特　集　人工知能（AI）とエネルギー・資源

エキスパートシステム構築ツールとエンジニアリング

Expert System Development Tool and Engineering

平　田　康　郎*・山　本　重　彦**
Yasuro Hirata　　Shigehiko Yamamoto

1. はじめに

ここ数年間に数多くのエキスパートシステムが作られてきた．その一部は実用化もされたが，あまり使用されずに結局はプロトタイプシステムのままに終わったシステムも多い．システム評価が甘く使用されないまま終わってしまうものも多い．

エキスパートシステムの構築は従来のソフトウェアに比較して容易であるとの当初の評判にもかかわらず知識獲得が期待されていたほど易しくないことが明らかになりつつある．しかし多くの失敗にもかかわらず，いまだエキスパートシステム構築方法の説明や、効能については未整理のままであると筆者らは考えている．

こうした中でどこに問題があり、どのような解決法が期待されるかを，具体的なエキスパートシステム構築の場を想定しながらまとめることにした．

またここでは特にプロセス用のエキスパートシステムと言うことに限定して話をまとめた．

2. エキスパートシステム構築上の諸問題

エキスパートシステムは，システム自体が成長するものであるということから，従来のコンピューターシステムと違って開発段階と運用段階が明確に区別されないと言うことがある．そのことはエキスパートシステムからの効用が，将来的にも増大するものと期待を抱かせると同時に，開発が完了した時点で実用に耐えないものであっても，必ずしもシステムが失敗していた訳ではないと弁解の余地を与えている．しかしながらこの弁解が可能であるためにはシステムが組織的にも，システム的にも実際の運用／保守を通じて開発時の不足を補って成長するものとなっていなければならない．問題は開発中よりむしろ，この運用段階に生じていると言われる．しかし同時にこの問題の根は，以下で説明するように，開発時点からあったのだと筆者らは考えている．

ここでまず多くのエキパートシステムと言われるものの構築手続き，あるいはその作成に当たっての組織等について，もう一度現状を認識してみよう．

多くのエキスパートシステム開発について常に重要視されるのがKEの役割である．確かにツールが極度に高度であり，通常の熟練者には近づきがたくかつそのような高度なツールを使わねば問題が解決しない場合は，KEと熟練者の分離は合理的である．しかしこの場合にはシステムの保守に当っては原則的には熟練者のみならず常にKEなる人が持続的に協力的に対応する必要があるだろう．何故ならばエキスパートシステムに於ては知識の追加修正は不可避であり，ここで完全にシステムが完了すると言う事がないからである．

しかし，ここで言うようなKEが常にシステムの保守に協力できるような体制は実際の会社組織運営の上で可能となっているだろうか．多くのシステムが声高にその成功を発表されていたにもかかわらず，使われないと言うことが生じている．その原因はどこにあるのだろうか．

少なくとも発表された時点でのアナウンスから言えば，それは現場の熟練者のノウハウをシステムとしては実現しているはずである．ところが，実際にはほどなく使われなくなる．何故ならほどなく知識ベースが老朽化する，あるいは期待ほどに補充されていないからである．あまり一義的に決め付けることはできないが，それにしても共通の原因があると考えるのは無駄ではないだろう．

ここでKEはその時，何をやっていたのか，あるいは果たして誰がKEであったのかを考えるのは重要である．

我々の多くの経験から言えばKEとして選ばれる人材とは通常今までSEと呼ばれていたコンピュータに

* 横河電機㈱市場開発本部アプリケーション技術部AI推進課長
** 横河電機㈱市場開発本部アプリケーション技術部長
〒163　東京都新宿区西新宿１－25－１

何らかの形で関与していたエンジニアである．それまでの多くの仕事のやり方としては開発スケジュールをまず設定して，納期までにシステムの立ち上げを完了するというのが通常である．こうした場合でも多くのシステムでシステムの保守に当っての問題が指摘されていた．それは開発終了後SEがシステムから離れてしまうため，たとえ多少の修正であってもかなりの時間待ちや余分な工数，費用が発生すると言うことである．

いまエキスパートシステムの原点を思い出してみよう．AI，あるいはエキスパートシステムへの要求はこうしたソフトウェア上の危機を解決する手段の一つとしても位置づけられていたはずである．その有用性を保つためには，熟練者がSEなどのコンピューター専門家の助けがなくともシステムの保守ができなければならない．エキスーパトからシステムが遠ざかるだけシステムはその有用性の一つを原理的にも失ってしまうのである．

エキスパートシステムは永続的に知識の追加によって完成度を高めるようなシステムであるとよく言われる．しかし実際現状の多くのシステムにおいてどのようにそのような知識の改善，追加が可能であろうか．あるいはそうした言い方が本当に意味を持つようになるような条件とはいかなるものであるかを考えてみよう．

理想的には熟練者が自分自身で知識を追加，修正することが出来るのが好ましい．ただエキスパートシステムを実際に構築している現場に赴くと多くの場合，熟練者が作るのではなく，彼等から情報を集め，原因，結果のマトリックスをSEなどスタッフ層が作成して，最終的にはスタッフ層のメンバーが作成してしまうものが圧倒的に多いと見られる．

この場合熟練者側から見ると，確かにシステムは彼の知識が活用されているはずであるが，彼にはその知識がどのように埋めこまれているかがシステムを構成するハードウェア，ソフトウェアからは見えなくなっているものが大半ではなかろうか．確かにドキュメントはあり，その過程を追いかければ彼の知識がベースになっていることは，はっきりしている．しかしそのシステムの開発，保守環境の前に立って彼は自分の知識がどう生かされているかが分からない．もちろんシステム作成者が傍にいれば，その疑問には答えてくれるであろう．ただその場合熟練者はシステム作成者を通してしかシステムそのものの改良，修正の場には近づけない．

システム作成者を通してしか熟練者がシステムを保守できないと言うことは単に人が一人余分に必要とされると言うことだけでなく，システムにとっては根本的な問題だと我々は考えている．熟練者，あるいは熟練者グループが直接システムの知識ベースを追加修正確認できることが肝要である．

実際エキスパートシステムを構築する過程にあって我々は幾度も熟練者に向けてエキスパートシステムのプロトタイプを作成し，デモしながらシステム構築を進めている．またそれが実際的なのである．その理由を我々はこう考えている．熟練者からシステムに必要な知識を出させ，知識を顕在化させるためには熟練者自身がシステムに近づき，その挙動を理解，別の言い方をすれば身に付ける必要があるのである．当り前のことだが，真の会話が成り立つためには相手の理解力を理解することが必要である．プロトタイプはシステムの理解力を熟練者に教える．それが理解されないかぎり，熟練者の知識は顕在化しない．

プロセス用のエキスパートシステムをいかに構築するかに当たって考慮すべき点は幾つかある．以下にその主なものを挙げてみる．

1. 出来上がった知識が明示的であって誰にとっても，とりわけ熟練者にとって分かるものでなくてはならない．これは知識のメインテナンスの上でも重要である．

熟練者の知識を得るためには彼等の知識がどのようにシステムの中に生かされているのかが分かるようなものでなくてはならない．

2. マンマシンインタフェイスが良く，単に診断結果だけでなくその結論に至った過程がよく分るようなものでなくてはならない．

3. いかにインタビューして熟練者のノウハウを整理するのかが重要である．

それに取り組む仕方は大きく言って二つあるだろう．一つはKEなる人物が現場に入りそこで身をもって彼らの知識を文章化することである．もう一つは熟練者自身がシステムを身に付けて，一体どのような形でシステムが動作してくるのかを知り，それに応じた知識を自分の中から抽出してシステム自身へと展開して行くことである．この二つの中間には両者の種々の協力形態があるだろう．しかしいずれにしても，システムの挙動と熟練者の挙動が高い密度で 有機的に反応し合うようなシステムでなければならないことには変わ

りはない.

3. エンジニアリングの立場から見たツール

エキスパートシステム構築支援ツールは以上のような観点から二つの特徴を持たねばならない. 一つは知識ベースへの登録が熟練者から見て明示的であり, 分かりやすいということである. もう一つはその知識の周辺部ではツールとして徹底的にフレキシブルであり, 場合によりプログラマブルである, 別の言い方をすれば従来言語とあらゆる層で結合できると言うことである.

前者の理由は説明は不要であろう. 後者の特徴の意味するところは. できあがるシステムが周辺部と種々の形で結合し合うような複雑なシステムを作成できるから便利だと言うことではない. そうではなくシステムの構造をできる限り簡単にしようとする意図に対してツールが逆らわずに調和できるようなものでなければならないと言うことである. その点をもう少し分かりやすく述べてみよう.

従来言語との結合性が重要だと言うことは既に種々の文献で強調されていることである. 実際, 従来言語と結合することができないツールなどはプロセス用のものである限りないであろう. しかし結合のレベルや, 結合の容易さは個々に違っており種々の制限があるのが普通である.

応用面で考えるならエキスパートシステムはどのシステムでも同じような結合方式で実現できる訳ではない. 特定のエキスパートシステムで便利であった結合方式は他のシステムではそのまま使えないことがある. どのようなレベルでの結合があるのかを考えると, a. 推論プロセス前後処理部分, b. 推論グループ出入り部分, c. 推論個別ルール前後, d. 推論個別ルール内部からの結合などがある.

これらの結合は具体的な個別のアプリケーションですべてが使われる訳ではない. しかし推論個別ルール内部からの直接結合を使えばシステムの機能が明瞭に分割できるのに, それができない場合には, 推論プロセス前後処理部分での結合で代用せねばならず, 本来一つの知識であるべきものが分断され, 一見では分からなくなる. そうして出来上がるシステムは熟練者から見て理解しがたいものとなる場合が多くなる. それは熟練者による知識修正／追加を不可能にし, 結果的にシステムは早かれ遅かれ使われなくなるだろう.

以上のような派生する種々の問題点を念頭に置きながら一例として横河のエキスパートシステム構築ツールを取り上げ, そのエンジニアリングなどについて述べることにする.

4. エキスパートシステム構築用ツールXLAI

XLAIは当社CENTUM　XLシリーズのAI機能を実現するユニットであるAIWS上に搭載されているエキスパートシステム開発支援ツールである. その内容については文献 (1) に概略の紹介があるので, ここではエンジニアリング上重要と思われる点についてのみまとめることとする.

XLAの知識ベースを理解する上で重要な用語は4つある. 以下に挙げる.

1. 推論グループ
2. 属性クラス・フレーム／属性フレーム
3. ルール・クラスフレーム／ルールフレーム
4. コントロールフレーム

推論グループは推論を幾つかの異った種類のグループに分類する時に使用する. また属性, ルールともにクラス化の概念が実現できるようになっており, 実行に当たってもクラスのレベルで実行順序等の制御が可能である.

属性とはプログラムで考えると変数に相当するものである. つまり知識の対象になるもの, 事象や仮説など属性を持つものの総称である. CENTUMのタグ名などもこの一つである. やはり, クラス化できるので, アプリケーションの知識のまとまりに応じて, まとめることが可能である.

```
属性フレーム　AS
＊クラス名　Tlll
＊値タイプ　text
＊個数指定　single　choice
指定値　"HI"　　属性定義
　　"NR"　　(事象部分)
　　"LO"
＊決定優先度　RU
エンドフレーム
```

この例ではタグTlllのAS (アラームステータス)の属性フレームの定義を行なっている. ここでは値のタイプ, 個数指定, 指定値を定義している. 決定優先度はこの属性フレームの値を決定する時の優先度を定義している. RUと言うのは, R (ルール) を最優先し,

U（ユーザー）をその次にしますという意味である．つまり，システムはこの属性フレームの値を決定するルールが知識ベース上にない場合は，CRT上に質問メッセージが出て，オペレータがキーボードから答えることになる．

ルールとは属性の関係を記述するものである．基本的にはIF～THENの構造を持つものである．やはりこの場合も，クラス化によってルールのグループ化が可能である．

```
ルールフレーム　ルール１－８
＊クラス名　１号反応器ルール
＊条件部　PV　of　F113<100.0
＊結論部　診断結果of　１号反応器＝”スチーム
　　供給量制御不良” [0.4]
＊実行部　（条件部が満足された時に実行する関
　　数／プログラムをここに記述できる）
エンドフレーム
```

この場合はルール１－８は１号反応器ルールクラスの一つであることを示し，PVof F113が100.0以下であると言う条件が成立すれば，診断結果of　１号反応器に”スチーム供給量制御不良”と言う内容を確信度[0.4] で与える．

結論部は推論した結果を確信度で与える部分である．また条件が満たされた時に行ないたい処理があれば，それは実行部で宣言できる．

このルールフレームの一つの特徴は実行部である．実際のシステムを作成する段階で，こうした機能は是非とも必要となるものである．この機能がなければ，ここでは何か別の変数（属性値）を無理やり作り，それを推論処理の後で参考にして別のプログラムを起動するなどということをしなければならない．こうしたやりかたはこれまでの多くのシステム設計者やプログラマーが経験したことである．しかしやっている内容が複数に分解され，ルールだけを見ていのでは，システムの挙動が分からなくなるのはもちろんである．こうした事態を避けるためにも，この実行部の機能は大変重要である．

重要な点は条件部と結合部が明確に区別されていて，かつ条件／結論の知識と熟練者が日常使用する知識との間に落差がないことである．このテキストを見て，熟練者がその意味を理解できることであり，自ら修正するときにも誤りを冒さないものでなければならない．

コントロールフレームと言うのは，推論の実行制御をする部分である．例えばcheckは指定ルールクラスにあるすべてのルールについて条件部が成立したか否かを調べる．またこのフレームからもC，FORTARANで記述したアプリケーションプログラムを自由に呼び出せる．

```
コントロールフレーム
＊呼び出し方　top. level
＊手続き　execute　”data—get ()”
　if（ASofT113　is　”LO”）
　then　{check　１号反応器ルール
　　display　（診断結果of１号反応器）
　　display　（対応処置of１号反応器）
　・・・
エンドフレーム
```

5. XLAIでのエンジニアリングの実例

ここでは実際にこのツールを使用してどのようにエキスパートシステムを作成するのかを例題を使用して説明する．この例はボイラー異常診断システムでの一例である．

次の図はあるボイラーシステムでの異常原因と，その時の現象をマトリックス状にまとめたものの一部である．

このような形で知識がまとまれば，エキスパートシステムに投入する知識は，このマトリックスに対応する形で，実現できる．その一部に注目してみよう．

この表１の矢印の部分の知識は以下のようなものとして実現できる．

```
ルールフレーム　高
＊クラス名　主蒸気流量
＊条件部　主蒸気流量　is　高
＊結論部　結果of水・蒸気流＝”SHバンク部破
　　壊の可能性あり” [−0.1]”主蒸気弁
　　閉で動作不良の可能性あり” [−0.4]
　結果of風・煙道系統＝”FDF故障の可能性あ
　　り” [−0.1]，
　　”FDF出口ダンパ全開で動作不
　　良の可能性あり” [−0.1]
　　”FDF出口ダンパ全閉で動作不
　　良の可能性あり” [−0.1]
```

表1

		A 給水出口側破壊	B 給水ポンプ故障	C SHバンク部破壊	D 給水弁全開で動作不良	E 給水弁全閉で動作不良	F ドラムレベル発信器異常	G 主蒸気遮断弁全閉で動作不良	H 燃焼不足	I 燃焼過多	J SHスプレー弁全開で動作不良	K SHスプレー弁全閉で動作不良	L SHスプレー弁出口破壊	M 異常なし	N FDF故障	O FDF出口ダンパ全開で動作不良	P FDF出口ダンパ全閉で動作不良	Q IDF故障	R IDF出口ダンパ全開で動作不良	S IDF出口ダンパ全閉で動作不良	T 燃料流調弁全開で動作不良	U 燃料流調弁全閉で動作不良
1	ドラムレベル 低	+0.2	+0.2	+0.4	−0.2	+0.2		−0.2														
2	ドラムレベル 高	−0.2	−0.2	−0.4	+0.2	−0.2		+0.2														
3	主蒸気温度 低		−0.1	+0.2				−0.2	+0.4	−0.4	+0.2	−0.2	−0.2		+0.1	+0.1	+0.1	+0.1	+0.1	+0.1	+0.4	−0.4
4	主蒸気温度 高		+0.2	−0.2				+0.2	−0.4	+0.4	−0.2	+0.2	+0.2		−0.1	−0.1	−0.1	−0.1	−0.1	−0.1	−0.4	+0.4
5	主蒸気圧力 低	+0.2	+0.2	+0.2				−0.1	+0.1	−0.1					+0.1	+0.1	+0.1	+0.1	+0.1	+0.1	+0.1	−0.1
6	主蒸気圧力 高	−0.2	−0.2	−0.2				+0.1	−0.1	+0.1					−0.1	−0.1	−0.1	−0.1	−0.1	−0.1	−0.1	+0.1
7	主蒸気流量 低			+0.1				+0.4	+0.1	−0.1					+0.1	+0.1	+0.1	+0.1	+0.1	+0.1	+0.1	−0.1
8	主蒸気流量 高			−0.1				−0.4	−0.1	+0.1					−0.1	−0.1	−0.1	−0.1	−0.1	−0.1	−0.1	+0.1
9	給水流量 低	+0.4	+0.4		−0.4	+0.4		+0.1														
10	給水流量 高	−0.4	−0.4		+0.4	−0.4		−0.1														
11	燃料流量 低								+0.6	−0.6											−0.4	+0.4
12	燃料流量 高								−0.6	+0.6											+0.4	−0.4
13	(給水流量−主蒸気流量)偏差大	+0.1	+0.1	+0.1	+0.1	+0.1		+0.1							+0.1	+0.1	+0.1	+0.1	+0.1	+0.1		
14	異常なし													+0.9								
15	炉内圧 高														−0.2	+0.4	−0.4	+0.2	−0.4	+0.4		

表2　データ収集定義シート1

系列 SAMPLE−101　　No. 1　　Date. '88.11.2

タイプ	タグ名	データタイプ PV	MV	SV	AS	LS	CHK	AS1	PV1	PVA	MAX	変換タグ タグ名	データタイプ	個数	式	推論グループ グループ名			TANK		BOILER							備考
																周期	10		60. 1		10 AS(F103)=M							
																SEC	IN	SEND	IN	SEND	IN	SEND	IN	SEND	IN	SEND	IN	SEND	
F	F101	○										F101	PV1	2														
F	F102	○										F102	PV1	30														
F	F102				○														○	○								
F	F103				○																							
F	※SWOLOSOW	○																			○	○						
F	※CTOIOSOL	○																			○	○						
K	Z101	○										$P_{VA(F_{101})}$		1														
V	F101								○			$P_{VA(F_{101})}$		1					○										
V	F101									○										○								
V	F102								○			$MAX(F_{101})$		2														
V	F102										○						○			○								
V	Z102	○												11						㊙								※ TANKでは PV(Z102)(10)
V	F103							○													K							
F	F102	○										$PV_{(Z_{102})(0)}$		3														

"IDF故障の可能性あり"
[−0.1]
"IDF出口ダンパ全開で動作不良可能性あり" [−0.1]
"IDF出口ダンパ全閉で動作不良の可能性あり" [−0.1]
結果of燃料系統＝"燃料流調弁全開で動作不良の可能性あり" [0.1]
＊実行部 execute fig—char (10, 400, "主流蒸気流量高い")

このように現象原因マトリックスからの知識と実際に知識ベースに投入されている知識との間には何の落差もない．マトリックスができていれば，そこから知識どおりに動くシステムを作成するには一日，あるいは二日と要さないのである．またこの例のように実行部での必要な実行指示をすれば，ルール条件が満たされれば，その単位で，必要な処理を実行させることができる．

さらにもう一つの重要なことがある．それはプロセス用エキスパートシステムである以上，いかにプロセスデータを収集し，推論システムにそれを渡すかである．この部分が複雑になり，場合より知識ベースそのものがその複雑さに巻き込まれてしまう．

次の頁はデータ収集定義ファイルの一例である．ここではデータ収集用のワークシートの一例を使用して説明する．表2

左例にタグ名が記述でき，次に変換したデータ名，どう言う周期で，どのような加工がされて推論側に渡されるかが一覧表で分かるようになっている．

このワークシートが完成されれば，実際の定義ファイルは機械的に作成できる．このように熟練者から見てエキスパートシステムが違和感なく実現されている．

6. エキスパートシステムの将来と展望

今回は主に診断系のエキスパートシステムのエンジニアリングの問題点を記述した．比較的方法の見え始めている診断系に比較すると，計画処理問題は，実現方法自体でも難しいところがある．また診断系にも学習的機能を何とか実現していきたい，かつ実効力あるものとして機能を付加したいとの要求がある．

新しい手法を開発すると，同時にそれがエキスパートには耐えがたい難解さになり，ツールとしてのわかり易さを失う．実現方法は一方でソフト上難解なものにならざるを得ない点もある．一方でアプリケーションソフトが難解では意味がないということがある．その意味で，この分野は一歩前進二歩後退だと感じない日はない．

しかし，振り返って半年とか，一年のオーダーで見ると，確実にツールもエンジニアリング手法も進歩しているのである．

参 考 文 献

(1) 小西清一 XLAI;オンラインエキスパートシステム構築ツール横河技法 Vol. 32 No.4 (1988)