
• when_attr スロットの verb ファシットを用いて未知動詞を示唆する.	suggest_attr_v	設計
ドメインフレームについて		
• コンテントフレームをドメインフレームの階層に統合する.	map_content	学習
• ドメインフレームを一般化する.	reorg_dframe	学習
動詞の辞書フレームについて		
• 候補概念スロットに新しい名詞を加える.	add_cands	学習
名詞の概念フレームについて		
• 階層内で一般化する.	reorg_nframe	学習
• 動詞の辞書フレームの候補概念スロットに加えられた名詞について, 他の概念との間に特別な関係がないか調べる.	check_sp_rel	学習

学習機能付き I^2S-LD が用いている質問戦略を表 4.1 に示す. 表の attention の欄には, その質問戦略に対応する attention が記述されている. また, 種別の欄に記述されている「設計」,「学習」という記述は, その質問戦略が「論理設計を行なうための質問戦略」と「知識を学習するため質問戦略」のどちらに属するかを示している. 前者に関しては, 学習機能のない I^2S-LD の質問戦略 (表 2.3 , 15ページ) に比べて, 類似ドメインの知識を利用することと全く新しい質問戦略 suggest_unknown が加えられたことが主な変更点である.

図 4.15 , 図 4.16 に, 建設業 (constructor ドメイン) に関する I^2S-LD との対話例を示す. 図 4.15 は建設業に関する一回目のインタビューの例で, 図 4.16 は二回目のインタビューの例である. 以下, この二つの対話例を比較しながら, 付加した学習機能の効果について述べる. なお, 図 4.15 と図 4.16 の対話例中の同じ数字の対話は, 基本的に同じ種類の質問戦略に基づいている.

二回目の対話例では, 一回目のインタビューによって「原価 (cost) が資材 (material) の属性である」こと判っているので, 図 4.15 (2) での質問①② は行なわれない. また, 対話例の (5) で接続する活動を示唆する際に, 多くの候補 (図 4.15 ③) を示さずに, 類似ドメインの類推によって適当な候補のみを提示している. さらに対話例 (6) では,「どこから供給されるか」に関する情報をユーザから入力 (図 4.15 ④) されなくても, 類似ドメインの類推によって,「会社 (company) から供給されるのではないか」と示唆を与えている.

また対話例の (8) に関して二回目の対話例では, 未知名詞の上位概念を図 4.15 ⑤ のような抽象概念ではなく, 建設業ドメイン固有の候補概念 (1 回目のインタビューで獲得) を提示している. 対話例 (10) では, 動詞の手掛りなしに入力された名詞, 従業員 (employee) の上位概念を, 図 4.15 ⑥ のように全ての候補概念を提示するのではなく, 類似ドメインでの従業員に関する知識を用いることによって適当な上位概念を示唆している. またこれは二回目の対話例に固有のものだが, 図 4.16 の (11) で, 類似ドメインの知識からユーザが入力し忘れているかもしれない活動や実体について示唆している.

```
Please enter the name of your database.
> constructorDB1.
Do you want to create a new database, constructorDB1 ?
> Yes.                                                          } (1)
Now, let's start designing conceptual schema of constructorDB1.
Please enter an information requirement.
> I want to know the costs of materials from which a warehouse
  is constructed.
Is 'material' an instance of an entity ? ..... ①
> No.
Is 'warehouse' an instance of an entity ?
> Yes.                                                          } (2)
Please enter the entity class name of 'warehouse'.
> Building.
Is 'cost' one of the attributes of the material ? ..... ②
> Yes.
Do you want to know the costs of material-1's from which
building-1 is constructed.
> Yes.
Please enter the next information requirement.
> Find the names of employees.
Is 'employee' an instance of an entity ?
> No.                                                           } (3)
Do you want to know the names of employee-1's ?
> Yes.
Please enter the next information requirement.
> No more.                                                      } (4)
Let's discuss the following fact:
F1:Building-1 is constructed from material-1's.
I think you should store the information about following
verbs which are possible to precede 'construct'.
  1.buy 2.sell 3.make 4.ship 5.lend 6.stock 7.supply ... ... ③ } (5)
Do you agree ?
> Yes.
Which do you want to store ?
> 7.
Please enter the information requirement about 'supply'.
> Materials are supplied from other companies. ..... ④
Is 'company' an instance of an entity ?
> No.                                                           } (6)
Do you want to store the information about
  "Material-1's are supplied from company-1" ?
> Yes.
```

図 4.15 建設業に関する一回目の対話例 (その 1)

```
Let's discuss the following facts:
F1:Building-1 is constructed from material-1's.
F2:material-1's are supplied from company-1.
Is the next fact true ?
  F:Building-1's in F1 are constructed from material-1's in F2.
> Yes.                                                          (7)
Let's discuss the following noun:
  N:Building.
I think it's one of the physicalObject. ..... ⑤
Is it right ?
> Yes.                                                          (8)
Please choose the attributes of the part from:
  1.size 2.weight 3.color 4.price 5.(other)
> 4.
Let's discuss the following noun:
  N:Company.
I think it's one of the individual or organization.
Is it right ?
> Yes.
Which does the company belong, individual or organization ?    (9)
> Organization.
Please choose other attributes of the company from:
  1.address 2.(other)
> 1.
Let's discuss the following noun:
  N:Employee.
Please choose the concept to which the employee belongs: ... ⑥
  1.abstractRelation 2.time 3.spaceAndLocation 4.volume 5.unit
  6.agentOfHumanActivity 7.individual 8.organization
  9.artificialObject 10.physicalObject 11.abstractObject
  12.naturalObjectAndPhenomenon 13.stimulus                    (10)
    14.natureAndStuff 15.livingThings 16.body
  17.other
> 7.
Please choose other attributes of the employee from:
  1.age 2.sex 3.address 4.occupation 5.(other)
> 1, 2, 3.
Ok, I start designing conceptual schema.  Wait for             (11)
a moment, please.
Designed conceptual schema is as follows:
  construct(building-1,material-1)
  supply(material,company)
  building(name,price)                                         (12)
  company(name,address)
  employee(name,age,sex,address)
  material(name,cost)
```

図 4.15 建設業に関する一回目の対話例 (その 2)

```
Please enter the name of your database.                                    } (1)
> constructorDB2.
Do you want to create a new database, constructorDB2 ?
> Yes.
Now, let's start designing conceptual schema of constructorDB2.            } (2)
Please enter an information requirement.
> I want to know the costs of materials from which a laboratory
  is constructed.
Is 'material' an instance of an entity ?
> No.
Is 'laboratory' an instance of an entity ?
> No.
Do you want to know the costs of material-1's from which
laboratory-1 is constructed ?
> Yes.
Please enter the next information requirement.                             } (3)
> Find the names of employees.
Do you want to know the names of employee-1's ?
> Yes.
Please enter the next information requirement.                             } (4)
> No more.
Ok.  I think you should store the information about 'supply'.              } (5)
Do you agree ?
> Yes.
Please enter the information requirement about 'supply'.
> Materials are supplied.
Do you want to store the information about
  "Material-1's are supplied" ?
> Yes.
Let's discuss the following fact:                                          } (6)
F2:Material-1's are supplied.
I think material-1's are supplied from companies.
Do you agree ?
> Yes.
Let's discuss the following facts:                                         } (7)
F1:Laboratory-1's are constructed from material-1's.
F2:material-1's are supplied.
Is the next fact true ?
  F:Laboratory-1's in F1 are constructed from material-1's in F2.
> Yes.
Let's discuss the following noun:                                          } (8)
  N:Laboratory.
I think it's one of the building.
Is it right ?
> Yes.
Please choose the attributes of the laboratory from:
  1.size 2.price 3.address 4.(other)
> 1, 2, 3.
```

図 4.16 建設業に関する二回目の対話例 (その 1)

```
Let's discuss the following noun:
  N:Company.
I think it's one of the organization.
Is it right ?
> Yes.
Please choose other attributes of the company from:
  1.address 2.(other)
> 1.
Let's discuss the following noun:
  N:Employee.
I think it's one of the individual.
Is it right ?
> Yes.
Please choose the attributes of the employee from:
  1.age 2.sex 3.address 4.occupation 5.(other)
> 1, 2, 3.
I think you should store the information about following
verbs and nouns:
  Verbs:  ship, stock
  Nouns:  part, factory, warehouse
Do you agree ?
> Yes.
Please enter an information requirements.
> Materials are stocked in some warehouses.
                •
                •
                •
```

(9)

(10)

(11)

図 4.16 建設業に関する二回目の対話例 (その 2)

4.5 学習機能の検討

I^2S-*LD* に付加した学習機能には, 次のような特徴がある.

(1) 獲得した知識をドメインごとに組織化し, 記憶しているため, 各ドメインに対応したインタビューが可能である.

(2)「ドメインに関する知識 (ドメインフレーム)」を用いるため, 全く初めてのドメインに対してもある程度の効率的な対応が可能である.

(3) インタビューを経験するにしたがって,「ドメインに関する知識」が蓄積されるため, インタビューの効率が向上する.

本学習機能の課題を以下に列挙する.

(a) 類似ドメインに関する課題

- 類似ドメインの類推方法の検討

現在は, 類似ドメインをドメインに含まれる活動と実体の一致する数が最も多いドメインとしているが, より有効な示唆や質問が可能になるような類似ドメインの類推方法を検討する必要がある.

一例として, ドメインフレームの階層における位置の近さの利用などが考えられる.

- 複数の類似ドメインによる示唆の検討

  現在のように類似ドメインの数を一つに限定せずに, 一つの示唆に関して, 複数の類似ドメインから示唆を生成し, その結果からユーザに行なう示唆を決定する.

(b) ドメインの階層化に関する検討

新しく獲得したドメインをドメインフレームの階層に統合するとき, またドメインフレームを一般化するときは, システムはポインタで接続されるドメインフレーム間の関係が妥当であるかを人間に確認する. 本来この I^2S-*LD* は複数のユーザに対して支援することを想定している. ひとつのドメイン内の知識の関係に関してはそのドメインのモデルを構築するユーザの判断に任せることができるが, 複数のドメインにわたる関係に関しては, 考え方の異なる複数のユーザが関与するとドメインフレームの階層の一貫性が保てなくなる. 従って, ドメインフレームの階層化に関しては, I^2S-*LD* のシステム管理者などを設定して管理させる方法が望ましいと考えられる.

(c)「ドメイン独立のインタビュー技術の学習」に関する検討

本学習機能では「ドメインに関する知識の学習」を実現した. より有効なインタビューシステムを開発するためには,「ドメイン独立のインタビュー技術の学習」の実現も必要になると考えられる. これには, 質問戦略の洗練などが考えられるが, 現時点では具体的な方法を論じるまでには至っていない.

4.6 結言

本章では, I^2S-*LD* に付加した学習機能について述べた. 本学習機能は, 個々のインタビューによって獲得した「ドメインに関する知識」を蓄積していく. そして以降のインタビューでそれらの知識を利用することによって, より効率的, 効果的なインタビューを可能にしている. これは, 獲得した知識を記憶する際に, 関係のある知識間にポインタを設けておいて, 以降のインタビューでそれらのポインタをたどることにより類似ドメインを類推し, その知識を利用するという方法で実現している.

第 5 章

インタビューシステムのアーキテクチャ

5.1 緒言

これまで, 第 2 章で静的解析を用いたインタビューシステムを, 第 3 章では動的解析について, また第 4 章ではインタビューシステムの学習機能について述べてきた. 本章ではこれらの検討結果を踏まえ, インタビューシステムに要求される知識およびアーキテクチャについて考察する.

まず, これまで開発されてきた代表的なインタビューシステムを取り挙げ, I^2S-*LD* (I^2S-*DB*, 学習機能) との比較, 検討を行なう. 次に, それらの検討をもとに, インタビューシステムが持つべき問題解決とその役割について検討し, さらに問題解決機能を補助するものとしての学習機能について考察する. 最後にインタビューシステムの一般的なアーキテクチャを設計, 検討する.

5.2 インタビューシステムの基本要素

5.2.1 従来のインタビューシステム

本節では, インタビューシステムの例として以下の 4 つを比較, 検討する

- TEIRESIAS [Davis82]
- MORE [Kahn85]
- MOLE [Eshelman86]
- I^2S/D[川口88]

TEIRESIAS は MYCIN のデバッガとして開発されたもっとも初期の知識獲得支援システムである. MYCIN が診断を誤った際に専門家によって起動され, ルールモデルおよびデータ構造スキーマと呼ばれる 2 種類のメタ知識を用いることにより, 新しいルールの入力および既存ルールの修正を支援する.

MORE は石油発掘時の掘穿泥水から地層を分析する MUD の開発から生まれた知識獲得支援システムである．獲得した知識を徴候，仮説，条件という 3 種類のノードからなるネットワークとして保持する．このドメインモデルに 8 種類の質問戦略を適用して，インタビューを進める．

MOLE は MORE の発展版である．MOLE ではドメインモデルの不確定さを許容できるようにしており，また動的解析を用いることによってインタビュー能力を強化している．まず，MORE と同様の質問戦略を用いて，ドメインモデルを構築する．つぎにこのドメインモデルから診断型エキスパートシステムを試作し，実際に診断タスクに適用する．試作システムの診断結果と専門家の結果を比較し，両者に相違がある際はインタビューを行なってドメインモデルの洗練を試みる．

I²S/D は油圧機器設計時の仕様獲得を支援するシステムである．発注者に適切な質問を行なうために，実際に設計を進めながら仕様獲得を行なう．システムは油圧機器設計に必要な基本知識を持っており，重要なパラメータの値の獲得，複数の設計案とそれらの長所，短所の提示，コストおよび安全面からの仕様変更の提案などを行なう．

I²S-LD は MORE 型，*I²S-DB* は MOLE 型のインタビューシステムと言える．*SIS* は MORE 型のインタビューシステムを一般化したものである．

4 つのシステムに共通するのは，何らかの形で問題解決機能をインタビューに利用していることである．この様子を図 5.1 に示す．TEIRESIAS は MYCIN を用いる．MORE は 8 つの質問戦略が相当する．MOLE は問題解決システムを試作して，インタビューに利用する．*I²S/D* は問題解決の過程でインタビューを行なう．

I²S-LD の場合は，4 種類の質問戦略が，また *I²S-DB* の場合は，試作したデータベースが問題解決機能にあたる．

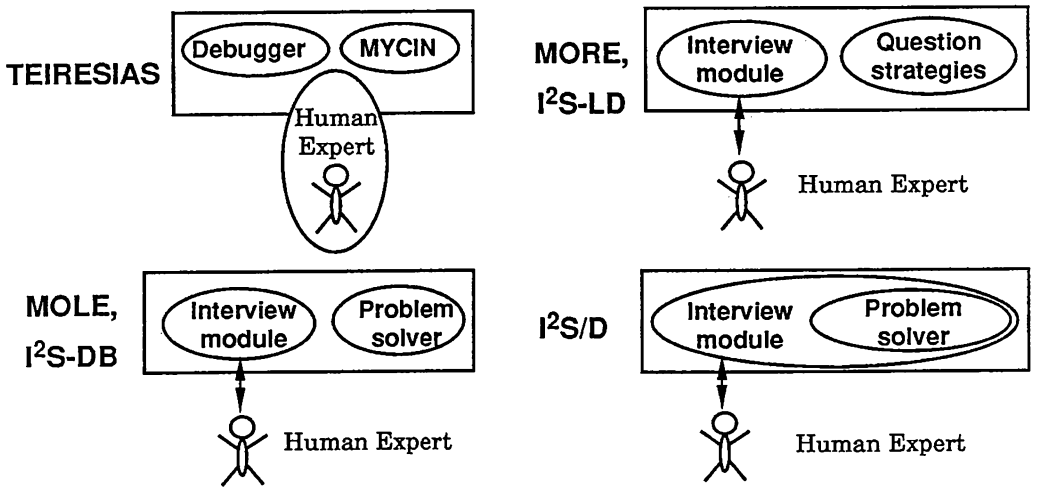


図 5.1 インタビューシステムと問題解決機能

5.2.2 問題解決とインタビュー

インタビューの目的は, 問題解決に役立つ知識を獲得することである. 人間がインタビューを行なうのは, その時点でその人間がある知識を必要としているからである.

インタビューシステムの基本課題は, どのような知識をどのように質問すべきかを決定することである. 特に質問の対象となっている知識がインタビューの受け手にとって複雑あるいは曖昧な場合, 質問の方法は受け手が答えられるかどうかということに対して大きく影響する.

どのような知識 (質問) が今必要かを決めるには, 実際に問題を解いて見るのがもっとも良いと思われる. 問題が解けなくなった時点で, どういった情報が不足しているかを検討し, それを聞き出すことができれば, 次回からは同種の問題を解けるようになる. すなわち, 知識を獲得してシステムの性能を向上できたことになる.

一方, 質問法としてはインタビューの受け手をも問題解決の過程に引き込む質問が良いと考えられる. なぜならば, 受け手に考えさせることによって, 受け手自身も用いている知識を意識し易くなるためである. したがって問題解決過程の提示あるいはそれに類似した提案, 示唆の提示などが有効となる.

先に挙げた 4 種類のシステムはすべてこのようにしてインタビューを進めているといえる. MORE の質問戦略は, ある時点までにシステムが獲得した知識が問題を解く (徴候から仮説を同定する) のに十分かどうか確かめていることに他ならない. MOLE も I^2S/D も実際に問題解決を試み, 困難が生じた際 (例えば結果が専門家と違うといったとき) に質問を行なう.

TEIRESIAS の場合は, MYCIN のデバッガとしての色合いが濃い. 問題解決 (MYCIN が担当している) の失敗の検出, 知識ベースの問題個所の同定, 知識ベースの修正などはすべて専門家に委ねられている. システムは単に先に触れたメタレベルの知識を用いて, ルール修正時の誤り防止や必要と思われる条件, 属性の示唆を行なう. この際に用いられる手法は統計的なもので, 正しい示唆を行なうとは限らない.

I^2S-LD の場合は, 4 種類の質問戦略によって現在のドメインモデルからデータベースを構築した時に生じるであろう問題を予測していると考えられる. *I^2S-DB* ではさらに実際にデータベースを試作して見て, ドメインモデルの不備の発見を試みる.

以上の考察から, インタビューシステムは次に示す構成を採っている必要があると思われる.

$$\text{インタビューシステム} = \text{問題解決モジュール} + \alpha_1$$

α_1 については以下, さらに検討を行なう.

5.2.3 学習とインタビューシステム

先に挙げた 4 つのシステムの内, I^2S/D は基本知識を用いて, 実際に問題を解きながらインタビューを進める. 基本知識を用いるのはシステムを (任意の油圧機器に関する仕様獲得ができるという意味で) 汎用にするためである. しかし, 油圧機器の種類によっては

自明な知識を利用していないため, インタビューも質問の多い, 非常に効率の悪いものになっている.

MORE の質問戦略は直接問題を解いてみることを不要にするためのもので, 人間があらかじめ行なった洞察に基づいている. 例えば, 仮説の区別 (Differentiation) という戦略は, まったく同じ徴候を持つ仮説の対に対して, それらを区別するための徴候を専門家に質問するというものである. このような仮説の対は実際に診断に用いられると, 一つの徴候に対して二つの仮説を導く. すなわちこの戦略は, 実際に診断を行なった際に予想される問題点にあらかじめ対処したものに他ならない.

このように MORE の 8 つの質問戦略は, 比較的簡単な処理で効果的な質問を可能としている. これは問題解決の過程で生じる困難をあらかじめ質問戦略という形式で整理してあるためである. 一方, I^2S/D のように問題を直接解きながらインタビューを進めるものは効率が悪い.

一般に問題を解決しながらインタビューを実施していくのは, 効率が悪い. 人間の場合でも, 聞き手が考え考えインタビューをするならば, 決して非常に時間がかかるインタビューとなるであろう.

効率は, 問題を解く過程をシステムに記憶していき, 後のインタビューで利用可能とすることによって改善できる. このような機能をインタビューシステムの学習機能と呼ぶことにする. 学習機能は, 以前のインタビューで得た知識を次のインタビューに用いることができるようにする機能である.

インタビューシステムに問題解決機能を持たせることによって, 良い質問の生成が可能となる. 効率は学習機能を持たせることによって, 徐々に良くしていくことが可能である. これをまとめると先のインタビューシステムの構成は次のように書くことができる.

$$\text{インタビューシステム} = \text{問題解決モジュール} + \text{学習モジュール} + \alpha_2$$

インタビューシステムの学習機能は, EBL [Mitchell86] と対比させると理解し易い. EBL のドメインに関する知識 (domain theory) に対応するのは, インタビューシステムにあらかじめ持たせる知識である. そしてこれを使って行なった問題解決およびインタビューの過程が EBL における説明であり, 獲得対象は不足するドメイン理論である. インタビューの過程を一般化して記憶することにより, 以後のインタビューでの効率が改善される.

第 4 章で述べた I^2S-*LD* の学習機能はこの考えに基づいたものとなっている.

先の式で α_2 に当たるものとしては,

1. 対話制御モジュール
2. 問題解決モジュールを監視するモジュール
3. 不足している知識を検討するモジュール

などが挙げられる. これらに関しては問題解決モジュールとの関係について, さらに検討を進める必要がある.

5.3 アーキテクチャとその動作

前節では, 従来の知識獲得支援システムや I^2S-LD, I^2S-DB を検討し, インタビューシステムには問題解決機能と学習機能が必要であることを述べた. 本節では, これをもとにインタビューシステムの一般的なアーキテクチャの設計を試みる. まず問題解決機能として, 簡単な汎用問題解決器のモデルを導入する. 次にこのモデルの上で問題解決過程のインタビューへの利用および学習機能の役割を議論する. 最後に, インタビューシステムの一般アーキテクチャを提案する.

5.3.1 問題解決器のモデル

さまざまな形式の問題解決器が考えられているが, ここではプロダクションシステムを用いる. 以下, 本節で定義する問題解決器のモデルを PS(Problem Solver) モデルと呼ぶ.

PS モデルの構成を図 5.2 に示す. 通常のプロダクションシステム [Winston77] と同様に, 問題解決の中間結果を格納しているワーキングメモリー (Working Memory, 以下 WM と略す), 問題解決用のルールを格納している知識ベース, ルールを解釈, 実行するインタープリタからなる.

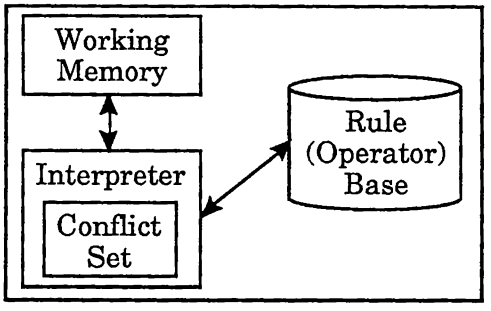


図 5.2 PS モデルの構成

WM には状態節,

$$s_1,\ s_2,\ \cdots\ s_i,\ \cdots$$

が格納される. 各状態節 s_i は, あるオブジェクトの属性と値を次のような形式で表現している.

$$(o\ (a_1\ v_1)(a_2\ v_2)\ \cdots\ (a_iv_i)\ \cdots)$$

ここで o はオブジェクト名, a_i, v_i はそれぞれ属性とその値を示す.

ある状態を別の状態に変化させるという意味で, PS モデルではルールのことをオペレータと呼ぶことにする. オペレータは次のように, 条件部と結論部からなる.

$$\begin{aligned} If\ \ & c_1\ \&\ c_2\ \&\ \cdots\ \&\ c_i\ \&\ \cdots\ \&\ c_m \\ Then\ \ & a_1\ \&\ a_2\ \&\ \cdots\ \&\ a_j\ \&\ \cdots\ \&\ a_n \end{aligned}$$

これは $c_1 \cdots c_m$ がすべて成立するとき, $a_1 \cdots a_n$ を順に実行するということを意味する. c_i のことを条件節と呼ぶ. 条件節は状態節と同様の形式で記述される. a_i を結論節と呼ぶ. 結論節は新しい状態節の WM への追加のみを表現できるものとする. すなわち WM 内の状態節の数は単調に増加する.

PS モデルは次の各ステップを 1 つのサイクルとして, それを繰り返すことによって問題解決を行なう.

Step 1 知識ベースからオペレータを 1 つ取り出す. 取り出したオペレータの条件部が現在の WM で満たされるときは, コンフリクトセットに登録する. これをすべてのオペレータについて行なう.

Step 2 コンフリクトセットの中から, オペレータを 1 つ選ぶ.

Step 3 選ばれたオペレータの結論部を実行する. この結果, WM に新しい状態節が追加される.

ただし Step 3 で同じ条件に関して同じオペレータが 2 回以上実行されることはないものとする. これは無限ループを防ぐためである.

Step 2 は様々な方法が考えられるが, ここではもっとも詳細な条件部を持つオペレータを選ぶこととする. これは WM の状態をもっとも詳細に指定しているオペレータがその状況で実行されるべきであるという考え方に対応する方法である. 条件部の詳細度は以下のように定義する.

1. 条件節の数がより多い条件部は, 他の条件部より詳細である.
2. 条件節の数が同じ条件部については, 値 (V) を指定した属性 (A) の数がより多い条件部を他より詳細とする.
3. 条件節の数, 値を指定した属性の数ともに同じ条件部の詳細度は等しいとする.

条件部のもっとも詳細なオペレータを選べなかった時は, 問題解決を止めるものとする.

このような文脈を無視したコンフリクトの解消法では, うまく問題解決を行えなくなることがある. PS モデルではこれを積極的にインタビューに利用する.

5.3.2 問題解決とインタビューの機会

(1) 問題解決とその失敗

PS モデルを用いた問題解決では以下の結果が考えられる.

1. 正しい答が得られる.
2. 答が出ない.
3. 答えは出るが間違っている.

答が出ないのは, 適用できるオペレータがなくなったときである. これにはコンフリクトセットが空になった場合と, もっとも詳細な条件部を持つオペレータが複数ある場合の二つがある. 答を誤るのは, オペレータのどれかに誤りがあって誤った方向へ推論を行なったときである. 以下, このように答が出ないときや誤っているときのことを問題解決の失敗と呼ぶことにする. なお答の正誤にかかわらず, 最終的にはコンフリクトセットが空になって問題解決が停止する.

先に定義した PS モデルの動作から, 問題解決の失敗が起こる原因を次のように整理できる.

1. 条件部の照合とコンフリクトセットへの登録において,
   a. 登録されるべきオペレータが登録されなかった.
   b. 登録されるべきでないオペレータが登録された.
2. コンフリクトの解消時に,
   a. 選ばれるべきでないオペレータが選ばれた.
3. オペレータの実行時に,
   a. 誤った状態節が WM に追加された.
   b. 必要な状態節が WM に追加されなかった.

ここで 2 による失敗は, 条件部の詳細度の定義から次のように分類できる.

1. 条件節の数に関して,
   a. 選ばれるべきオペレータの条件節の数が十分でなかった.
   b. 選ばれるべきでないオペレータの条件節の数が多過ぎた.
2. 値を指定された属性の数に関して,
   a. 選ばれるべきオペレータの条件部の, 値を指定された属性の数が十分でなかった.
   b. 選ばれるべきでないオペレータの条件部の, 値を指定された属性の数が多過ぎた.

1, 3 は WM を仲介として相互に依存している. しかしこれら失敗原因は, 最終的にはオペレータセットの不備に帰結される. 例えば, 1a が起こる理由として

- 必要な状態節が WM にない
- 登録されるべきオペレータの条件部に誤りがある.

がある. ここで前者の必要な状態節は, 他のオペレータが結論しなかったために WM にないと考えられる. すなわちいずれにしてもオペレータセットの不備によるものといえる.

(2) インタビューの機会

5.2.2 節で述べたように, 問題解決の失敗はインタビューの際の質問生成の良い手掛りである. ここでは PS モデルを題材として, 問題解決の失敗と質問の生成について検討する.

問題解決器を用いることによってインタビューの際に受け手に与えることができる示唆としては,

- 問題解決を継続できなくなったことの提示
- 問題解決過程の提示

の 2 つが考えられる. PS モデルの場合, 問題解決が継続できなくなったことは, コンフリクトセットが空になることによって検出できる.

先に述べたように問題解決の失敗には, 答が出ない時と答が誤っている時の 2 つがある. ここでインタビューの際に問題解決システムの立場から問題解決の失敗を自動的に検出できるのは, 前者のみである. 後者については何らかの形で, インタビューの受け手から誤りを指摘される必要がある. すなわち, 問題解決の失敗には

1. インタビューシステムが自動的に検出できる場合と
2. インタビューの受け手に指摘してもらう必要がある場合

の 2 つが考えられる.

一方, 問題解決の失敗が検出されたとき, その時点で正しい答をインタビューの受け手から得ることができるかどうかも重要である. 正しい答を得られる時は, その答から逆向きにオペレータを適用していくことによって必要な状態節を同定し, オペレータセットの不備を修正することができる (例えば PDS[Shapiro82]). 一方, 正しい答を得られない時は, インタビューシステム側で問題解決の妨げとなったと思われる原因を仮定し, 受け手と質疑応答を行ないながらオペレータセットの不備を発見していく必要がある.

以上を整理すると, インタビューシステムが行なう示唆, 質問は, 問題解決器の立場からは次のように分類できる.

1. 問題解決が継続できなくなった時,
    a. 継続できなくなったことを提示する.
    b. 結果が正しいかどうか質問する.
    c. 不足していると思われる問題解決に必要な情報 (状態節) を質問する.
    d. 誤っていると思われるオペレータを提示し, 正しいかどうか質問する.
2. 問題解決を行ないながら,
    a. 問題解決過程を提示する.
    b. 中間結果が正しいかどうか質問する.

ただし, 2 は問題解決過程が非常に複雑なときは, すべてを提示することはできず, 適当な中間結果を設定する必要がある. これは結局, 中間結果ごとに 1 の示唆, 質問を行なうことに帰結できる.

以下, 具体的に PS モデルを使って 1 を検討する.

コンフリクトセットが空になることから, 問題解決が継続できなくなったことがわかる. この時, 次のようにして不足していると思われる状態節や誤っていると思われるオペレータを求めることができる.

まず, 問題解決の一周期ごとに, 次の 3 つ組を第 i 周期の履歴として保存しておく.

$$(WM_i \ \ CS_i \ \ op_i)_i$$

ここで WM_i, CS_i, op_i はそれぞれ, 第 i 周期の WM, コンフリクトセット, 実行されたオペレータを示す. 第 0 周期の履歴は $(WM_0$ 空 なし$)_i$ になる.

次に各周期の履歴ごとに以下の手続きを行なう.

Step 1 WM_i について知識ベース中のすべてのオペレータから現在成立していない条件節を取り出す. 得られた条件節集合を NC_i とする. すなわち,

$$NC_i = \{ c_i \mid c_i \text{ は第}i\text{周期で成立していない条件節}\}$$

Step 2 NC_i の条件節の内, 他のオペレータの結論となっていない状態節に対応する条件節の集合を NC_i-, 逆に他のオペレータで結論される状態節に対応する条件節の集合を NC_i+ とする. すなわち,

$$NC_i- = \{c_i | c_i \in NC_i \text{ かつ } c_i \text{ は他のオペレータで結論されない}\}$$
$$NC_i+ = \{c_i | c_i \in NC_i \text{ かつ } c_i \text{ は他のオペレータから結論される}\}$$

Step 3 NC_i+ の各条件節を結論するオペレータの集合を求める. この集合を NOP_i と呼ぶ.

$$NOP_i = \{op_i | op_i \text{は } NC_i + \text{ の条件節のどれかを結論する}\}$$

このようにして得られた集合, NC_i は問題解決過程で成立しなかったために, その継続を妨げた条件節の集合である. また, NOP_i は実行されなかったために, NC_i の各条件節が成立できなかったオペレータの集合となる. また, NC_i- は他のオペレータから結論されないため, 今後も成立する見込みのない条件節の集合といえる.

このようにして得られた各周期での状態を用いてインタビューを行なう. その際, 履歴の最初から順に質問していく方法と, 履歴をさかのぼっていく方法の二つが考えられる. 前者は, インタビューの受け手に問題解決過程を示しながらインタビューを進めることに対応し, 受け手が良く考え易いという利点がある. しかし先に述べたように問題解決過程が多数の周期からなるときは, 質問が多くなることが予想される. 一方後者は, 問題解決

の継続を狙ったインタビューと考えられ, 質問は少なくなるがそれまでの過程が示されないため, 受け手にとっては考えにくくなることが予想される.

まず, WM_iの利用を考える. WM には問題解決の中間結果が蓄えられている. したがってこれを受け手に提示することにより,

- 存在すべきでない状態節
- 存在すべき状態節

を指摘してもらえる可能性がある. 前者については履歴をたどることにより, 直接の原因となっている周期を特定することができる. そしてこの周期から再び, インタビューを試みることができる. 一方, 後者の指摘は, NC_iの中で特に成立しているべき条件節が指定されたことに対応する.

NC_i-の各条件節はオペレータによって結論されないもので,

- オペレータのどれかが結論すべきである.
- それを結論するオペレータが定義されていない.
- 対話などを通して外部から獲得すべき情報である.

などの可能性が考えられる. インタビューの立場からは, システムが現在必要としている情報と考えることができる. NC_i-を受け手に提示することにより, いずれであるかを指定してもらえる可能性がある.

NC_i+の条件節に関しては, 以下の可能性が考えられる.

- 結論されているべきである.
- 実は対話などを通して外部から獲得すべき情報である.

前者はこれまで実行されたオペレータが結論すべき場合, それを結論するオペレータが実行されていない場合, あるいは結論すべきオペレータが定義されていない場合が考えられる. op_i, NOP_iなどとあわせて受け手に提示することにより, いずれであるかを指示してもらえる可能性がある.

NOP_iのオペレータについては, 実行されているべきオペレータである場合がある. この場合, NOP_iを受け手に提示することによってそのようなオペレータの指摘を受けられる可能性がある.

CS_iをWS_iとあわせて提示することにより, 不備なオペレータを直接特定することが可能である. 特にもっとも詳細な条件部を持つオペレータが複数あったために, コンフリクトの解消ができなかった場合は, CS_iを受け手に提示することによって

- 詳細度の不足している条件部を持つオペレータ
- 詳細過ぎる条件部を持つオペレータ
- 定義されていないオペレータ

の指摘を受けられる可能性がある.

以上の議論を整理すると, PS モデルをインタビューシステムの問題解決機能として用いることにより, 表 5.1 に示すような示唆, 質問を行なうことができる.

表 5.1 PS モデルとインタビュー

提示するもの	提示の意味	答の可能性	備考
WM_i	問題解決の中間結果の提示	存在すべき, あるいはすべきでない状態節の指摘	存在すべきでない状態節から次に注目すべき周期 i を特定できる.
NC_i-	問題解決に必要とされている情報の提示	オペレータの不備あるいは外部から獲得すべき情報であることの指摘	
NC_i+	問題解決に必要とされている情報の提示	オペレータの不備あるいは外部から獲得すべき情報であることの指摘	op_i, NOP_i と合わせることにより, 実行されたオペレータの結論部の誤り, 実行されるべきオペレータが特定できる可能性がある.
NOP_i	問題解決に関与しなかった (できなかった) オペレータの提示	実行されるべきオペレータの指摘	
CS_i	十分な条件部を持たないオペレータの提示	条件部の詳細度の過不足の指摘	もっとも詳細な条件部を持つオペレータが複数あるとき, 特に有効である.

さて, 次に提示法について検討する.

特に WM_i は多数の状態節からなることが予想されるので, 提示するものの絞り方が課題となる. 先に述べたように, 問題解決過程を提示することの利点は, インタビューの受け手を問題解決過程に引き込み, よく考えさせることができる点にある. そこで提示に際しては, その時点でもっとも関係が深いと考えられるものを重点的に提示することによって受け手が思考に集中しやすくできると考えられる.

ただし, もっとも関係が深いものは問題解決の進展に応じて常に変化することが予想される. PS モデルの場合, 単純にオペレータの適用を繰り返すのみなので, これを決定するは困難である. そこで以下のような方法を用いて, 関係が深いと思われるものを求めて提示することが考えられる.

まず, WM 中の各状態節に生成順に番号を付けてやる. この番号のことをタイムタグ (Time Tag, 以下 TT と略す) と呼ぶ. より新しい状態節にはより大きな TT が付くことになる. また, 問題解決の過程においては関係の深い状態節が時間的に連続することが予想される. そこで例えば WM_iを提示するときは, 大きな TT を持つ状態節を選べば, その周期に比較的関係が深い状態節が提示できると思われる.

NC_i+, NC_i-に関しては, それらの条件節の重要度を次のようにして推定することが考えられる. 二つの集合の各条件節について, それが成立している (WM に対応する状態節が存在する) ものとして CS を作ってみる. このとき, コンフリクトセットが大きくなる条件節は, 多くのオペレータが必要としていると考えられる. そこでそのような条件節を提示し, 質疑応答を行なえば, 問題解決に貢献できると思われる.

別の方法としては, 同様にして作った各 CS について, より新しい状態節を条件部で参照しているオペレータが多い CS に対応する条件節を提示することが考えられる. これは第 i 周期で, 問題解決に比較的関係が深いと思われ, しかも必要とされている条件節を提示することに対応する.

NOP_iの場合は, 各オペレータが結論する状態節について, NC_i+, NC_i-の場合と同様の操作を行なうことが考えられる. その結果選ばれた状態節を結論するオペレータの実行は, その時点での問題解決に必要性が高いとみなすことができる. またさらに選ばれたオペレータの各条件部の内, WM_i(その周期での WM) のより新しい状態節を参照している条件部を持つオペレータを選べば, それはその時点により関係が深いとみなすことができる.

CS_iについても同様に, より新しい状態節を条件部で参照しているオペレータを提示すれば, その時点で問題解決に深い関係があると思われるオペレータを提示したことになる.

以上, いずれにしてもこのような方法は, 問題解決に関係の深いものを正確に選び出すことはできない. しかしこれは問題解決器として PS モデルという簡単な汎用のものを用いたためである. よりインタビューの対象に適した問題解決器を用いることにより, 解決可能と考えられる.

5.3.3 学習機能の役割り

本節では, PS モデルに学習機能を付加することによる, インタビューの効率改善について検討する.

インタビューの効率としては,

1. 問題解決にかかる時間の観点からの効率
2. 質問生成にかかる時間の観点からの効率
3. 質疑応答から得られる知識の良し悪しに関する効率

が挙げられる. PS モデルを問題解決器として用いるインタビューシステムでは, これらをさらに次のように分類することができる.

1. 問題解決にかかる時間に関して

a. CS を作る時間

b. コンフリクトの解消にかかる時間

c. オペレータを実行するのにかかる時間

2. 質問生成にかかる時間に関して

a. NC_i-, NC_i+, NOP_iを作るのにかかる時間

b. それらから提示するものを決定するのにかかる時間

c. さらに実際の質問を作成するのにかかる時間

3. 得られる知識の良し悪しに関して

a. 質問は的を射たものか.

b. 質問が多過ぎないか.

c. 似たことを繰り返し聞いていないか.

d. 得られた知識が問題解決に貢献するか.

以下, 順に学習機能とその利用を検討する.

まず, 問題解決にかかる時間に関してであるが, PS モデルの場合はその機構が単純であるため, 各ステップごとの効率改善は難しい. しかし複数の周期にまたがって頻繁に行なわれる類似の推論を一つのオペレータにまとめることによって, 問題解決全体のステップ数を削減することが考えられる. これはある特定問題解決に特に有効なオペレータを特別に定義していくことに当たる. この考え方はチャンキング (Chunking) と呼ばれ, SOAR [Laird87] などで試みられている.

チャンキングを行なうことによって, インタビューに次のような効果がもたらされることが期待される.

- チャンキングによって作成されたオペレータを中心に質疑応答を進めることによって問題に固有の知識を議論することができる.
- 問題解決の周期数が少なくなるため, 考慮すべき NC_i-, NC_i+, NOP_iの数が減少する. これは最終的には質問の減少につながる.

次に質問生成にかかる時間について検討する. 5.3.2 で述べたように, PS モデルではインタビューの受け手に提示するものを選択する方法に問題がある. 学習機能の利用としては, この選択過程にそれ以前に得られた情報を利用することによってより良いものを選ぶことが考えられる. 具体的にはまず, 関係の深い状態節間をポインタで結んで行く. そしていくつかの状態節のグループについて一般化を行なう. 提示する際には一般化された状態節のグループを提示する.

関係の深さは問題に固有の尺度で測られねばならない. PS モデルの場合, オペレータが問題に対する知識を含んでいるのでこれを利用することが考えられる. 例えば, CS_i中の

オペレータについて, いくつかの状態節が複数のオペレータの条件部で同時に参照されている場合が考えられる. このような場合, それらの状態節はまとめて参照すべきものであり, 関係が深いと判断できる.

このような機能を PS モデルに適用することによって, インタビューに次のような効果がもたらされることが期待される.

- 示唆が断片としてではなく, 意味のあるまとまった知識として提示される.
- 一般化によって, 些細な情報を示唆から除くことができる.

得られる知識の良し悪しに関しては, 最終的には問題解決能力の向上度で判断することができる. これはこれまで述べてきた学習機能によってもたらされるより良い質問, 示唆を通じて改善できる.

最後に, PS モデルで考えられる学習方法とその効果を表 5.2 にまとめる.

表 5.2 PS モデルと学習

学習方法	インタビューにもたらす効果
チャンキング	チャンキングによって作成されたオペレータを中心に質疑応答を進めることによって問題に固有の知識を議論することができる. 問題解決の周期数が少なくなり, 質問が減少する.
ポインタ	示唆が断片としてではなく, 意味のあるまとまった知識として提示される.
一般化	些細な情報を示唆から除くことができる.

5.3.4 インタビューシステムの一般アーキテクチャ

以上の検討をもとに設計したインタビューシステムの一般アーキテクチャを図 5.3 に示す. 図中の各モジュールの役割を以下, 順に示す.

(1) Problem Solver

インタビューの際の問題解決を担当するモジュールである. インタビューの対象に適したものを用いる.

(2) Watcher

Problem Solver の動作を監視するモジュールである. 問題解決が継続できなくなったら,

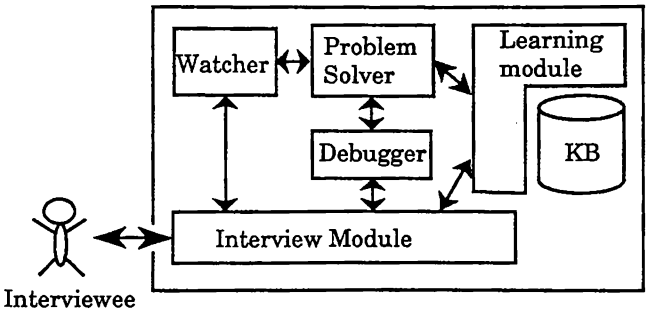


図 5.3 インタビューシステムの一般アーキテクチャ

- 問題解決過程の履歴
- 必要とされる知識

を Interview Module に渡し, 質問を促す.

(3) Debugger

Problem Solver を制御し, 部分的に問題解決を試みることを可能にするモジュールである. Interview Module が問題解決過程の制御や, 獲得された知識の試行をするために使用する.

(4) Interview Module

インタビューシステム全体の動作を制御するモジュールである. Watcher 経由で Problem Solever を監視し, 必要に応じてインタビューの受け手 (Interviewee) と質疑応答を行なう. また Debugger を使って, 問題解決を試行し, 獲得された知識についてさらに質疑応答を試みる.

(5) Learning Module と 知識ベース (KB)

知識ベース (KB) にはインタビューのための知識や獲得された知識が蓄えられる. Learning Module は知識の参照のされ方を監視し, 必要に応じて知識間のリンクを生成, 管理する.

5.4 結言

以上, 本章ではインタビューシステムのアーキテクチャについて検討した. インタビューにおいては, 問題解決とその過程の学習が重要な役割を果たしている. これについてまず

PS モデルを用いて検討した. そしてインタビューシステムの一般アーキテクチャを提案した.

インタビューシステムに関する研究は, 十分になされているとは言えない状況にあり, 一般アーキテクチャの提案も抽象的にならざるを得なかった. 今後もさまざまな分野でインタビューシステムの検討を行ない, アーキテクチャを洗練していくことが課題として残されている.

第 6 章

結論

本論文では, 知的インタビューシステムについて, 以下の 4 項目を検討した.

(1) 静的解析を用いたインタビューシステム, *I²S-LD*

(2) 動的解析を用いたインタビューシステム, *I²S-DB*

(3) *I²S-LD* に付加した学習機能

(4) インタビューシステムの一般的なアーキテクチャ

以下, 各章で述べた本研究の成果をまとめる.

第 2 章 においては, インタビューにおける質問戦略の役割を検討した. そしてデータベースの論理設計を題材として試作した知的インタビューシステム *I²S-LD* について述べた. *I²S-LD* はユーザから獲得したドメインに関する知識 (ドメインモデル) をプラン構造という形式で表現する. このドメインモデルに 4 種類の質問戦略を用いることにより, *I²S-LD* は論理設計に当たって様々な示唆をシステムのユーザに与えることができる. この章ではまた, 質問戦略を用いたインタビューシステムの内部機構の一般化を試みた. そして, ドメインモデルの表現形式をネットワーク表現に限ったインタビューシステムのためのシェル, *SIS* を構成した. 7 個の質問戦略プリミティブモジュールを提案し, *I²S-LD* と MORE の *SIS* による構成例を通してその有効性を示した.

第 3 章 では, 静的解析を補完する動的解析を取り上げ, その実現に必要な機構, 知識を検討した. そしてデータベースの設計・構築を題材として, データベースを試作しながらインタビューを進める *I²S-DB* について述べた. *I²S-LD* に DBMS インタフェースを付加し, データベース試作に必要な知識を整備することによって, インタビューの結果からデータベースを実際に試作し, システムのユーザに使ってもらうことが可能となった. また, ユーザが実行する検索要求文を解析, ドメインモデルと比較することによって, ドメインモデルに存在する不備を検出する方法を述べた. 検出された不備をもとにさらにユーザにインタビューを実施することにより, ユーザが納得のいく論理設計が行なえるようになった.

第 4 章 においては, インタビューシステムの学習機能について考察した. インタビューシステムにおける学習としてはインタビュー技術の獲得とインタビューを通じて得たド

メインの知識の再利用が考えられる．本論文では後者を取り上げ，その機構およびインタビューにもたらす効果について検討した．そして I^2S-LD にドメイン知識の学習機能を付加することを試みた．プラン構造で表現されたドメインモデルの主要要素である動詞，名詞の辞書を動的に再構成し，個々のドメインモデルの抽象化であるドメインフレームと関連付ける機構を示した．そして類似ドメインの概念を導入し，これをインタビュー時に利用することによってよりよい示唆がユーザに与えられることを示した．

第5章では，上記の各章での検討をもとにインタビューシステムの一般アーキテクチャの設計，検討を行なった．まず従来の知識獲得支援システムや I^2S-LD，I^2S-DB の比較から，インタビューシステムにおける問題解決機能と学習機能の必要性を示した．次に汎用の問題解決器を定義し，その上で問題解決過程がインタビューに果たす役割を具体的に議論した．また学習機能の考え方を一般的に整理した．最後にインタビューシステムの一般アーキテクチャを提案した．

以上，本論文ではインタビューシステムの持つべき機構および知識について検討した．コンピュータの応用分野は今後ますます広がっていくものと考えられる．その際，ユーザとの円滑な意思疎通を図る一手段としてインタビューシステムの考え方は欠かすことのできないものとなるであろう．しかしながらインタビューシステムの研究は始まったばかりである．今後も様々な分野でインタビューシステムを試作，検討を行ない，インタビューに必要な機構や知識をより一般的に議論していく必要性が残されている．

謝辞

本研究の全過程を通して, 直接懇切なる御指導, 御鞭達を賜った大阪大学産業科学研究所角所収教授, ならびに溝口理一郎助教授に衷心より感謝の意を表する.

大学院前期, 後期両課程において電子工学一般および各専門分野に関し, 御指導と御教示を賜った大阪大学電子工学科児玉慎三教授, 寺田浩詔教授, 白川功教授, 西原浩教授, 浜口智尋教授, 吉野勝美教授, 大阪大学電子ビーム研究施設裏克己教授, 塙輝雄教授に厚く御礼申し上げる.

本研究の遂行にあたり, 終始有益な御助言と御鞭達を頂いた関西大学電子工学科野村康雄教授, 郵政省通信総合研究所音声研究室柳田益造室長, ならびに, 角所研究室の山口高平助手, 山下洋一技官に厚く感謝の意を表する.

筆者の研究仲間であった和田充弘氏 (現, 野村総合研究所), 沼田薫氏 (現, 野村総合研究所), 松山映二氏 (現, 関西大学大学院生), 山田勉氏 (現, 松下電工), 松田勝志氏, 高岡和宏氏には様々な面で御協力を頂いた. また, 池田満氏, 辻野克彦氏, 堀雅洋氏をはじめとする角所研究室の諸氏, ならびに光成和子嬢, 福田恵美嬢には種々の面で御世話になった. ここに記して深く感謝する次第である.

参考文献

[国研64] 分類語彙表, 国立国語研究所, 1964.

[川口88] 川口敦生, 溝口理一郎, 角所収, 山田勉, 野村康雄, 馬場富男, 中島裕生, "設計時の仕様獲得を支援する知的インタビューシステム I^2S/D — 油圧回路の設計に関して —", 情報処理学会知識工学と人工知能研究会資料, 59-16, 1988.

[穂鷹81] 穂鷹良介, データベースの論理設計, 情報処理学会, 1981.

[Chen76] Chen, P. P., "The Entity-Relationship Model Toward a Unified View of Data", *ACM Transaction of Database Systems*, 1(1):9–36, 1976.

[Codd70] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, 13(6):337–387, 1970.

[Date84] Date, C. J., データベース・システム概論, 丸善株式会社, 1984.

[Davis82] Davis, R. and Lenat, D. B., *Knowledge-Based Systems in Artificial Intelligence*, MacGrow-Hill, 1982.

[Dyer83] Dyer, M. G., *In-Depth understanding*, p.p. 385–435, The MIT Press, 1983.

[Eshelman86] Eshelman, L. and McDermott, J., "MOLE: A knowledge acquisition tool that uses its head", In *Proc. of AAAI'86*, p.p. 950–955, 1986.

[Kahn85] Kahn, G., Nowlan, S. and McDermott, J., "Strategies for Knowledge Acquisition", *IEEE PAMI*, PAMI-7(5):511–522, 9 1985.

[Kaplan83] Kaplan, S. J., "Cooperative Responses From a Portable Natural Language Database Query System", In *Computational Models of Discourse*, p.p. 209–266, The MIT Press, 1983.

[Laird87] Laird, J. E., Newell, A. and Rosenbloom, P. S., "SOAR: An Architecture for General Intelligence", *Artificial Intelligence*, 33:1–64, 1987.

[Lederberg64] Lederberg, J., *DENDRAL-64: A system for computer construction, enumeration and notation of organic molecules as tree structures and cyclic graphs*, NASA, 1964.

[Mitchell86] Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T., "Explanation-based generalization:A unifying view", In *Machine Learning*, p.p. 47–80, Springer-Verlag, 1986.

[Mizoguchi87] Mizoguchi, R., Kobayashi, H., Isomoto, Y., Toyoda, J. and Kakusho, O., "Interactive Synthesis of Conceptual Schema Based on Queries — Towards an expert system of relational database design —", *Journal of INFORMATION PROCESSING*, 8(3):207–216, 1987.

[Shapiro82] Shapiro, E. Y., *Algorithmic Program Debugging*, The MIT Press, 1982.

[Shortliffe76] Shortliffe, E. H., *Computer-based medical consultations: MYCIN*, American Elsevier, 1976.

[Winston77] Winston, P. H., *Artificial Intelligence*, Addison-Wesley, Reading, Mass, 1977.

付録 A

MOREの*SIS*による実現例

*SIS*を用いて MORE を実現するための知識の記述例および対話例を示す. 本文中で述べたように, *SIS*を用いてインタビューシステムを構成するには 4 種類の知識を記述すればよい. 以下, prototype, task, ask, apply の順に MORE を実現するための定義について述べ, そうして実現されたシステムとの対話例を示す. ただし例は, 仮説の区別戦略に関するもののみを示す.

MORE のドメインモデルは, 主として仮説および徴候からなる. 図 A.1 にそれらの prototype 定義の例を示す. 図中, 各 prototype の上位概念 (super_class) となっている noun は, *SIS*に組み込みの prototype である. また, s_h_link はドメインモデル中の仮説および徴候を結び付けるリンクの prototype である.

```
prototype hypothesis
   super_class noun
   attributes
      [link_to_symptom, frequency_cond].
prototype symptom
   super_class noun
   attributes
      [link_to_symptom, link_to_hypothesis, link_to_test].
prototype s_h_link
   super_class noun
   attributes
      [symptom,hypothesis,condition,attribute,value].
```

図 A.1 MORE のための prototype 定義例

システムの振る舞いは task を記述することによって定義できる. MORE の場合, インタビューは,

(1) 初期ドメインモデルを構築する.

(2) 8 つの質問戦略をドメインモデルに適用し, モデルを洗練する.

という順に進む. (2) は *SIS*に組み込みの QPM を用いることによって適用対象の検出を自動化できる. 図 A.2に task の定義例を示す.

```
task construct_KB_in_MORE_manner                    ┐
   sub_task                                         │
      make_initial_domain_model,                    │
      while attentionListNotNil                     ├ (a)
      do check_domain_model,                        │
           generate_rule_set,                       │
           check_rule_set.                          ┘
task make_initial_domain_model                      ┐
   sub_task                                         │
      ask_initial_set_of_symptoms,                  ├ (b)
      ask_initial_set_of_hypothesis,                │
      ask_initial_association.                      ┘
task check_domain_model                             ┐
   sub_task                                         ├ (c)
      while getAttention -> A                       │
      do process_attention(A).                      ┘
```

図 A.2 MORE のための task 定義例

各 task 定義は, sub-task の列挙およびいくつかの制御記述からなる. また, 質問を行なうための ask 呼び出し (callAsk) および *SIS* ライブラリー呼び出し (callProlog) が用意されている. 図 A.2(a) の定義によって, attention が生じている間 (while attentionListNotNil), ドメインモデルの検討 (check_domain_model), ルールの作成 (generate_rule_set), ルールの検討 (check_rule_set) が行なわれる.

さて, インタビューが開始され, ドメインモデルが構築されていくと随時 QPM が起動され, attention が生成される. 生成される attention の処理法を記述することによって質問戦略が定義される. この記述は attention 処理も task と考え, task 記述で行なう. 図 A.3 に仮説の区別戦略に対応する task 記述の例を示す.

```
task differentiation2                                                    ┐
   attention sameSource,Symptom,[PathList]                               │
   sub_task                                                              │
      callProlog plannerGetPathEndList(                                  │
                          PathList,EndList),                             ├ (a)
      callAsk ask_differentiation_start(Symptom,EndList),                │
      differentiation3(Symptom,EndList,HypoNot),                         │
      when HypoNot=[H1,H2|_]                                             │
      do symptomDistinction(Symptom,HypoNot).                            ┘
task differentiation3(Symp,EndList,ResultList)                           ┐
   sub_task                                                              │
      allocVariable(RR),                                                 │
   ( while getOneElement(EndList) -> H                                   ├ (b)
      do callProlog remove(H,EndList,HR),                                │
         callAsk ask_new_symptom_by_diff(Symp,H,HR,R),                   │
         when R == [] do pushToVariable(R,RR)                            │
   ),getVariable(RR) -> ResultList,                                      │
      freeVariable(RR).                                                  ┘
```

図 A.3 differentiation 戦略のための task 定義例

図 A.3(a) のように attention (この例の場合, sameSource) を指定することによって,

attention 処理法の定義になる. この定義によって, ある徴候 (Symptom) からいくつかのパス ([PathList]) が伸びているときに,

(1) パスの末端, すなわち仮説の集合を求め (plannerGetPathEndList),

(2) differentiation3 を呼び出して仮説の区別を行なう.

(3) もし, 区別されなかった仮説が二つ以上あるとき (HypoNot=[H1,H2!_]) は, 徴候の区別を行なう.

ということが行なわれる.

differentiation3 (図 A.3(b)) では質問および結果の処理が記述されている. 区別すべき仮説の集合 (EndList) から順に仮説を取り出し (getOneElement(EndList)), 関連する徴候を質問する (ask_new_symptom_by_diff). そして徴候が得られなかった仮説 (when R == []) を ResultList に返すといったことが記述されている.

実際の質問作成および結果の処理は ask で定義される. 図 A.4 に仮説の区別のための定義を示す. (a) で区別されていない仮説 (E) が受け手に提示され, (b) である仮説 (H) と他の仮説集合 (RestHypoS) を区別する徴候が質問される. 徴候 (SS) が得られたときは, s_h_link を作成し, 対象となっていた仮説との間に張る.

一方, attention 生成は apply 記述によって定義される. 本文中で述べたように, インタビュー中にドメインモデルが変更されたときには自動的に QPM が適用され, 場合によっては attention が生成される. apply 定義によって, 質問戦略モジュールが適用されるドメインモデル内の対象を記述する. 図 A.5に仮説の区別戦略のための定義例を示す.

仮説の区別は, ある徴候から複数の仮説が導けるときに行なう必要がある. そこでこの定義で planner に徴候 (domain(symptom)) → 徴候と仮説間のリンク (domain(s_h_link)) → 仮説 (domain(hypothesis)) のパスをたどるよう指示する. これによってある徴候から複数の仮説へのパスが存在するとき, planner の機能によって attention, sameSource が生成される. この attention は先に示した task, differentiation2 によって処理され, 仮説の区別が行なわれる.

以上が *SIS* を用いて MORE を実現するための定義例である. システムとの対話例を図 A.6に示す. 図中の (a), (b), (c) で示される対話がそれぞれ, make_initial_domain_model, check_domain_model (および differentiation2), generate_rule_set という task 定義によって行なわれる対話に対応する.

この対話例で最初に構築されるドメインモデルは本文図 2.16(a) のものである. 徴候 decrease_in_density と仮説 shale_contamination, water_influx に対して仮説の区別戦略が適用され, (b) で新しい徴候, increase_in_unemulsified_water が獲得されている.

```
ask ask_differentiation_start(S,E)                                    }
   doQandA                                                            } (a)
      showNl (E,' can give the same symptom ',S, '.').                }
ask ask_new_symptom_by_diff(Symp,H,RestHypoS,R)                       }
   doQandA                                                            }
      showNl('Can you provide a symptom associated with ',            }
          H, ' that cannot be explained by ',                         }
          or(RestHypoS), ' ?'),                                       }
      get_answer A1 in yesOrNo,                                       }
      when A1 == no                                                   }
      do callProlog R=H                                               }
      else callProlog R=[],                                           }
         showNl('Enter symptoms associated with ', H,'.'),            }
         get_answer SS in symptom,                                    }
         while getOneElement(SS) -> S                                 } (b)
         do create domain(s_h_link) -> L1,                            }
            assign S to L1 -> symptom,                                }
            assign H to L1 -> hypothesis,                             }
            assign L1 to S -> link_to_hypothesis,                     }
            assign L1 to H -> link_to_symptom,                        }
            showNl('May ',S,' be a hypothesis to ', Symp,' ?'),       }
            get_answer A2 in yesOrNo,                                 }
            when A2 == yes                                            }
            do create domain(s_h_link) -> L2,                         }
               assign Symp to L2 -> symptom,                          }
               assign S to L2 -> hypothesis,                          }
               assign L2 to Symp -> link_to_hypothesis,               }
               assign L2 to S -> link_to_symptom.                     }
```

図 A.4 differentiation 戦略のための ask 定義例

```
apply planner
   object
      domain(symptom)->link_to_hypothesis
      ==>domain(s_h_link)->hypothesis
      ==>domain(hypothesis).
```

図 A.5 differentiation 戦略のための apply 定義例

```
Please tell me any symptoms you know in this domain.
> increase_in_viscosity.
And please tell me all hypothesis in this domain.
> shale_contamination, water_influx.
Please select symptoms associated with
shale_contamination.
   1.  increase_in_viscosity
> 1.
Please select symptoms associated with water_influx.
   1.  increase_in_viscosity
> 1.
shale_contamination and water_influx can give the
same symptom increase_in_viscosity.  Can you provide a
symptom associated with shale_contamination that
cannot be explained by water_influx ?
> no.
Can you provide a symptom associated with water_
influx that cannot be explained by shale_contamination ?
> yes.
Enter symptoms associated with water_influx.
> increase_in_unemulsified_water.
May increase_in_unemulsified_water be a hypothesis to
increase_in_viscosity ?
> no.
Ok.  I will make a rule KB.
Wait a moment, please.

If   increase_in_viscosity
Then shale_contamination
If   increase_in_viscosity
Then water_influx
If   increase_in_unemulsified_water
Then water_influx
```

(a) (b) (c)

図 A.6 *SIS* によって実現された MORE との対話例

付録 B

DBMS シミュレータの仕様

I^2S-*DB* の DBMS として使用した, Prolog による DBMS シミュレータの仕様を示す. まず, データ操作言語 (DML) の構文則を示し, いくつかの例を通してその意味を述べる. 次に, データ定義言語 (DDL) に関して同様に記述する.

2.1 データ操作言語

以下に DML の構文を BNF 記法で示す.

< *data manipulation* >::=
    < *retrieval operation* > | < *update operation* > |
    < *insert operation* > | < *delete operation* >
< *retrieval operation* >::=
    UPDATE < *relation* >
    SET < *attribute* >=< *arithmetic expression* > |
    UPDATE < *relation* >
    SET < *attribute* >=< *arithmetic expression* >
    WHERE < *conditions* >
< *insert operation* >::=
    INSERT_INTO < *relation* >:< *constants* >
< *delete operation* >::=
    DELETE < *relation* > |
    DELETE < *relation* > WHERE < *conditions* >
< *select blocks* >::=
    < *select blocks* > |
    < *select block* > UNION < *select blocks* >

< *select block* >::=
SELECT < *target list* > FROM < *relations* > |
SELECT < *target list* >
FROM < *relations* > WHERE < *conditions* > |
SELECT < *target list* >
FROM < *relation* > GROUPED_BY < *attribute* > |
SELECT < *target list* > FROM < *relation* >
GREOPED_BY < *attribute* > HAVING < *function* > |
SELECT < *target list* > FROM < *relation* >
GROUPED_BY < *attibute* > WHERE < *condition* >

< *target list* >::=
< *target* > | < *target* >,< *target list* >

< *target* >::=
< *attribute* > |
< *relation* > . < *attribute* > | < *function* >

< *conditions* >::=
< *condition* > | < *condition* >,< *conditions* >

< *condition* >::=
< *simple condition* > | < *chaining condition* > |
< *join condition* > | < *set requirement* >

< *simple condition* >::=
< *atribute* >< *comparison operator* >< *constant* > |
< *relation* > . < *attribute* >< *comparison oprator* >< *constant* >

< *chaining condition* >::=
< *attribute* >< *chainingoperator* >< *select block* >

< *join condition* >::=
< *relation* > . < *attribute* >< *comparison operator* >< *relation* > . < *attribute* >

< *set requirement* >::=
(< *attributes* >) < *chaining operator* >< *select block* >

< *arithmetic expression* >::=
< *constant* > |
< *something* >< *arithmetic operator* >< *something* >

< *chaining operator* >::=
IN | NOT_IN |

< | > | = | <= | >=

< *comparison operator* >::=

< | > | = | <= | >=

< *arithmetic operator* >::=

+ | − | * | /

< *function* >::=

< *textstylebuiltin function* > (< *attribute* >)

< *builtin function* >::=

CNT | MAX | MIN | SUM | AVG

< *something* >::=

< *integer* > | < *attribute* >

< *relations* >::=

< *relation* > | < *relation* >,< *relations* >

< *relation* >::=

< *relation name* >

< *attributes* >::=

< *atribute* > | < *attribute* >,< *attributes* >

< *constants* >::=

< *constant* > | < *constant* >,< *constants* >

< *constant* >::=

< *atom* > | < *integer* >

この DML は SQL に準拠しており, その機能には, データの更新 (UPDATE)・追加 (INSERT)・削除 (DELETE), データの検索 (SELECT) がある. それぞれの操作を行うためには, UPDATE,INSERT,DELETE,SELECT に続く 1 ブロックが必要である. また < *relation* > は関係データベースでの表の名前を表し, < *attribute* > はその表の属性, < *condition* > は条件を示す.

以下に, それぞれの機能の例とその説明を示す.

```
UPDATE suppliers
SET status = 2 * status
WHERE city = Paris
```

これは,「suppliers という表の city という属性が Paris であるデータの status を 2 倍にする.」という操作である.

```
INSERT_INTO parts:(p7,washer,grey,2,athens)
```

これは,「 parts という表に, (p7,washer,grey,2,athens) というデータの組を加える. 」という操作を表す.

```
DELETE parts
WHERE weight>17
```

これは,「 parts という表で, weight が 17 以上であるデータの組を削除する. 」という操作である.

```
SELECT warehouse
FROM stock
WHERE part = p20
```

これは,「 stock という表の, part が p20 であるデータの属性 warehouse の値を求める. 」という操作である.

2.2 データ定義言語

以下に DDL の構文を BNF 記法で示す.

< *data definition* >::=
    DATABASE_DEFINITON
    < *define blocks* >
    END_DATABASE_DEFINITION.
< *define blocks* >::=
    < *define block* > | < *define block* >, < *define blocks* >
< *define block* >::=
    TABLE < *table name* >
    ATTRIBUTES < *target list* >
    KEY < *key list* >
< *target list* >::=
    < *target* > | < *target* >, < *target list* >
< *target* >::=
    < *attribute* >:< *type* >
< *type* >::=
    CHAR_NOT_NULL | CHAR | INT | REAL

< *define block* > は, データベースの各々1つの表の定義ブロックを表している. このブロックの TABLE の後に続く < *table name* > は, 関係データベースの表の名前である. そして, ATTRIBUTES のあとにつづく < *target list* > は < *attribute* > と < *type* > の組のリストで, < *attribute* > は表の属性を, < *type* > はそのデータ型を表す. KEY の後の < *key list* > は, それぞれの表のキー属性を表す.

以下に part, construct, stock という表を持つデータベースの定義例を示す. 例えば表 part (TABLE part) は, name, color, size を属性に持ち, キー属性は name である. また, color のデータ型は文字型, size は実数型である. name のデータ型に CHAR_NOT_NULL とあるのは, この属性のデータが空でない文字型であることを表している.

```
DATABASE_DEFINITION
    TABLE part
        ATTRIBUTES
            name:CHAR_NOT_NULL
            color:CHAR
            size:REAL
        KEY name
    TABLE construct
        ATTRIBUTES
            part_1:CHAR_NOT_NULL
            part_2:CHAR_NOT_NULL
            quantity:int
        KEY part_1,part_2
    TABLE stock
        ATTRIBUTES
            part:CHAR_NOT_NULL
            warehouse:CHAR
        KEY part
END_DATABASE_DEFINITIONS
```