知的システム構築用シェル HyperAuk の構築支援環境[†]

藤 村 茂*・富 田 昭 司**・栗 原 修 二**

The Development Environment of HyperAuk : Shell for Building Intelligent Systems

Shigeru FUJIMURA*, Shoji TOMITA** and Shuji KUWAHARA**

The paper describes attempts to provide a shell for building intelligent systems in consideration of a tight integration of the knowledge representation, the inference mechanism, the methodology of construction and the development environment. The Shell is called HyperAuk : Shell for building Intelligent Systems. It is a knowledge engineer's assistant tool that provides the graphical development environment to support the construction process, that includes planning, editing, debugging and testing phases.

HyperAuk enhances the object oriented knowledge representation auk : autonomic knowledge representation, we proposed. The methodology behind HyperAuk provides the concept of the top-down planning, the bottom-up construction and the phaseless development, and defines the sequence of its construction process. According to this concept, HyperAuk constructs the development environment to support the whole development process. This environment is also including the function of the graphical user interface construction. HyperAuk provides a uniform framework by generalization of top-down planning from the methodology view. So the environment for top-down planning is able to be used in any cases.

HyperAuk is now operational using Smalltalk-80, and its environment is constructed using the user interface management system Talkie, we also developed.

Key Words : intelligent system, knowledge representation, construction methodology, development environment

1. はじめに

従来から、知識ベースの構築手法として、宣言的な知識をボトムアップ的にまとめあげていくといった手法が提唱されている。また、現在ではインタビュー方式による知識収集の手法や、KJ 法により断片的知識の整理を行っていく方法もいくつかのシステム上に実現され^{1),2)}、ドメインシェルにおける知識獲得手法^{3),4)}として

もその有用性が示されている。しかし、知識処理のみならず、手続き的な記述も含み、ユーザインタフェースをも重視した実際の多様化しているシステム(以下、このようなシステムを知的システムと呼ぶ)全体を、ボトムアップ的な構築手法のみで実現することは困難である。

それに対して、知的システムの開発においては、トップダウン的な設計手法が有用であり、ソフトウェア工学的なアプローチとしての要求分析手法や、モジュール構造設計手法が利用されている。このような手法を取り入れることは、大規模なアプリケーションを構築するためには特に重要であり、いくつかのシステム設計方法論が提案されている⁵⁾。しかし、これらの方法論は、従来のシステム開発における開発工程/品質管理を行うための指針を示し、より上流過程の開発手法を定式化するものに留まっており、ツールの機能として組み入れられているものではない。

一方、知的システム開発の過程は、一般のシステムの

† 情報処理学会第 41 回全国大会で発表(1990・9)

* 横河電機(株)高度制御技術部 東京都新宿区西新宿 1-25-1 新宿センタービル 50 階

** 横河電機(株)オープンシステム研究所 武蔵野市中町 2-9-32

* Advanced Technology Dept., Yokogawa Electric Corp., Shinjuku-ku, Tokyo

** Open Systems Lab., Corporate R&D, Yokogawa Electric Corp., Musashino
(Received November 19, 1993)
(Revised July 29, 1994)

開発と同様に、問題の整理/モジュール設計/内部実現/テストのフェーズに分割される。しかし、人間の知的な作業をシステム化するためには、一般のシステムを開発する以上に試行錯誤性が重要である。そこで、各フェーズで得られる情報がフェーズ間で有効に用いられることが望ましい。しかし、現在、下流のフェーズを支援するツールは多々存在するが⁶⁾、すべてのフェーズをサポートするまでに至っていない。

以上のように、ボトムアップ構築、トップダウン設計の手法は、共に知的システムの構築において重要な手法であり、試行錯誤的な作業が予想される知的システムの構築過程を支援するためには、これらの手法を適時組み合わせ、開発の上流フェーズから下流フェーズまでを支援する開発支援環境を実現することは、開発効率の向上に大きく寄与すると考えられる。

筆者らはすでに、オブジェクト指向知識表現 auk (autonomic knowledge unit: 自律的知識単位) を提案し、この知識表現/推論機構をベースにした知的システム構築用シェル AUK について報告している⁷⁾。しかし、トップダウン設計/ボトムアップ構築/フェーズレス開発を統合して支援する構築環境は実現していなかった。そこで、まず、構築手法の明確化を行い、知識表現/推論機構・構築支援環境において実現すべく機能の要件を明らかにし、それらを満足する auk の拡張機能/構築支援環境の機能を考察し、開発過程を統合的に支援する開発環境を備えたツール HyperAuk を実装したので報告する。

尚、HyperAuk は現在 Smalltalk-80⁸⁾ 上で稼働し、その構築支援環境は、筆者らが開発したユーザインタフェース管理システム Talkie⁹⁾ を用いて実装している。

2. オブジェクト指向知識表現 auk

auk では、全体の知識をいくつかの部分的なモジュールに分割し、それらが協調して問題を解決していくというモデルに従って知識を表現する。この部分的なモジュールを、知識モジュールあるいは aukSet と呼ぶ (HyperAuk においてはさらに一般化し aukAccessor と呼ぶ) ことにする。これらの知識モジュールは、統一した外部インタフェースプロトコル (情報の追加 (add:), 情報の削除 (remove:), 情報の取り出し (get:), 情報の再取り出し (redo:)) のもとで情報の授受を行い、個々の知識モジュールによって規定されている推論方式に従って内部的に推論が行われていく (Fig. 1)。このプロトコルによって伝搬される情報は、スロット名/スロット値の辞書情報を保持する aukAtom と呼ばれるものである。

このように、統一した外部インタフェースプロトコルを介して情報の授受を行うことによって、モジュール内

Fig. 1 Object-oriented knowledge representation: auk

部でどのような推論が行われようと、外部的には全ての知識モジュールを同じように扱うことができ情報隠蔽がなされる。そのためにモジュール性/統合性のよい知識ベースを構築することができる。さらに、知識モジュールの推論機能は、この統一した外部インタフェースプロトコルのもとで拡張可能であり、種々の処理に対応可能である。

3. 構築手法

本構築手法では、トップダウン設計/ボトムアップ構築/フェーズレス開発を基本とし、Fig. 2 に示す手順に従って作業が進められていくものとする。本章ではまず、その手順を説明し、知識表現/推論機構・構築支援環境において実現すべく機能の要件を示す。そして、現在提案されているオブジェクト指向ソフトウェア分析/設計手法との比較を行う。

3.1 トップダウン設計

トップダウン設計は、外部インタフェースを定義することにより問題を明確にし、モジュール間の関係を規定していく過程であり、以下の手順に従う。

- 1) 知識モジュールへの分割 (Fig. 3-1)
- 2) 情報の流れの定義 (Fig. 3-2)
- 3) 外部インタフェースの定義 (Fig. 3-3)

1) は、問題を再帰的にいくつかの知識モジュールに分割する過程である。尚、全体の問題も一つの知識モジュールとして考える。2) は、部分問題として展開された知識モジュール間の情報の流れを定義していく過程である。これらの作業が Fig. 2 における Step 1 に相当する。さらに 3) は、その流れに従って知識モジュール間の制御の流れを指定し、全体を協調して動作させるための過程であり、Fig. 2 における Step 2 に相当する。具体的には、知識モジュールに制御を受け付ける外部インタフェースを定義することである。たとえば、Fig. 3-2 における a の情報の流れを外部から制御する場合には、知識モジュール A において、情報が付加 (add:) されたら何かを実行すると

いう外部インタフェースを定義しなければならない。Fig. 3-3では、外部インタフェースの定義は●で表わしている。また、たとえば、Fig. 3-2におけるbの情報の流れを知識モジュールAから制御する場合(モジュールAからモジュールBに情報を取りに行く場合)には、知識モジュールBにおいて、情報の取り出しプロトコル(get:)による外部インタフェースを定義しなければならない。

さらに、Fig. 2のStep 3は、知識モジュールの内部仕

様の概要設計を行い、実現の型を決定する処理であり、通常、制御の流れが明確にされた後に行われる。しかし、制御の流れが明確になっていない場合のボトムアップ構築の前処理としても行われる。

3.2 ボトムアップ構築

ボトムアップ構築は、内部仕様がまだ明確になっていない知識モジュール内で宣言的に知識を記述していく過程である。これはFig. 2のStep 4に相当する。記述された知識は、粒度、種類の異なるものを含み、知識モジュールの分割/統合/再構成を行いモジュール構成を明確化していく過程も含む。

3.3 フェーズレス開発

知識モジュールの構築過程は、実際にはかなりの試行錯誤によって行われる。たとえば知識モジュール設計の過程においては、全てのモジュールを設計した後にそれらの内部実現を考えるのではなく、いくつかの設計案からよりよいモジュール構成を選択したり、ある部分だけを設計し、その内部実現まで行い、部分的に動作を確認して、ほかのモジュールについて設計を行っていく方法なども考えられる。また、テストのフェーズなども、部分モジュールのみのテストから全体の動作のテストへと移行するのが望ましい。

以上の観点から、Fig. 2の各Stepは、同一レベルの知識モジュール(直接情報の授受を行う知識モジュール)間または包含関係にある知識モジュール間で繰り返されるか、あるいはStep間の矢印の流れに従って作業が進められていくものとして考える。

3.4 知識表現/推論機構、構築支援環境に対する要件

1) フェーズレス開発

3.3節で示したように、四つのStepは、モジュールの完成度によらず、任意のモジュールの任意のフェーズの作業が行える自由度が必要である。また、上流フェーズで得られる情報は、下流フェーズにおいて有効に利用されることが望ましい。

2) 外部仕様定義と内部仕様定義の分離

内部実現を進めていく過程においては、外部仕様の変更なしに内部実現形態を変更したい場合がある。そのよ

Fig. 2 Construction sequence flow

Fig. 3 Sequence of top-down planning

うな場合、自由な変更を可能とするために、外部仕様定義と内部仕様定義は分離する必要がある。このようにすることによって、外部仕様定義としては、全ての知識モジュールの記述を同一枠組み内で扱うことができる。

3) 知識モジュール包含関係の記述

モジュールはその粒度(抽象度)に応じて階層的に表現される。そのために、定められた外部インタフェースを通じて受理されたメッセージを下位モジュールに伝播する機能が必要である。また、内部実現の過程が進むに従って明らかになるモジュールの粒度の変更に対応するための包含関係を移動する機能、モジュールの分割/統合に対応する機能が必要である。

4) 整合性チェック機能

モジュールの実現過程において、その完成度によらず適時動作をチェックする機能が必要である。また、モジュール構成の変更に対するモジュール間整合性チェック機能が必要である。

3.5 オブジェクト指向分析/設計手法との比較

現在、オブジェクト指向ソフトウェア分析/設計手法としていくつかの手法が提案されている。トップダウン設計手法をベースにしたものとしては、Rumbaugh らの OMT 手法¹⁰⁾、Code/Yourdon による手法¹¹⁾などが提案されている。これらの手法は、オブジェクト間の静的な関係(OMT においてはオブジェクトモデル)と動的な関係(動的モデル、機能モデル)を定義し問題分析/設計を行っていく手法である。一方、ボトムアップ的な設計手法をベースにしたものとしては、Jacobsen による代表的な処理のシナリオに従って設計を行っていく OOSE 手法¹²⁾が提案されている。

これらの手法においては、設計者の設計作業の思考過程に追従し詳細度を高めていく手法が明確になっていない。そのために、従来からのウォーターフォールモデルを脱却することができず、試行錯誤性が重要である複雑なシステムの開発には必ずしも対応することができない。この問題を解決するためには、詳細度を高めていく試行錯誤の過程に対応するための機能を明らかにし、システム開発の上流過程のみならず、下流の過程までを一貫して支援する開発環境に必要な機能について考察する必要がある。

また、トップダウン的な設計手法は、オブジェクト指向の特徴を生かして、システムで実現しようとする対象をモジュール性良く実現できるために、拡張性/保守性に優れているが、実システムの開発においては、システムとして実現したい機能のシナリオに従ったボトムアップ的な設計手法も重要である。そこで、トップダウン手法とボトムアップ手法の融合が必要であり、そのためには、

両手法を融合していく試行錯誤の過程に対応するための機能を明らかにする必要がある。

それに対して、本構築手法では、トップダウン設計/ボトムアップ構築/フェーズレス開発を基本とし、構築手順を明らかにし、知識表現/推論機構、構築支援環境の要件を示している。

4. 知識表現/推論機構

本章では、構築手法からの要件を実現するために auk を拡張した機能について示す。

4.1 知識モジュールの一般化(aukAccessor の導入)

手続き的な記述やユーザインタフェース部分の記述も、aukSet と同一の外部インタフェースプロトコルを介して情報の授受を行う知識モジュールとして実現することにより、知的なシステムを構築する上で、全てを知識モジュールとして考え、設計作業を同一枠組み内で行うことが可能となる。また、情報源を気にすることなく、トップダウン的にモジュール設計を行うことが可能となる。HyperAuk では、このように拡張した知識モジュールを aukAccessor と呼ぶことにする。たとえば、ユーザ自身も一つの知識モジュールとして考えることにより、情報の追加、取り出しの外部インタフェースプロトコルを用いて入出力操作が行える。さらに、入出力操作の形態について考慮せずにモジュール設計が行えるため、知識ベース開発の際には単にスタップとして用いたり、外部インタフェースが合致すれば種々の入出力形態を用いることができるといった自由度が得られる。Table 1 に、現在 HyperAuk で用意している aukAccessor の型と機能を示す。この機能を実現することによって、構築手法の要件 1)、2)を実現することが可能になる。

4.2 知識モジュールの型変換機能の導入

型変換は、Table 1 に示した階層において上位から継承する機能は保存されるものとして実現する。問題をいくつかに分割する過程において生成される知識モジュールの実現の型は、生成される時点では、最も汎用的な最上位の型である aukAccessor とする。この機能を実現することによって、構築手法の要件 2)を実現することが可能になる。

4.3 conceptualAukSet の導入

定められた外部インタフェースを通じて受理されたメッセージを下位モジュールに伝播する conceptualAukSet という aukAccessor を用意した。この conceptualAukSet は、転送するメッセージの外部インタフェースとそのメッセージを転送する下位 aukAccessor を指定することにより利用できる。conceptualAukSet 内では、下位 aukAccessor 間のメッセージ授受関係

Table 1 Type hierarchy and function of aukAccessors

aukAccessor	下位の aukAccessor の抽象的な概念。 設計の段階でスタブとして用いる。
aukBag	要素として auk をもつ集まり(要素の重複を許す)で、単純な要素の追加/削除/取り出しを行う。
ruleAukBag	ルール表現を扱う aukBag の抽象的な概念。
forwardAukBag	ルールの前向き推論用の ruleAukBag。
backwardAukBag	ルールの後向き推論用の ruleAukBag。
aukSet	要素として auk をもつ集まり(要素の重複を許さない)で、単純な要素の追加/削除/取り出しを行う。
frameAukSet	フレーム表現用の aukSet。
conceptualAukSet	aukAccessor を要素とし概念の階層表現に用いる。
proceduralAuk	外部インタフェースの呼び出しによって手続きを起動する。
userAuk	ウィンドウ入出力操作を実現する。

の定義が行えるようにする。この機能を実現することによって、構築手法の要件3)を実現することが可能になる。

5. 構築支援環境

本章では、まず構築支援環境としてどのような機能を提供して構築手法からの要件を解決したかを示す。そして、そのような構築支援環境を実現するために用意したカード群の操作方法を示す。

5.1 構築手法の要件を解決するための機能

1) フェーズレス開発

視覚的なカード形式の環境を用意し、試行錯誤的に適時カードを操作できる環境を用意する。構築手法の各Stepに対しては以下のカードを用意する。

—Step 1: 仕様定義カード (Fig. 4(b))

外部インタフェースプロトコルリストカード (Fig. 4(d))

aukBag の内部実現カード (Fig. 4(a))

—Step 2: 外部インタフェース定義カード (Fig. 4(c))

—Step 3: 仕様定義カード (Fig. 4(b))

—Step 4: 各 aukAccessor 内部実現カード (Fig. 5, Fig. 6, Fig. 7)

各カードは以下のような機能をもつ。

(a) aukBag の内部実現カード

モジュール構成を視覚的に記述し、情報/制御の流れを定義するためのカードである。

(b) 仕様定義カード

知識モジュールの仕様を記述するためのカードである。また、カードを必要以上に多く開かずすむように、モジュール実現の型の指定も行い。外部仕様定義と内部仕様定義の分離という意味合いから、それぞれ別個のタイミングで指定可能とする。

Fig. 4 Outward of cards for top-down planning

(c) 外部インタフェース定義カード

ほかのモジュールからこのモジュールを利用するための外部インタフェースを定義するためのカードである。

(d) 外部インタフェースプロトコルリストカード

外部インタフェース定義をリスト表示、および生成するためのカードである。Step 1において(a)のカードと併用されるが、インタフェース定義はモジュール自身に対するもので、ここでの定義で十分であり、情報/制御の流れを指定する必要はない。しかし、(a)での記述はモジュール間の流れを意識して記述できるという面で有用であり、(a)での定義は(d)に反映される。

各カードは、開いたままで放置することが可能で、作業のフェーズに従って自由に操作することができ、各カードの情報間の整合性は維持されるものとする。このよう

なカード間の関係を維持することによって、具体的には以下のような利点が得られる。

—動作チェックの際、他モジュールで外部仕様詳細記述があればスタップとして利用できる。

—内部仕様詳細記述の際、入力を簡略化できる。

2) 外部仕様定義と内部仕様定義の分離

1)で示したように、型に関連しない外部仕様関連のカードと各型に応じた内部実現のカードを分離する。また、内部実現のカードが表示されていて、型変換を行った場合、カードの内容を更新する。

3) 知識モジュール包含関係の記述

知識モジュールの包含関係を記述するために、conceptualAukSet, aukBagの内部実現カードにグラフィカルなエディタを用意する。conceptualAukSetについては、その外部インタフェースを通じて受理されたメッセージがどの下位モジュールに伝播するかもエディタ上で定義できるようにする。また、モジュールをマウスで指定することによって、上位モジュールあるいは下位モジュールへそのモジュールを移動する機能を実現し、モジュールの分割/統合に対応する。

4) 整合性チェック機能

(a) スタップを利用した動作チェック機能

—知識モジュール単体動作チェック

外部インタフェースとして定義された aukAtom の照合パターンを送信することで、モジュール単体の動作チェックを行う。モジュール内部でほかのモジュールに対してメッセージを送信している場合は、ユーザに対して逐時報告/質問を行う。

—システム全体動作チェック

モジュール全体に対するテスト、部分的に実現されているモジュールに対するデバッグ機能としても用いられる。内部実現記述が未定義のモジュールを含む場合には、そのモジュールをスタップ化して処理が行われる。

(b) モジュール間参照パスチェック機能

モジュールの分割/統合/移動を行うとき、内部実現記述の中で用いられているほかのモジュールへの参照パス(モジュール間の階層的な位置関係を示すもので、ほかのモジュールにメッセージを送る際に利用される)の矛盾をチェックする。矛盾している場合には、警告を発する。

(c) モジュール内チェック機能

各 aukAccessor に対する内部実現記述のチェックを行う機能を用意する。たとえば ruleAukBag 内では、ルール群のチェック機能として、ルール群の連結グラフを示し、非連結グラフの検出による自動分割支援や、ルールに記述されている属性名のスペルチェック機能を用意する。

5.2 外部仕様定義

(a) aukBag の内部実現カード

ノード(Fig. 4(a)①)は知識モジュール(aukAccessor)を表わし、リンク(同図(a)②)は情報の流れを表わす。また、■(同図(a)③)は、■側に示されたモジュールの外部インタフェースが利用されることを意味している。知識モジュールは、このカードに付随するボタン[new]によって生成される。また、リンクは、一方のモジュールからもう一方のモジュールへのマウスドラッグで生成される。

(b) 仕様定義カード

aukBagの内部実現カード内のモジュールを表わすノード(Fig. 4(a)①)の上半分をクリックすることにより開くことができる。モジュール名、機能、モジュール実現の型の記述/表示を行う。モジュール実現の型を変更することができる。

(c) 外部インタフェース定義カード

インタフェース名、プロトコルの型、規定されるインタフェースの照合パターン、コメントなどを記述することができる。また、外部インタフェース仕様の記述のほかに、ここで定義されたインタフェース動作チェックは[send protocol]ボタンを用いて行うことができる。また、知識モジュールを含む aukBag の内部実現カードのリンク上の■(Fig. 4(a)③)をダブルクリックすることでも同様の操作が行える。

(d) 外部インタフェースプロトコルリストカード

リスト上で選択されているインタフェースの外部インタフェース定義カードを開くことができる。このカードは、aukBagの内部実現カード(Fig. 4(a))内の知識モジュールを表わすノード(Fig. 4(a)①)の中央部分をクリックすることにより開くことができる。

5.3 内部仕様定義

aukAccessorの内部実現記述は、その型に応じたカードによって行われる。これらのカードは、aukBagの内部実現カード内のモジュールを表わすノード(Fig. 4(a)①)の下半分をクリックすることにより開くことができる。aukBagの内部実現カードは5.2節で示した。aukSet, conceptualAukSetのカードもaukBagのものと同様である。conceptualAukSetに関しては、そのモジュールの外部インタフェースプロトコルが内部に伝播するようにそのプロトコルを表わすノードが表示される。Fig. 5(a)ではconceptualAukSetの内部実現カードを、同図(b)ではproceduralAukのカード群を、同図(c)ではaukAtom, daemonAtom(デーモン処理を行うように拡張した aukAtom)のカードを示す。

ルールによって推論を行う aukAccessor の構築過程

(a) conceptualAukSetのカード

(b) proceduralAukのカード群

(c) aukAtom/daemonAtomのカード

Fig. 5 Outward of cards for internal description

Fig. 6 Outward of cards for ruleAukBag

で用いる各種カードの外観を Fig. 6 に示す。Fig. 6(a) は、その要素を示す ruleAukBag の内部実装カードである。Fig. 6(b) は、同図(a)における [rule list] ボタンのクリックによって得られるカードで、その ruleAukBag

Fig. 7 Outward of cards for userAuk

に登録されているルール名を示している。さらに、そのカードで [open] ボタンをクリックすることによって、同図(c)のカードが開かれ、このカードを用いてルールが編集される。

userAuk の内部実装カードは、Fig. 5 の proceduralAuk の内部実装カードと同様で、ルール編集カードは、Fig. 7(a) のようである。このカードにおいて、外部インタフェースの照合パターンの指示、ルールの起動によって表示されるべきカードの設定、変数の初期状態の設定を行う。ユーザインタフェース部分は、Talkie におけるカードと同様の方法で生成される⁹⁾。同図(b)のような設定カードによって、部品と変数との束縛を行うことができる。

6. 評価

6.1 記述例

以下の二つの問題を記述し評価を行った。尚、本評価は、Sun-4/330(主記憶 24 MByte)上の Smalltalk-80 ver. 2.5 で実装された HyperAuk を用いて行った。

(1) 診断型問題(ワイン選定問題)

文献 13) で示されているシェル評価標準問題で、食事の内容に適したワインを選定し、アドバイスを行う問題である。ワイン、主食、ソースは、階層的に色/味/強さの属性が定義されている。ワインの銘柄は 9 種である。食事は、主食とそのソースが設定され、それぞれの相性度の加減によって、最高の得点を獲得したワインを選択するという問題である。

(2) 計画型問題(生産計画問題)

筆者らが開発したパッチプラントの生産計画システムにおいて、スケジュール担当者のノウハウを設定する部分に利用し、以下のような二つの問題からなる。一計画対象となるジョブ群を時間軸上に割り付ける順番

を決定するために逐次的にジョブを選択する問題

一利用する装置を選択する問題

ジョブ選択の問題については、生成物の相性、納期の比較、作業者の割付け時のノウハウなどを記述しなければならない。この例は、実際のプラントにおいて実際に立案されたデータに対して記述したもので、ジョブ数約60、装置に割り当てる操作単位が約1800程度のものである。

6.2 記述内容

(1) 診断型問題(ワイン選定問題)

ワイン/主食/ソースの属性データなどの静的な情報を管理するモジュールと推論を行うモジュールに大きく分割した。推論を行うモジュールの分割方法としては、情報の流れと共に各モジュール自身が推論の制御を行っていくパイプライン方式の分割手法と、あるモジュールがほかのモジュールにメッセージを送信することにより全体の制御を行う集中制御方式を用いた手法、両者を組み合わせた手法、それぞれについて記述した。ここでは、パイプライン方式による構成について、各モジュールの型/機能を説明する。

一属性モジュール(forwardAukBag)

推論起動メッセージを受け取ると、ユーザに主食とソースを質問し、それに対して色/強さ/味の属性の望ましいものと望ましくないものを導出する。その結果の情報を評価点モジュールに追加する。

一評価点モジュール(proceduralAuk)

属性モジュールから食事の属性の情報が追加されると、各ワインに対する評価点を計算し、評価点の高い順に逐次、銘柄選定モジュールに情報を追加する。

一銘柄選定モジュール(proceduralAuk)

評価点モジュールによって選択されたワインの候補についてユーザに確認を取り、確認が得られた時点で終了メッセージを評価点モジュールに送り、全体の推論を終了する。

一ほかのモジュール(frameAukSet)

ワイン/主食/ソースの属性データなどの静的な情報を管理するモジュールと、ユーザへの質問/確認用のモジュールを用意した。

以上ように8個のモジュール群によって構成し、内ルールで記述されたモジュールのルール数の合計は約40個である。最初の候補が得られるまでの推論は瞬時で終了した。

(2) 計画型問題(生産計画問題)

ジョブ選択のためのモジュールと装置選択のためのモジュールを共に集中制御方式で実現した。以下では、各問題についての記述内容の概要を示す。

一ジョブ選択

“品質に従って生産の順番が決定される”という条件から、生成物に対して品質のグループを決定するモジュールと、実際にジョブを選択するモジュール群を生成した。選択するモジュールとしては、各グループに対する固有の規則を記述したモジュール群と基本処理である納期順に並べるモジュールを用意し、前者のモジュールで選択できない場合に後者のモジュールで選択するように記述した処理手順を制御するモジュールも用意した。グループに対する固有の規則としては、たとえば、“同一生成物のジョブで納期の差が小さければ、そのジョブは連続して選択する”といった記述を行った。

一装置選択

ジョブ選択と同様にグループに対して異なる処理を行うために、各グループに対する固有の規則を記述したモジュール群、“利用時間あるいは利用回数の少ない装置を選択する”といった基本処理を行うモジュール、これらの処理を制御するモジュールを用意した。グループに対する固有の規則としては、たとえば、“全体の生産量および納期を考慮して並列生産(複数の同一機能装置を同時に使用する)可能台数内で装置を選択する”といった記述を行った。

全体としては、30個のモジュールによって構成し、内ルールで記述されたモジュールのルール数の合計は約70個である。処理時間は、このようなジョブ選択/装置選択に対するスケジュール担当者のノウハウを加味しなかった場合の15分に対して、約20分であったということから、推論処理には約5分要した。

6.3 パイプライン方式と集中制御方式

例題の記述を通じて、モジュール分割方式に以下の二つの方式とそれらの混合方式があること、およびそれらの手法がどのような問題の記述に適しているかがわかった。

1) パイプライン方式

情報の流れと共に各モジュール自身が推論の制御を行っていく方式である。すなわち、add: プロトコルの連鎖、あるいは、get: プロトコルの連鎖によってモジュールごとに得たい情報を渡していく方法である。この方法は、ある程度構造が明確になった小規模のモジュールには、記述が簡単で情報の流れがわかりやすく有用である。

2) 集中制御方式

ある中心的なモジュールがほかのいくつかのモジュールにメッセージを送信することにより全体の制御を行う方式である。詳細の明確になっていない大規模なモジュールのように、モジュール構成の変更の可能性があるものに適している。

Table 2 Requirements for methodology

構築方法論からの要件	知識表現/推論機構の拡張	開発支援環境の機能
1) フェーズレス開発	— aukAccessor の導入	— カード形式インタフェース — 動作チェック機能 — 内部仕様記述簡略入力機能
2) 外部仕様定義と内部仕様定義の分離	— aukAccessor の導入 — 型変換機能の導入	— 外部仕様設定カード/内部実現カードの分離
3) 知識モジュール包含関係の記述	— conceptualAukSet の導入	— 内部実現カードのグラフィックエディット機能 — conceptualAukSet/aukBag の階層移動機能
4) 整合性チェック機能		— 単体動作チェック機能 — 全体動作チェック機能 — モジュール間参照パスチェック機能 — モジュール内チェック機能

6.4 その他の評価

—モジュール構成時には、conceptualAukSetによって表現されるモジュールの階層の変更や、モジュールのカット/ペーストの機構による構成の変更は有益で、問題の整理過程としても利用できた。この機能は最終的なモジュールの整理の段階まで用いられた。

—設計されたモジュール構成に対して、内部実現を行っていく際には、モジュールのテスト機能(実現されていないモジュールのスタブ機能を含む)は、モジュールの段階的詳細化においてかなり有用であり開発効率を高めた。

—外部インタフェースの定義は、種々の状況において行えるため、設計時に行う必要がなく、必要に応じて自然に行え、開発者に負担をかけずに行えた。

—カード形式のインタフェースについては、ハイパーテキストの研究においても問題とされている現在の状況が把握しにくいという問題点を感じた。また、作業を進めるに従って、不用なカードが散乱することや、情報の分散によって、不必要に異なるカードを参照する必要が生じてしまうという問題点もあげられる。

7. おわりに

以上のように、知識表現/推論機構、構築手法、構築支援環境を一体化して考察を行い、知的なシステムを構築する際の上流過程に位置づけられる問題の整理から、下流に位置づけられるユーザインタフェース部分の開発までを支援する環境を備えた知的システム構築用シェル HyperAuk を開発した。その際、まず構築手法として、トップダウン設計/ボトムアップ構築/フェーズレス開発を基本とした構築手順を明らかにした。そして、知識表現/推論機構、構築支援環境に対する要件を整理し、オブジェクト指向知識表現 auk の拡張、構築支援環境についての考察を行い、Table 2 に示すように要件を解決し、多

様化している知的なシステムを構築するための構築支援環境の枠組みを示した。

本研究の機会を与えて下さったアプリケーション事業部福井康裕部長、システム事業部 PMK 久保哲也課長に深謝致します。また、本研究に対して、助言を頂いたオープンシステム研究所の諸氏に感謝致します。特に、故鈴木明氏には多大な御指導を頂きました。この場を借りてご冥福をお祈りします。

参考文献

- 1) G. Kahn, S. Nowlan and J. McDermott: MORE: An Intelligent Knowledge Acquisition Tool, Proc. IJCAI 85, 581/584 (1985)
- 2) 篠原, 寺野: 関係の階層化を利用した知識ベース構築支援システム, 電子通信学会技術研究報告(人工知能と知識情報処理研究会資料), AI86-32 (1986)
- 3) J. Boose: Personal Construct Theory and the Transfer of Human Expertise, Proc. of the National Conference on Artificial Intelligence, 27/33 (1984)
- 4) J. Boose and J. Bradshaw: Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge-acquisition Workbench for Knowledge-based Systems, Int. J. Man-Machine Studies, 26, 3/28 (1987)
- 5) JIPDEC: 知的情報処理システムに関する調査研究報告書, 知識システム開発方法論 (1987)
- 6) M. Eisenstadt, J. Domingue, T. Rajan and M. Enrico: Visual Knowledge Engineering, IEEE Trans., SE -16-10, 1164/1177 (1990)
- 7) 藤村, 富田, 飯間, 鈴木: オブジェクト指向知識表現 auk を用いた知的システム構築用シェル AUK, 情報処理学会論文誌, 31-1, 76/87 (1990)
- 8) A. Goldberg: Smalltalk-80: The Interactive Programming Environment, Addison-Wesley (1984)
- 9) 桑原, 藤村, 富田: MVC の拡張, MVCG とそれに基づく UIMS, Talkie の実装, コンピュータソフトウェア, 9-1, 27/41 (1992)
- 10) J. Rumbaugh: Object-Oriented Modeling and Design, Prentice-Hall (1991)
- 11) P. Code and E. Yourdon: Object-Oriented Design, Prentice-Hall (1991)
- 12) I. Jacobsen: Object-Oriented Software Engineering, Addison-Wesley (1992)
- 13) JIPDEC: 知的情報処理システムに関する調査研究報告

書, 第2分冊, 知識システム構築ツールの評価 (1989)

[著者紹介]

藤村 茂

1985年, 早稲田大学大学院理工学研究科電気工学専攻修士課程修了。同年横河電機(株)入社。現在, 高度制御技術部に所属。プラント操業支援システム構築のための要素技術の研究開発に従事。知的ユーザインタフェース, オブジェクト指向開発手法などに興味をもつ。情報処理学会, 日本ソフトウェア科学会, 電気学会, ACMなどの会員。

富田 昭司 (正会員)

1987年, 東京理科大学大学院理工学研究科電気工学専攻修士課程修了。同年横河電機(株)入社。オープンシステム研究所にてプラント運転支援。プロセスの動的挙動の認識/理解に興味をもつ。94年11月より, フィールドバス開発センター所属。電気学会の会員。

栞原 修二

1988年, 早稲田大学理工学部電気工学科卒業。同年横河電機(株)入社。93年(株)横河総合研究所へ出向。ユーザインタフェース, FDDI関連の研究開発に従事。システムプログラムへのオブジェクト指向設計の適用に興味をもつ。情報処理学会の会員。