ANALIZA ŞI EVALUAREA PRINCIPALELOR NUCLEE ŞI MEDII DE PROGRAMARE EXISTENTE PENTRU ELABORAREA DE SISTEME EXPERT

Ing. Badea Liviu

Institutul de Cercetări în Informatică

Rezumat

Creşterea continuă a numărului de sisteme expert, scrise în ultimii ani, precum şi gradul mare de complexitate al acestora a condus la proiectarea şi scrierea unor nuclee şi medii de programare ("shell"-uri), care să faciliteze elaborarea sistemelor expert. Acest pas înainte în ingineria software poate fi comparat cu dezvoltarea primelor limbaje de nivel înalt, menite să înlocuiască programele scrise direct în limbaj de asamblare. Practic, s-a constatat că folosirea nucleelor pentru elaborarea de sisteme expert (ESBT = "Expert System Building Tools") reduce timpul de scriere al unui sistem expert, cu un ordin de mărime (faţă de scrierea lui într-un limbaj convenţional, cum ar fi LISP sau C++).
Lucrarea conţine două părţi. Prima parte analizează structura generală a unui nucleu ("shell") pentru elaborarea de sisteme expert, facilităţile oferite inginerului de cunoştinţe şi principalele aplicaţii realizabile cu un astfel de sistem.
A doua parte a lucrării prezintă pe scurt principalele facilităţi ale celor mai răspîndite nuclee comerciale . Fiecare sistem este analizat din punctul de vedere al principalelor criterii de comparaţie adoptate (reprezentarea cunoştinţelor, motorul de inferenţă, interfeţe cu sistemul şi programatorul, cost, performanţe, utilizări funcţionale etc).
Lucrarea este însoţită de două tabele sintetice, care surprind importanţa diferitelor atribute ale nucleelor de sisteme expert pentru o aplicaţie dată şi. respectiv, principalele atribute ale sistemelor expert comerciale.

1. Structura unui nucleu pentru elaborarea de sisteme expert (ESBT)

Partea centrală a unui sistem expert este constituită dintr-o bază de cunoştinţe şi un motor de inferenţă, care operează asupra acesteia pentru construirea unei soluţii a problemei. Pentru comunicarea cu mediul exterior sistemului, mai sînt necesare interfeţe cu lumea exterioară. De asemenea, pentru facilitarea dezvoltării sistemului expert, un ESBT mai trebuie să includă o interfaţă cu programatorul, care are ca scop:

- construirea bazei de cunoştinţe pentru domeniul particular al problemei;
- dezvoltarea unei interfeţe cît mai naturale cu utilizatorul;
- posibilitatea controlului procesului de inferenţă.

1.1. Reprezentarea cunoştinţelor

Principalele aspecte ale reprezentării cunoştinţelor, luate în considerare de nucleele comerciale existente, sînt următoarele:

- reprezentarea cunoştinţelor prin:
  - "frames" (cu sau fără moştenire);
  - obiecte;
  - predicate, logică;
  - reguli;
  - lumi multiple;
- factori de certitudine;
- acţiuni:
  - reguli (grupate sau nu pe clase de reguli);
  - exemple (folosite în sistemele inductive pentru facilitarea achiziţiei de cunoştinţe);
  - logica (o formă particulară de reguli);
  - mesaje sau demoni (în programarea orientată pe obiecte);
  - proceduri.

Majoritatea nucleelor de sisteme expert existente oferă posibilitatea folosirii mai multor paradigme de programare (programare orientată pe obiecte, procedurală, reguli, programare logică etc), pentru a putea rezolva cît mai natural aplicaţia specifică.
Un punct deosebit de important în reprezentarea cunoştinţelor este dacă se foloseşte sau nu un model cauzal sau structural ("deep model") al problemei. Un astfel de model poate mări performanţele sistemului expert din punct de vedere a corectitudinii expertizei, însă cu preţul scăderii performanţelor în ceea ce priveşte timpul de calcul şi memoria ocupată.

1.2. Motorul de inferenţă ("Inference Engine")

Motorul de inferenţă este o maşină virtuală, care funcţionează conform unui model abstract de rezolvare a problemelor. Majoritatea ESBT pot folosi mai multe modele, respectiv tehnici diferite de rezolvare a problemelor, ca de exemplu:

- "backward chaining" (înlănţuire înapoi, dirijată de scopuri);
- "forward chaining" (înlănţuire înainte, dirijată de fapte);
- raţionare bazată pe ipoteze (non-monotonică):
  - "viewpoints", contexte, lumi;
  - "truth maintenance" (TMS) şi backtracking dirijat de dependenţe;
  - raţionament de tip ipoteză-test;
- rezolvarea problemelor folosind programarea orientată pe obiecte (bazată pe transmiterea de mesaje între obiecte);
- arhitectura bazată pe "tabla" (blackboard), care permite cooperarea mai multor subsisteme în rezolvarea unei probleme, comunicarea între ele făcîndu-se printr-o structură de date comună numită tablă; controlul construcţiei

soluției pe tablă se realizează printr-un mecanism de agendă;
- programare logică, bazată pe un demonstrator automat de teoreme; un rol deosebit îl joacă aici algoritmul de "unificare" a variabilelor, care este determinant pentru puterea de reprezentare a sistemului;
- generarea de reguli și/sau arbori de decizie în mod inductiv, din exemple; tehnicile de învățare inductivă facilitează achiziția cunoștințelor de la experții umani, care în general nu sînt capabili să-și exprime cunoștințele sub forma unor reguli generale, ci mai degrabă sub formă de exemple;
- paradigma rezolvării problemelor folosind "valori active" (demoni) este potrivită aplicațiilor de monitorizare în timp real;
- meta-controlul se referă la alegerea și controlarea strategiei de inferență și are ca scop evitarea exploziei combinatoriale a timpului de căutare a soluției.

O caracteristică importantă a motorului de inferență folosit este și raționarea în situații de incertitudine (care apar în majoritatea problemelor "reale").
Ținînd seama de faptul că, în rezolvarea unei probleme se folosesc de obicei mai multe strategii de inferență diferite, este deosebit de importantă posibilitatea legării soluțiilor subproblemelor obținute, de motoare de inferență diferite. Acest lucru depinde în mare măsură de uniformitatea reprezentării interne, folosită de ESBT (de exemplu "frames" sau obiecte).
Interfața oferită programatorului ("developer") trebuie să includă: facilități pentru crearea și actualizarea bazei de date (procesor de texte, meniuri, interfețe grafice, "help", posibilitatea compilării incrementale a bazei de cunoștințe și extensibilitatea acesteia), facilități de depanare (editor grafic pentru baza de date, verificarea consistenței acesteia, posibilități de trasare, justificare <explicații> etc.), facilități de control al procesului de inferență și de creare a interfeței sistemului cu utilizatorul. Aceasta din urmă trebuie să permită dialogul interactiv cu utilizatorul, într-un limbaj cît mai apropiat de cel natural (ținînd seama de faptul că utilizatorul nu are de obicei cunoștințe de programare), trebuie să poată accepta răspunsuri incerte și, eventual, să poată explica modul de obținere al unor concluzii intermediare și/sau motivul din care cere o anumită informație de la utilizator. Aceste facilități măresc încrederea utilizatorului în sistem și-i permit acestuia să influențeze strategia de inferență, folosită de sistemul expert.
Facilitățile grafice avansate (cum sînt imaginile active, posibilitățile de simulare și de afișare grafică a bazei de cunoștințe) au un rol deosebit în succesul sau insuccesul unui sistem expert. O facilitate deosebită este posibilitatea explorării alternativelor ("what-if queries"), ceea ce permite utilizatorului eliminarea unor alternative pe care acesta le consideră nerelevante.

Limbajul în care este scris sistemul expert are o importanță deosebită, deoarece el determină posibilitatea compilării (eventual incrementale) sistemului, ținînd seama de faptul că, un sistem compilat are o viteză de execuție mai ridicată (împreună cu cerințe de memorie mai scăzută) în vreme ce, posibilitatea compilării incrementale duce la creșterea vitezei de dezvoltare a sistemului.
Limbajele cele mai folosite sînt: LISP-ul (pentru flexibilitate) și chiar C-ul (pentru viteză). Alte limbaje folosite sînt PROLOG, SMALLTALK, PASCAL etc.
De asemenea, sînt importante resursele hardware, necesare pentru dezvoltarea sistemului (de obicei mașini Lisp: Symbolics, LMI, Xerox, TI Explorer sau stații de lucru: DEC MicroVAX, Sun Station, Apollo, Tektronix, Masscomp, IBM-RT), dar și pentru rularea sistemului ("target machine" = mașina țintă), care sînt, de obicei, sisteme mai ieftine (de exemplu: calculatoare personale IBM PC/AT, TI Prof, ATT, Macintosh).

1.3. Tipuri de aplicații realizabile cu ESBT

Deși există nuclee de sisteme expert, care își propun să poată construi sisteme expert în aproape toate domeniile cunoscute (sisteme "universale"), majoritatea nucleelor de sisteme expert se pretează unei anumite clase de aplicații, cum ar fi de exemplu:

1. Probleme de clasificare:
   a. interpretarea măsurătorilor (selecția unei ipoteze bazîndu-se pe date obținute din măsurători);
   b. diagnoza (sistemul, nu numai că interpretează datele măsurate, ci și cere date suplimentare, care să-l ajute în raționament);
   c. depanare, tratament sau reparare (întreprinderea unor acțiuni sau recomandarea unor măsuri pentru corectarea unei situații adverse, care a fost diagnosticată);
   d. "use advisor" (ghid de folosire): ghidarea unui utilizator neexperimentat în folosirea unui sistem complex (de exemplu: repararea unei mașini, pilotarea unei aeronave).
2. Proiectare și sinteză (probleme constructive):astfel de probleme construiesc o soluție pe baza unor restricții date;
3. Predicție (bazată fie pe experiența anterioară, fie pe modele cauzale ale sistemului);
4. Asistent inteligent (ajută utilizatorul în luarea deciziilor, poate furniza informații sau efectua subtask-uri);
5. "Scheduling" (ordonarea temporală a unui set dat de taskuri sau acțiuni, ținînd seama de restricții temporale și de resursele existente);
6. Planificare (alegerea unui set de acțiuni, care să satisfacă un scop dat); planificarea este mai complexă decît "Scheduling", pentru că presupune alegerea acțiunilor;

7. Monitorizare (observarea evoluției unui sistem și alertarea utilizatorului în cazul devierii de la normal);de exemplu, în cazul zborurilor spațiale, proceselor industriale, condițiilor clinice ale pacienților, acțiunilor inamicilor în simulări militare;
8. Control (combinație între monitorizarea unui sistem și acțiunea efectuată pentru atingerea unor scopuri); acesta este cazul sistemelor cu răspuns în timp real;
9. Prelucrarea informațiilor (de exemplu: construcția inductivă a unui arbore de decizie din exemple);
10. Descoperire: astfel de sisteme pun accentul pe găsirea de noi relații sau concepte într-un anumit domeniu (de exemplu, în matematică sau fizică);
11. Raționare bazată pe exemple;
12. Învățare.

Aplicațiile enumerate mai sus nu sînt neapărat disjuncte, în sensul că, două asemenea aplicații pot prezenta părți comune sau una dintre ele ar putea fi descompusă în altele mai simple.

În tabelul 1 (vezi [1]) este evaluată importanța diferitelor atribute ale ESBT, pentru diferite aplicații posibile. Tabelul este oarecum subiectiv deoarece, dependent de ingeniozitatea programatorului, unele aplicații, deși n-ar putea fi realizate direct cu un ESBT dat, ar putea fi totuși descompuse în subprobleme rezolvabile cu ESBT-ul dat (deși rezolvarea nu ar fi nici elegantă, nici eficientă).

2. Descrierea principalelor nuclee și medii de programare pentru construirea de sisteme expert

2.1. ART (Automated Reasoning Tool)

ART, produs de Inference Corporation (5300 W. Century Blvd. Los Angeles, CA 90045 (213) 417-7997) este un sistem relațional bazat pe reguli și dirijat de date, avînd încorporate "frames", programare orientată pe obiecte, TMS ("Truth Maintenance") și lumi multiple ("multiples viewpoints"). S-a pus un accent deosebit pe performanțele sistemului, care să-i permită lucrul în timp real. Arhitectura sistemului include:

1. baza de cunoștințe (compusă din fapte ("facts"), reguli, "viewpoints" (lumi, contexte), scopuri, scheme ("frames"), metode, valori active (demoni);
2. editoarele grafice ale bazei de cunoștințe;
3. compilatorul bazei de cunoștințe (care transformă faptele și regulile în cod procedural și baze de date indexate);
4. motorul de inferență (care realizează procesul de raționare);
5. monitorul execuției (folosit pentru trasarea, depanarea și colectarea statisticilor pentru evaluarea performanțelor);
6. ARTIST (ART Interface Synthesis Tool), care permite dezvoltarea interactivă a interfețelor grafice, ale aplicațiilor.

2.1.1. Reprezentarea cunoștințelor în ART

ART este un sistem bazat pe reguli și dirijat de date, în care faptele și scopurile sînt reprezentate relațional. Totuși, este permisă și chiar încurajată reprezentarea cunoștințelor, sub forma schemelor (frames), care sînt compilate în tuple relaționale. Atributele (sloturile) pot avea valori multiple, acestea fiind traduse în fapte multiple. În terminologia ART, relațiile sînt legături care stabilesc o ierharhie de scheme și care permit moștenirea atributelor. Deși există niște relații de moștenire predefinite, este posibilă definirea unora noi (deoarece relațiile de moștenire sînt și ele scheme accesibile utilizatorului). Unele dintre schemele de relație predefinite sînt următoarele:

Relații de moștenire:
- instance-of
- is-a

Relații între mulțimi:
- subset-of
- element-of
- prototype-of

Relații inverse:
- kinds — este inversa lui is-a;
- has-instances — este inversa lui instance-of;
- has-elements — este inversa lui element-of;
- has-subsets — este inversa lui subset-of;
- prototype — este inversa lui prototype-of;
- inverse — este inversa lui inverse.

Dintre atributele relațiilor predefinite, cele mai importante sînt reflexive, symmetric, anti-symmetric, transitive, inverse, closure-of.

ART permite moștenire multiplă (care poate fi tratată, printre altele și prin raționament non-monotonic, bazat pe ipoteze).

2.1.2 Inferență și control în ART

ART este un sistem care încurajează programarea prin reguli, evitîndu-se, pe cît posibil, programarea procedurală. Totuși, aceasta, ca și programarea orientată pe obiecte, sînt posibile în ART, scrierea metodelor făcîndu-se în LISP sau C.

Mecanismul de inferență, bazat pe reguli folosește în principal înlănțuirea înainte ("forward chaining"), însă există și mecanismul necesar pentru înlănțuirea înapoi ("backward chaining"). De obicei, înlănțuirea înapoi se folosește în următoarea situație: cînd ART încearcă aplicarea unei reguli forward, verificînd o precondiție a acesteia prin compararea (pattern-matching) cu baza de date, s-ar putea ca această condiție să reprezinte concluzia unei reguli backward. În acest caz, respectiva

condiţie devine un scop (inserat în baza de date) pe care ART va încerca să-l demonstreze folosind înlănţuirea înapoi. Deci, înlănţuirea înapoi are rolul de a micşora numărul de fapte necesare în memoria de lucru.
ART s-a inspirat iniţial din OPS5 în ceea ce priveşte regulile forward. Sintaxa unei astfel de reguli este următoarea:

(defrule <nume-regulă-forward>
<condiţii>
⇒
<acţiuni>)

Sintaxa unei reguli backward diferă puţin de cea a unei reguli forward, existînd chiar două formate diferite:

(defrule<nume-regula-backward>
goal (X), <condiţii>
⇒
<acţiuni>)
(defrule <nume- regulă-backward>
goal (X)
⇐
<condiţii>)

Deşi cel de-al doilea format ("backward format") seamănă cu cel folosit în PROLOG, aplicarea regulilor nu se face în adîncime, ci poate fi controlată printr-un mecanism de agendă.
Pattern-matching-ul este foarte complex şi puternic permiţînd folosirea de variabile (notate ?X) sau chiar liste de variabile (notate $?X), precum şi conectivi logici.
Mecanismul de control include priorităţi ataşate şi un mecanism de agendă, folosit pentru selectarea regulilor (care însă nu poate fi controlat explicit de acestea, fiind invizibil în exterior).

2.1.3. TMS (Truth Maintenance) şi raţionare non-monotonică

Există două tipuri diferite de TMS în ART. Primul exprimă dependenţele logice dintr-un singur context (viewpoint), retrăgînd concluzii cînd premisele acestora sînt retrase. Al doilea este un ATMS ("Assumption based TMS" = TMS bazat pe asumpţii) care elimină contexte, cînd se retrag ipotezele (asumpţiile) acestora.
Raţionamentul de tip non-monotonic este realizat prin contexte (viewpoints) care pot fi create şi eliminate de reguli. ART construieşte un arbore de contexte ("branching tree of viewpoints"), care poate fi privit şi ca un arbore de lumi multiple.
În afara mecanismului de contexte, ART nu suportă baze de cunoştinţe multiple. De asemenea, regulile nu sînt grupate pe clase de reguli.

2.1.4. Alte facilităţi

O versiune apropiată de limbajul natural al regulilor poate fi folosită pentru explicarea (justificarea) liniei de raţionament.
ART nu conţine un mecanism specializat pentru raţionarea în situaţii de incertitudine, însă acesta poate fi realizat de utilizator. Sistemul de dezvoltare este foarte puternic, flexibil şi bine integrat. ART este interfaţat cu Common LISP şi C şi, prin acestea, cu multe alte limbaje. Deşi este implementat în LISP, ART este foarte eficient, din punct de vedere al vitezei fiind lipsit de "garbage-collection". Recent, ART a fost rescris în C (versiunea C se numeşte ART-IM = "ART Information Management").
Spre deosebire de versiunea Lisp, ART-IM nu suportă moştenire multiplă, căi de moştenire specificate de utilizator, "viewpoints" şi "backward chaining".
Principalele clase de aplicaţii, realizabile cu ART sînt: planificare/ "Scheduling", simulare, generarea de configuraţii, proiectare. Sistemul rulează pe maşini puternice: Symbolics, LMI, TI Explorer, Sun-3, Dec VAX.
Versiunea C (ART-IM) rulează şi pe IBM PC/AT, IBM PS/2, DEC VAX.

2.2. KEE (Knowledge Engineering Environment)

KEE este produs de IntelliCorp (1975 El Camino Real West, Mountain View, CA 94040-2216; (415) 965-5500) şi reprezintă cel mai folosit mediu de programare, pentru construcţia de sisteme expert sofisticate. Este caracterizat prin marea sa complexitate, prin multiplele facilităţi oferite programatorului şi prin faptul că rulează pe calculatoare performante (în ultima vreme a apărut o versiune şi pentru maşini cu Intel 386, sub Unix).
Inspirat de sistemul "Units" de la Stanford, KEE are o arhitectură hibridă, care este accesibilă, extensibilă şi chiar modificabilă, integrarea diferitelor componente fiind foarte bine făcută. Arhitectura sistemului conţine:

1. un limbaj de reprezentare, care include:
   - obiecte/unităţi (frames);
   - organizare ierarhică;
   - moştenire multiplă;
   - verificarea tipurilor;
   - lumi multiple;
   - ATMS ("Assumption Based Truth Maintenance System");
2. instrumente de inferenţă şi analiză:
   - sistemul de reguli de producţie, care include:
     - înlănţuirea înainte şi înapoi;
     - backtracking automat;
     - căutare bazată pe un mecanism cu agendă, controlabil de utilizator;
     - raţionare monotonică şi non-monotonică;
     - compilator de reguli;
     - limbaj logic de tip TellandAsk;
     - programare orientată pe obiecte (mesaje);

- programare dirijată, de date (demoni/valori active);
- programare dirijată de acces.

3. instrumente de interfaţare:
   - interfaţa programatorului pentru dezvoltare, depanare şi explicaţii;
   - KEE pictures (pentru construirea de interfeţe grafice cu utilizatorul);
   - "ActiveImages" (bibliotecă de obiecte grafice active);
   - "CommonWindows" (pentru probabilitate).

2.2.1. Reprezentarea cunoştinţelor în KEE

Obiectele sînt reprezentate în KEE prin unităţi ("units") care pot fi organizate în ierarhii. Pot exista, simultan, ierarhii multiple (care se pot afla în interacţiune) şi, de asemenea, sînt suportate bazele de cunoştinţe multiple. KEE permite adăugarea de sloturi la o unitate, putîndu-se specifica restricţii (atribute) în ceea ce priveşte domeniul de valori şi cardinalitatea. Sloturile pot avea valori multiple. Restricţiile asupra sloturilor sînt locale contextului, astfel că un slot poate avea, de exemplu, valori multiple într-un context şi o unică valoare în altul. Valorile sloturilor pot fi expresii LISP, referinţe spre alte unităţi sau chiar metode. De asemenea, se pot ataşa demoni la sloturi (care sînt şi ei moşteniţi).
Reprezentarea unităţilor în KEE este următoarea:
(nume.unit
(creator data.creare modificator data.modificare)
- lista.părinţilor.superclase;
- lista.părinţilor.member.of;
- comentariu;
- lista."member".sloturilor;
- lista."own".sloturilor);

Reprezentarea sloturilor este:
(nume.slot
- valori.locale;
- rol.moştenire;
- clasa.de.valori;
- lista.valorilor.implicite;
- lista faţetelor)

"Member-slot"-urile sînt moştenite de toate subclasele şi de toţi membrii clasei curente. "Own-slot"-urile sînt locale clasei curente şi nu sînt moştenite de descendenţi. Subclasele moştenesc "member-slot"-urile ca "member-slot"-uri, în vreme ce, instanţele moştenesc "member-slot"-urile ca "own-slot"-uri. KEE suportă moştenire multiplă, existînd 12 moduri predefinite (utilizatorul îşi poate defini propriile moduri de moştenire):

1. OVERRIDE.VALUES (are prioritate valoarea de la nivelul cel mai scăzut în ierarhie);
2. UNION (include toate valorile fără duplicări, într-o listă de valori; valoarea din unitatea "fiu" este prima în listă);
3. RUNION (la fel ca la UNION, dar în ordine inversă);
4. SAME.VALUES (valorile unităţii "fiu" trebuie să coincidă cu cele ale unităţilor "părinte");
5. UNIQUE.VALUES (valorile nu pot fi moştenite);
6. VARIABLE.VALUES (valorile nu pot fi moştenite şi KEE nu notifică schimbarea valorii slotului);
7. MAXIMUM (cea mai mare valoare dintre valoarea locală şi cele ale părinţilor);
8. MINIMUM (cea mai mică valoare);
9. METHOD (codul unei metode e împărţit în MAINCODE, BEFORECODE, AFTERCODE şi WRAPPERCODE; partea MAINCODE este moştenită folosind OVERRIDE.VALUES);
10. VCSIMPLIFY (moştenire, în cazul claselor de valori);
11. UNION.EACH.VALUE (realizează reuniunea pe poziţiile corespunzătoare din lista de valori ale slotului);
12. RUNION EACH. VALUE (UNION. EACH. VALUE în ordine inversă).

Pentru eficienţă, KEE permite definirea de unităţi compacte, care sînt însă mai puţin flexibile decît unităţile standard şi nu permit adăugarea/eliminarea de sloturi.

2.2.2. Inferenţă şi control în KEE

Metodele de inferenţă în KEE includ reguli cu înlănţuire înainte/înapoi, demoni (valori active), metode (programare orientată pe obiecte) şi chiar programe procedurale (funcţii Lisp). Paradigma centrală în KEE este programarea orientată pe obiecte (metode şi demoni), deşi sistemul de reguli este şi el foarte puternic. Sintaxa pentru regulile forward şi backward este aceeaşi astfel încît, o aceeaşi regulă poate fi aplicată în ambele direcţii. De asemenea, înlănţuirea înainte şi cea înapoi pot fi folosite intercalat (înlănţuirea înainte este dirijată de date, cea înapoi este dirijată de scopuri sau interogări ale bazei de cunoştinţe). Există un mecanism încorporat de backtracking automat. Regulile sînt reprezentate ca unităţi şi pot fi organizate în clase de reguli (pentru claritate şi eficienţă).
Sistemul de reguli este extensibil (deoarece este o ierarhie de unităţi). Regulile pot fi scrise, fie într-o sintaxă apropiată de LISP, fie într-o sintaxă apropiată de limbajul natural, denumită TellandAsk. TellandAsk este limbajul de interfaţă cu o bază de cunoştinţe (interogări, aserţiuni, retractări).
Un exemplu simplu ar fi:

(assert' (toţi filosofii sînt oameni))

(query' (toţi ?X sînt oameni) :how.many' all)

Formatul regulilor KEE permite atît concluzii, cît şi

premise multiple. În KEE se face o distincție netă între regulile de acțiune și cele de deducție. Regulile de deducție nu adaugă noi asumpții la baza de cunoștințe, în vreme ce regulile de acțiune modifică baza de cunoștințe sau chiar creează lumi noi. Aplicarea regulilor este controlată de un mecanism cu agendă, care este modificabil de către utilizator.
Valorile active sînt reprezentate în KEE ca unități care pot fi atașate la mai multe sloturi. De asemenea, un slot poate avea mai multe valori active (demoni), care se declanșează, fie cînd slotul respectiv este inspectat, fie cînd este modificat.

2.2.3. ATMS (Assumption Based Truth Maintenance) și raționarea non-monotonică în KEE

KEE are un TMS bazat pe asumpții (conform teoriei lui Johann de Kleer). Dacă o concluzie depinde de o asumpție care s-a dovedit falsă, concluzia este retrasă. ATMS-ul menține structura de justificări, corespunzătoare aplicării regulilor de deducție, care au următoarea sintaxă:

(<nume regulă>
(while <precondiții>
believe <acțiuni/postcondiții>))

pentru a putea fi deosebite de reguli de acțiune, care se scriu:

(<nume regulă> (if<precondiții>
then <acțiuni>))

KEE permite existența lumilor multiple (KeeWorlds), eventual paralele, care pot fi create, modificate, șterse, fuzionate ("merged") și inspectate. O regulă de acțiune poate crea o nouă lume. În acest fel, se poate crea o ierarhie de lumi, în care există moștenire. Se pot crea lumi exclusive astfel încît, orice încercare viitoare de fuzionare a acestora va eșua. În terminologia KEE, o lume ("world") este un set de fapte nestructurate și/sau sloturi, care permite inferențe de tip non-monotonic (ipotetic), foarte utile în aplicații complexe (de exemplu, în planificare). Grafurile de derivare suportă atît înlănțuirea înainte, cît și înlănțuirea înapoi, avînd cîte un nod activ pentru fiecare regulă.

2.2.4. Alte facilități

KEE nu are mecanisme specifice pentru tratarea situațiilor de incertitudine, însă acestea pot fi dezvoltate de utilizator.
KEE are facilități grafice deosebite (bazate pe ferestre), pentru afișarea/ modificarea bazei de cunoștințe și, în special, a ierarhiilor de unități, reguli și lumi.
Regulile sînt executate de un interpretor, însă există și un compilator de reguli pentru îmbunătățirea performanțelor de viteză.
KEE este scris în CommonLISP, astfel că programatorul poate scrie metodele atașate unităților în LISP sau poate chiar modifica și/sau extinde sistemul. De asemenea, KEE poate fi interfațat direct cu limbajul C sau FORTRAN și indirect cu alte limbaje.
Depanarea în cadrul sistemului de reguli este facilitată de posibilitățile de trasare dinamică a grafului AND/OR de inferență, care are noduri active și admite puncte de întrerupere.
Se pot afișa, de asemenea, grafuri de explicație a secvențelor de raționare.
SimKit este un instrument grafic (orientat pe meniuri și mouse), construit peste KEE, care ajută programatorul în modelarea și simularea evenimentelor discrete. SimKit include un editor de biblioteci, verificarea automată a modelului, logica controlului, obiecte compuse, colecții de date, generatoare de numere etc. Există o bibliotecă de obiecte de simulare, din care utilizatorul poate alege în mod interactiv. SimKit este un produs separat de la IntelliCorp, la fel ca alte două produse, care asigură un acces mai bun la baze de date relaționale. "KEE conection" conectează aplicațiile KEE la bazele de date relaționale. Specificațiile pentru această conectare sînt făcute folosind un "editor de mapare" grafic. Din interiorul aplicației KEE, se generează dinamic interogări SQL.
IntelliScope este construit peste KEE și asigură instrumentele pentru utilizarea bazelor de date, de către un utilizator novice.
"PC-Host" este sistemul "țintă" (de cost mai scăzut) pentru rularea aplicațiilor dezvoltate cu KEE și constă dintr-un DEC VAX și din mai multe terminale (IBM PC). Aplicațiile dezvoltate cu KEE sînt în întregime portabile, deoarece sînt scrise în CommonLISP, inclusiv interfețele grafice cu utilizatorul (scrise în CommonWindows, care este un sistem de ferestre, independent de mașina dezvoltată de IntelliCorp).
"KEEpictures" este un sistem pentru dezvoltarea interfețelor grafice, independent de mașină și extensibil. Obiectele sînt reprezentate ca unități cu atribute grafice.
ActiveImages este o bibliotecă de imagini de obiecte, construite cu "KEEpictures", care pot fi atașate la sloturi pentru afișarea sau modificarea valorilor curente, ale sloturilor respective. La schimbarea valorii sloturilor, demoni atașați afișează modificarea. Unele imagini sînt "actualizatori", în sensul că modificările de pe ecran (făcute de exemplu cu mouse-ul) se reflectă și în baza de cunoștințe. KEE poate fi folosit pentru aplicații de complexitate ridicată în diagnoză, monitorizare, control în timp real al proceselor, planificare, proiectare, simulare etc. Sistemul de dezvoltare rulează pe mașini puternice: Symbolics, LMI, TI Explorer, Xerox 1100, Sun-3, Dec VAX, IBM PC/RT, HP 9000/ 300.

2.3. Knowledge Craft (KC)

Knowledge Craft dezvoltat de Carnegie Group Inc (650 Commerce Court Station Square PittSburgh, PA 15219 (412) 642-6900), este un sistem hibrid, bazat pe "frames" (cu moştenire definibilă de utilizator), reprezentînd integrarea versiunii Carnegie-Mellon a limbajului bazat pe reguli forward OPS5, cu PROLOG-ul şi cu limbajul CRL, de reprezentare prin "frames".

"Frame"-urile sînt utilizate pentru reprezentarea cunoştinţelor declarative, în vreme ce cunoştinţele procedurale sînt implementate prin reguli şi demoni. KC permite raţionamente de tip nonmonotonic, folosind contexte.

Arhitectura sistemului include:

1. limbajul de reprezentare CRL (Carnegie Representation Language);
2. o versiune îmbunătăţită a limbajului bazat pe reguli forward OPS5 (numită aici CRL-OPS);
3. o versiune de PROLOG, cu sintaxa LISP (numită CRL-PROLOG);
4. Common LISP;
5. programare orientată pe obiecte;
6. un mecanism bazat pe agenda pentru programarea orientată pe evenimente;
7. sistemul grafic;
8. sistemul de ferestre;
9. sistemul de comandă.

KC este implementat în Common LISP.

2.3.1. Reprezentarea cunoştinţelor în KC

Limbajul de reprezentare CRL este o extensie a limbajului SRL, dezvoltat la Universitatea Carnegie-Mellon. CRL este bazat pe "frame"-uri şi reţele semantice şi include:

1. o reprezentare a schemelor (frames);
2. relaţii definibile de utilizator;
3. semantică;
4. meta-cunoştinţe;
5. demoni;
6. restricţii asupra domeniului, cardinalităţii şi domeniului de valori ale sloturilor;
7. valori implicite;
8. contexte;
9. un model de gestiune a bazelor de date (pentru versiunile de pe VAX).

În KC, orice este o schemă. Există trei tipuri de sloturi: relaţii, atribute şi metode. Programatorul poate defini comportamentul unui slot (moştenire complexă, relaţii inverse etc.).

Metacunoştinţele sînt reprezentate prin scheme ataşate de altă schemă, alt slot sau altă valoare dintr-o schemă. Mecanismul de moştenire include, în afara moştenirii de tip "superclasă-subclasă", şi "clasă-instanţă", posibilităţi de definire a moştenirii de către programator (inclusiv specificarea de căi de moştenire în ierarhia de scheme). Relaţiile pot avea inverse, astfel că, la momentul specificării unei valori într-una din relaţii, valoarea inversă este automat calculată. Obiectele pot moşteni valori sau metode. Valorile atributelor moştenite sînt calculate de fiecare dată cînd sînt necesare, însă pot fi eventual memorate prin setarea unui switch. Strategia implicită, de căutare a valorilor moştenite este "breadth-first" (parcurgerea ierarhiei de obiecte pe nivele), căutarea oprindu-se la prima valoare găsită sau colectînd toate valorile. Căutarea poate fi controlată prin specificarea căilor de moştenire.

2.3.2. Inferenţă şi control în KC

În KC există două mecanisme de inferenţă diferite, bazate pe reguli: CRL-OPS (pentru înlănţuire înainte) şi CRL-PROLOG (pentru înlănţuire înapoi), care sînt însă bine integrate cu limbajul de reprezentare a cunoştinţelor CRL. De asemenea, este suportată programarea orientată pe obiecte (subset LOOPS).

Sintaxa regulilor forward, respectiv backward, diferă. O regulă forward CRL-OPS se scrie:

(p <nume-regulă>
<premise>
→
<acţiuni>)

Partea stîngă a regulii (premisele) poate conţine constante, variabile, conjuncţii ({valori}), respectiv disjuncţii (<<valori>>) de valori şi operatori relaţionali.

Partea dreaptă a regulii conţine acţiuni care pot fi: forme LISP, scheme CRL şi operaţii legate de contexte, apeluri PROLOG, transmiteri de mesaje sau operaţii specifice OPS5.

Nu există ierarhii de reguli, acestea fiind memorate nestructurat. Strategiile de rezolvare a conflictelor (în cazul în care mai multe reguli au premisele îndeplinite) se bazează pe momentele de timp, la care respectivele condiţii au devenit adevărate.

Cele două strategii de rezolvare a conflictelor sînt LEX şi MEA.

LEX înseamnă ordonare lexicală şi constă în:

Pasul 1: LEX ordonează momentele în care premisele fiecărei reguli au devenit adevărate, formînd cîte o listă ordonată descrescător; listele astfel obţinute sînt ordonate lexicografic;

Pasul 2: dacă mai există conflicte, se selectează regula cea mai specifică (cu cele mai multe condiţii);

Pasul 3: dacă mai există conflicte, se face o alegere arbitrară.

MEA (Means Ends Analysis) este la fel ca LEX, cu excepţia primului pas:

Pasul 1: ordonează regulile doar după momentul de timp, corespunzător primei condiţii; dacă

rămîn conflicte, acestea se rezolvă că în strategia LEX (prin ordonare lexicală).

Strategia implicită este LEX. Nu se pot specifica direct priorităţi semantice.

Înlănţuirea înapoi este realizată de CRL-PROLOG şi are o sintaxă specifică LISP. O regulă CRL-PROLOG se scrie:

```
(assertr
        (<concluzie>
        <
        <premise>))
```

Ambele sisteme de reguli CRL-OPS şi CRL-PROLOG includ cîte un compilator pentru îmbunătăţirea performanţelor de viteză. Regulile OPS sînt compilate în reţele RETE.

Programarea orientată pe obiecte, din KC conţine obiecte (scheme), metode, demoni (specificaţi în sloturi, scheme sau meta-scheme).

Mecanismul de gestiune al evenimentelor este bazat pe o agendă şi este utilizat pentru simulare şi "scheduling". El distinge între diferitele lumi şi planifică evenimentele în raport cu un ceas real sau simulat. În Simulation Craft, există trei tipuri de planificare a evenimentelor:

- imperativ;
- simulare (ordonarea cronologică a evenimentelor, relativ la un ceas simulat);
- planificare (executarea unui task, la un moment specificat de ceasul de timp real).

2.3.3. Raţionament de tip non-monotonic în KC

KC nu are TMS (Truth Maintenance System), însă acesta poate fi realizat de utilizator folosind mecanismele de contexte, demoni şi metacunoştinţe Raţionamentele de tip non-monotonic sînt suportate de mecanismul de Contexte, care permite existenţa lumilor multiple prin crearea de copii virtuale ale bazei de cunoştinţe. Contextele pot fi create, fuzionate sau şterse. Dacă o ipoteză care a dus la crearea unei noi lumi se dovedeşte acceptabilă, atunci consecinţele acesteia pot fi transferate în contextul lumii părinte.

2.3.4. Alte facilităţi

KC nu are încorporate mecanisme pentru tratarea situaţiilor de incertitudine, însă utilizatorul poate încorpora informaţiile privitoare la certitudine, în metacunoştinţele asociate unei scheme.

KC are un editor grafic al bazei de cunoştinţe (cu meniuri şi mouse). Sistemul include, de asemenea, următoarele modele:

1. CRL Workcenter (editor grafic pentru scheme, verificarea automată a reţelei, "help", afişarea ierarhiei de scheme etc.);
2. CRL-OPS Workcenter;
3. CRL-PROLOG Workcenter.

Facilităţile de depanare includ posibilităţile de trasare grafică şi de afişare a arborilor de derivare. Versiunea VAX este compatibilă cu formatul RMS, pentru baze de date. Sistemul de ferestre este independent de maşină. Sistemul de comandă ("Command System") este folosit pentru dezvoltarea interfeţelor aplicaţiilor cu utilizatorii. Language Craft este un mediu pentru dezvoltarea interfeţelor în limbaj natural şi poate produce la ieşire cod LISP, PROLOG, OPS5 sau SCHEME. KC rulează pe Symbolics 3600, TI Explorer, DEC VAX, Sun-3.

2.3.5. Instrumente pentru dezvoltarea aplicaţiilor specifice

Carnegie Group dezvoltă nuclee pentru rezolvarea aplicaţiilor uzuale; de exemplu:

1. gestiunea producţiei (modelarea activităţilor industriale, monitorizare, planificare, simulare, generare de rapoarte);
2. diagnoza (care include două componente: un controlor de procese, ataşat la senzori şi un mediu pentru construirea şi testarea bazelor de reguli, pentru diagnosticare);
3. proiectarea asistată de calculator, în electronică (sinteza logică a circuitelor VLSI, analiza circuitelor);
4. inginerie software (conţine instrumente software pentru ajutarea programatorilor în configurarea modulelor de program şi în gestiunea proiectelor);
5. prelucrarea de texte, în domenii restrînse.

2.3.6. Concluzii

Knowledge Craft este mult mai greu de învăţat decît sistemele de dimensiuni comparabile KEE sau ART, din cauza slabei integrări a diferitelor module (de exemplu: sintaxa în cele două sisteme de reguli, ca şi denotarea variabilelor diferă). Totuşi, KC este foarte puternic şi flexibil. Fiind un sistem proiectat să funcţioneze în timp real, KC este potrivit pentru aplicaţii ca: planificare/ "scheduling", controlul proceselor etc.

2.4. S. 1.

(Teknowledge Inc. 1850 Embarcadero Road P.O. Box 10119, Palo Alto, CA 94303).

S.1. (evoluat din EMYCIN) este un nucleu pentru construirea sistemelor expert, specializate îndeosebi pe probleme de clasificare. Cunoştinţele declarative (faptele) sînt reprezentate prin "frame"-uri, iar cele procedurale prin reguli.

2.4.1. Reprezentarea cunoştinţelor în S.1

Tipurile de bază din S.1. sînt următoarele:

1. clase;
2. tipuri de clase;
3. atribute;
4. ierarhii de valori;
5. reguli;
6. blocuri de control;
7. funcţii;
8. funcţii externe;
9. blocuri de control externe.

Fiecare obiect al bazei de cunoştinţe are o definiţie care constă din: tipul obiectului, numele acestuia şi o listă de sloturi.

Clasele nu sînt organizate în ierarhii, în schimb se pot defini ierarhii folosind tipuri de clase. Tipurile de clase pot fi folosite pentru organizarea claselor şi a altor tipuri de clase, însă nu pot avea instanţe.

Atributele definite pe un tip de obiecte reprezintă funcţii unare. Se pot defini atribute pe produsul cartezian, a două sau mai multe clase de obiecte, obţinîndu-se, astfel, funcţii sau relaţii n-are. Valorile atributelor au un tip bine specificat (text, integer, real sau boolean). Atributele pot admite o singură valoare sau mai multe valori. În S.1. nu există structuri (records) sau vectori. O secvenţă de obiecte poate fi accesată folosind un set ordonat de valori de atribute, însă nu poate fi tratată ca o listă. Moştenirea este puternic restrictivă, în sensul că se pot moşteni sloturi de la tipurile de clase, dar nu se pot moşteni valori. Atributele pot avea valori implicite (care însă pot fi moştenite). Valorile implicite coexistă cu noile valori în sloturile care admit mai multe.

2.4.2. Inferenţă şi control în S.1.

S.1. este un sistem mai aproape de paradigma programării procedurale, decît toate celelalte. Există totuşi un sistem de reguli, care însă pune accentul pe proceduri şi pe controlul explicit. Toate regulile sînt "backward" (cu înlănţuire înapoi). O regulă este activată de o încercare (făcută într-un bloc de control) de a determina valoarea unui slot, pe care această regulă l-ar putea determina (pentru că este concluzia sa). Nu există reguli forward, deci nici inferenţe dirijate de date.

Motorul de inferenţă poate crea instanţe ale claselor şi poate determina valori ale atributelor.

În S.1. există 4 tipuri de cunoştinţe procedurale:

1. blocurile de control (sînt proceduri care specifică o secvenţă de taskuri şi sînt scrise în limbajul de control al lui S.1.);
2. funcţiile (sînt scrise în limbajul lui S.1. şi pot accesa valorile atributelor din baza de cunoştinţe, întorcînd valori calculate);
3. funcţiile externe (scrise în C, pot accesa date externe din baze de date şi pot fi integrate cu mediul exterior);
4. blocurile de control externe (sînt proceduri C, care pot accesa date externe din baza de date şi pot fi integrate cu exteriorul).

Definiţiilor de atribute, li se pot asocia blocuri de postdeterminare, care reprezintă blocuri de control, activabile la crearea instanţei unui atribut sau la setarea valorii atributului. Acest mecanism este echivalent cu demonii din alte limbaje. Blocurile de control sînt strict procedurale. Deoarece inferenţele în S.1. sînt strict (explicit) controlate de programator, problema nedeterminismului în aplicarea regulii nu se mai pune, deoarece nu mai există necesitatea unei strategii de rezolvare a conflictelor (necesară în alte limbaje în care activarea regulilor este gestionată dinamic).

Regulile pot fi grupate în categorii şi sînt automat indexate, după atributele pe care le calculează. Cînd un bloc de control caută o valoare de atribut, se poate specifica categoria regulii. Dacă pot fi activate mai multe reguli, ele sînt invocate în ordine arbitrară. Nu se pot specifica priorităţi şi nici nu se poate selecta o regulă din mai multe reguli care se pot activa (decît folosind categorii de reguli).

În S.1. nu se pot schimba valorile atributelor. Pentru a evita această limitare, se foloseşte o tehnică numită "reconsiderare", care crează instanţieri succesive şi care o poate detecta pe cea mai recentă.

În S.1. nu există TMS şi nici raţionamente de tip non-monotonic. S.1. conţine mecanisme de explicaţie, bazate pe sloturi speciale ale obiectelor.

2.4.3. Factori de incertitudine

Tratarea situaţiilor de incertitudine este bazată pe modelul EMYCIN. Factorii de certitudine (FC) pot avea valori reale între <-1.0> şi <1.0>, existînd următoarele modalităţi de combinare:

1. asigurarea unui FC unei concluzii a unei reguli, cînd toate premisele sînt "adevărate" (adică au un FC mai mare ca limita pozitivă <0.2>):

FC (concluzie) = FC (regulă) * FC (premise);
FC (premise) = min (FC al premiselor individuale, dacă acestea sînt legate prin AND);
max (FC ai premiselor individuale dacă acestea sînt legate prin OR şi dacă cel puţin unul este ><0.2>).

2. Reguli multiple, care suportă o concluzie.

Combinarea lui <1.0> (respectiv <-1.0>) cu X este <1.0> (respectiv <-1.0>).

a) dacă FC1 şi FC2 > 0:
FC combinat = FC1 + FC2 * (1 - FC1) = FC1 + FC2 - FC1 * FC2;
b) dacă FC1 şi FC2 < 0:
FC combinat = FC1 + FC2 + FC1 * FC2;
c) dacă FC1, FC2 sînt de semn opus,
FC combinat = (|FC1| + |FC2|) / (1 - min (|FC1|, |FC2|)).

2.4.4. Alte facilităţi

S.1. are un editor al bazei de date, bine structurat şi posibilităţi grafice de trasare (Event Trace Panel, care arată derivările de valori ale atributelor, Rules Window). Mediul de achiziţie a cunoştinţelor este interactiv, bazat pe meniuri şi mouse, însă sistemul nu oferă instrumente pentru dezvoltarea interfeţelor grafice cu utilizatorul. Sistemul este scris în C şi este foarte rapid. El permite compilarea incrementală (ceea ce măreşte viteza de dezvoltare). Din cauza structurării regulilor pe categorii, se pot scrie programe cu mii de reguli, fără a asista la o explozie combinatorială (timpii de rulare sînt liniari cu numărul de reguli).
S.1. rulează pe Dec VAX (sub Unix sau VMS/C), MicroVax, IBM PC/RT, PC/AT (sub Xenix), AT&T 7300, NCR Tower 32, Hp 9000/300, Sun, Apollo, Tandem T16, Motorola 6350 & 8000 IBM, mainframes.

2.5. Goldworks

Goldworks este un produs al firmei GoldHill Computers (163 Hardward St. Cambridge MA 02139), care rulează pe IBM PC/AT (512 octeţi memorie de bază, 5MB memorie extinsă şi 7MB spaţiu pe hard-disc) şi oferă aproape toate facilităţile din sistemele "mari", care rulează pe maşini sofisticate (KEE, ART, Knowledge Craft). Sistemul este livrat pe 32 dischete plus documentaţie foarte bină scrisă, astfel ca Goldworks este cel mai complet nucleu de sisteme expert, existent pentru calculatoarele personale. Este scris în GCLISP (versiunea Gold Hill de Common LISP) care include pachetul de dezvoltare "Gold Hill 286/386 Lisp Developer" şi poate fi interfaţat cu LOTUS 1-2-3, dBASE, C, limbaj de asamblare şi reţele locale (LAN). Sistemul este rezident în memoria extinsă şi are o interfaţă grafică, care rulează sub MS-Windows.
Goldworks reprezintă cunoştinţele sub formă de "frames" care au un sistem de reguli forward şi backward.
Sistemul de "frames" permite moştenire multiplă de sloturi, ierarhia de frames fiind numită "latice". Orice, inclusiv întreaga bază de cunoştinţe, poate fi tratat ca un slot într-un "frame". Principalele tipuri de obiecte existente în GoldWorks, în afară de "frames" şi reguli sînt: instanţe, relaţii, aserţiuni, demoni, seturi de reguli, "attempts", sponsori şi agende.
"Frame"-urile pot fi instanţiate, existînd deci o distincţie clară între relaţia clasă-subclasă şi relaţia clasă-instanţă. Instanţele pot fi valori ale sloturilor unei alte instanţe, ceea ce permite existenţa obiectelor compuse. Sloturile pot avea faţete care-i descriu funcţionalitatea. Relaţiile, similare predicatelor PROLOG sînt de trei feluri: aserţiuni relaţionale, aserţiuni funcţionale şi relaţii date de funcţii LISP.
În Goldworks, metodele ataşate "frame"-urilor se numesc "message-handlers" şi sînt definite cu define-handler. Mesajele sînt transmise cu "send-msg". Regulile sînt bidirecţionale, însă pot fi făcute să funcţioneze unidirecţional (fordward sau backward) printr-un marcaj special. Algoritmul de "pattern matching" utilizează algoritmul Rete, care este foarte eficient (şi admite variabile în reguli). De asemenea sînt suportaţi factori de incertitudine şi priorităţi.
Pentru controlul procesului de inferenţă, Goldworks foloseşte un mecanism bazat pe agendă (similar cu KEE sau ART) care însă poate fi activată/dezactivată de un sponsor.
Goldworks menţine o reţea de dependenţe logice între fapte, permiţînd retragerea automată a concluziilor obţinute pe baza unor aserţiuni care s-au dovedit false (TMS = Truth Maintenance System). Această facilitate este necesară pentru raţionamente de tip nonmonotonic, în lumi diferite (Goldworks nu suportă direct raţionamente de tip nonmonotonic). În afara interfeţelor grafice cu utilizatorul, Goldworks poate fi interfaţat cu Microsoft Lattice C. Goldworks este nucleul de sisteme expert, care oferă cele mai multe facilităţi dintre sistemele care rulează pe calculatoare personale.

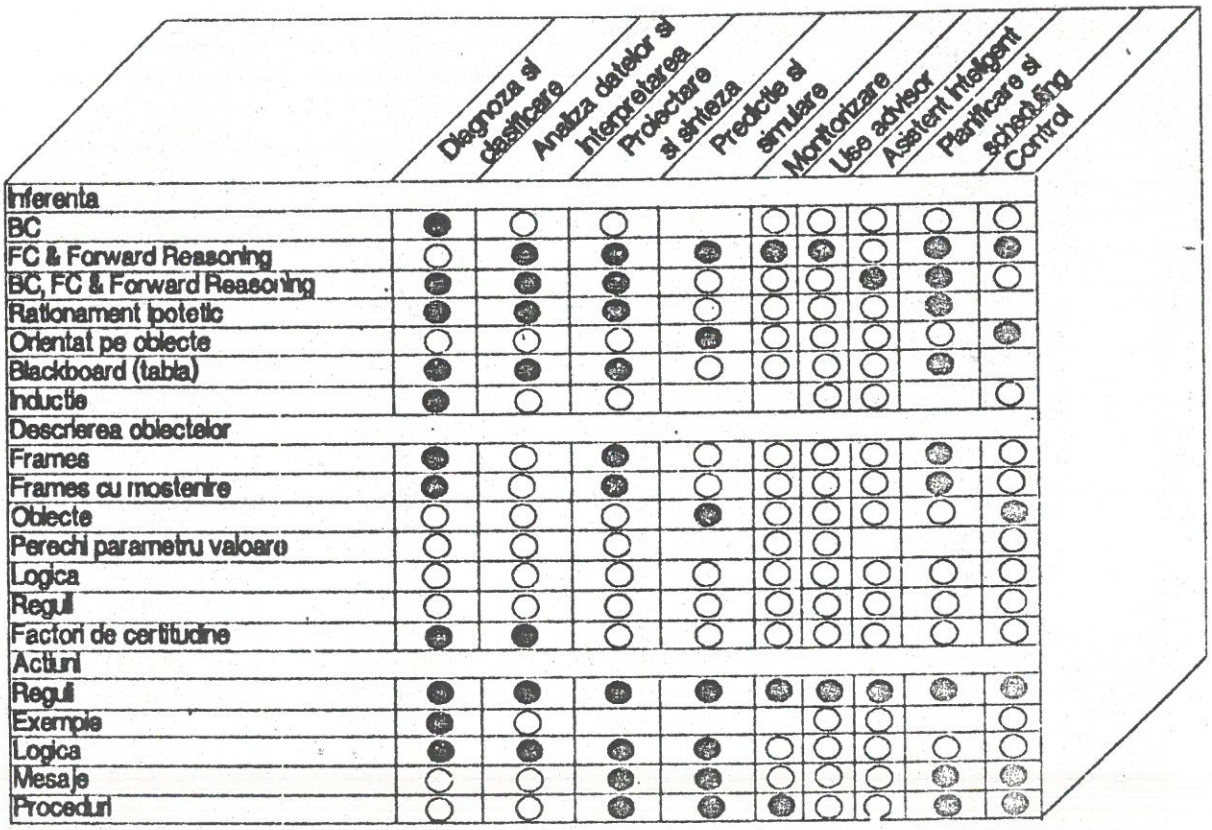
2.6. Nexpert Object

Nexpert Object este un nucleu puternic pentru construit sisteme expert; este bazat pe reguli şi scris în C pentru a fi rulat pe IBM PC/AT sau Mac Plus. Acesta are facilităţi care sînt găsite şi la sistemele "mari", care rulează pe maşini sofisticate. Sistemul permite programatorului să grupeze regulile în categorii, regulile admiţînd variabile şi avînd acelaşi format, atît pentru forward, cît şi pentru backward chaining. Nexpert Object are editoare pentru reguli, clase şi obiecte (editoarele sînt grafice).
Reprezentarea cunoştinţelor este făcută prin obiecte ("frames"), cu moştenire multiplă. De asemenea, se face distincţie între relaţia clasă-subclasă şi clasă-instanţă, un obiect putînd fi instanţa mai multor clase. O caracteristică importantă este posibilitatea reprezentării grafice a procesului de inferenţă.
Sistemul este produs de Neuron Data Corp (444 High St. Palo Alto, CA 94301; (415) 321-4488.

3. Calitatea şi posibilitatea de utilizare a unui nucleu pentru construit sisteme expert

Calitatea şi posibilitatea de utilizare a unui nucleu pentru sisteme expert sînt dependente de factori precum:

- capacităţi funcţionale, domenii de aplicabilitate:
  - clasificare (bazată sau nu pe un model "adînc" al domeniului);



Tabelul 1 Importanta diferitelor atribute ale nucleelor de sisteme expert pentru o aplicatie data

SISTEMUL	ART 3.0	KEE 3.0	KNOWLEDGE CRAFT	PICON	S.1	ES ENVIRON/ VM OR MVS	ENVISAGE	KES
Utilizări funcţionale								
Clasificare	x	x	x	x	x	x	x	x
Proiectare	x	x	x		x			
Planificare/"Schedulling"	x	x	x					
Controlul Proceselor	x	x	x	x				
Reprezentarea B C								
Reguli								x
Reguli structurate	x	x	x	x	x	x	x	
Factori de certitudine					x	x	x	x
Nr. maxim de regull								
"Frames" cu moştenire	x	x	x	x	No inherit.			
Orientat pe obiecte	x	x	x	x				
Logică	x	x	x		x			
Exemple								
Exemple structurate								
Proceduri	x	x	x		x	x	x	x
Motorul de inferenţă								
Forward chaining	x	x	x	x	FR	x	x	
Backward chaining	goal rules	x	x	x	x	x	x	x
Demoni	x	x	x	x	x		x	
Blackboard (tablă)	x							
Modelare temporală	x	x		x				
TMS	x	x						
Meta Control	x	x	x	x	x	x	x	x
Logică	x	x	x		x			
Inducţie								
Calcule matematice	x	x	x	x	x		x	x
Contexte	x	x	x					
(Viewpoints, lumi)								
Pattern matching								
Variabile	x	x	x	x				x
Literali	x	x	x	x		x		
Secvenţe	x		x	x			x	
Segmente	x		x	x				x
Wildcards (sabloane)	x		x					
Interfata programatorului								
Crearea bazei								
de cunoştinţe								
Procesor de texte	x		x		x	x	x	x
Procesor de linii		x	x	x	x			
Editor al bazei								
de cunoştinţe (BC	x	x	x	x	x	x		
Meniuri	x	x		x	x			
Verificarea consistenţei		x		x	x			
Reprezentarea grafică a								
bazei de cunoştinţe (BC)	x	x	x	x	x			
Trasarea inferenţei	x	x	x	x	x	x	x	x
Utilizări grafice								
pentru utilizator	x	x	x	x	e			
Formatul ecran	x	x	x	x	x	x		
Simulare grafică	x	Sim Kit	Simulation Craft	x	e			
De ce ?	x	x	x		x	x	x	x
Cum ? (explicaţii)	x	x	x	x	x	x	x	x
Expandarea explicaţiilor	x	x	x	x	x	x	x	x
Help online	x	x	x		x	x	x	
Help pentru sintaxă	x	x	x	x	x		x	
Extensibilitate	x	x	x	x			x	

Tabelul 2. Atributele unor nuclee de sisteme expert comerciale

M.1	NEXPERT OBJECT	PERSONAL CONSULT +	EXSYS 3.0	EXPERT EDGE	ESP FRAME ENGINE	NSIGHT 2 +	TIMM	RULE-MASTER	KDS 3	1 st CLASS
X	X	X	X	X	X	X	X	X	X	X
	X							X	X	
	X								X	
	X						X	X	X	
X			X							
	Cataloged	X		X	X	X		X		X
X	X	X	X	X		X	X	X	X	X
1000	2000	800	5000			2000			6000 from 4000 examples	
	X	X			X				X	
					X					X
							X	X	X	X
	X	X			X					
FR	X	X	X		X	X	X	X	X	X
X	X	X	X	X	X	X		X	X	X
X	X	X			X				X	X
	X			X					X	
X	X	X			X					
					X					
X	X	X	X	X	X	X	X	X	X	X
									X	
X	X		Numeric	Numeric	Numeric & String	Numeric	X			Numeric & Logical
			X							
				X						X
X			X	X	X					X
	X	X	X			X	X	X	X	X
X	X	X	X	X		X		X	X	X
X	X	X	X	X			X	X	X	X
X	X	X	X	X		X			X	X
	X				X					X
X	X	X	X	X		X	X		X	X
	X									e
X	X	X	X	X	X	X		X	X	
X	X	X	X	X	X	X		X	X	TREE
X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X		X	X	X
X	X	X		X				X	X	
	X	X		X	X			X	X	X

SISTEMUL	ART 3.0	KEE 3.0	KNOWLEDGE CRAFT	PICON	S.1	ES ENVIRON/ VM OR MVS	ENVISAGE	KES
Software								
Limbajul utilizat	COMMON LISP	COMMON-LISP	COMMON-LISP	ZETA-LISP.C	C	PASCAL	PASCAL	C
Compilabil	Increm.	Increm.	Incremental	x	x	Incremental	x	x
Alte limbaje cerute	COMMON LISP	COMMON LISP	COMMON-LISP	LISP		CMS, GDDM, PASCAL/VS		
Sistem de operare	VMS UNIX				UNIX VM/CMS VMS	VM/CMS or MVS	VMS	MSDOS on PC
Calculatoare pe care rulează								
Simbolică	x	x	x					
LMI	x	x	x	x				
TI Explorer	x	x	x	x				
VAX	x		x		VMS ULTRIX		x	x
XEROX		x						
IBM PC								x
MacIntosh								
TI Prof.								
APOLLO		x			x			x
SUN	x	x			x			x
IBM 370					x	x		
Altele		x	x		x		x	x
Interfața sistemului								
Lang. Hooks	Via Host Computer	Via Host Computer	Via Host Mach	Via Host Mach	C, Others	x	PASCAL etc.	C
DB Hooks	Via Host Computer	Via Host Computer	Via Host Mach	Via Host Mach	Via "	x	x	x
Interfața utilizatorului								
Facilități grafice								
Linie		x	x	x	x	x	x	x
Meniuri	x	x	x	x	x	x	x	x
Grafică	x	x	x	x	e			e
Simulare	x	Sim Kit	Simulation Craft	x	e			
De ce ?	x	x	x	x	x	x	x	x
Cum ? (explicații)	x	x	x	x	x	x	x	x
Help	x	x	x	x	x	x	x	x
Initial Pruning	x	x						
Soluții multiple	x	x	x					x
Exemple salvate			x	x	x	x	x	x
Răspunsuri multiple și incerte de la utilizator	x	x				x	x	x
What If	x	x	x	x	x		x	x
Compania producătoare	Inference	Intelli-corp	Carnegie Group	LMI	Tek-now-ledge	IBM	Sys. Designers Software	S/W Arch. & Engr.
Cost	$65K	$52K Includes training	$50K Includes Training	$60K	$25K 1 user mach. $45K mult.user	$35K	$25K on VAX $15K om MicVAX	$4K PC $7K WkStns $25K Vax
Costul sistemului de run-time	$1-8K	PC Host + VAX $20K +0.5K/ PC	None Yet		$9.5K	$25K		10 % of Develop Sys.

Tabelul 2. Atributele unor nuclee de sisteme expert comerciale (continuare)

M.1	NEXPERT OBJECT	PERSONAL CONSULT +	EXSYS 3.0	EXPERT EDGE	ESP FRAME ENGINE	NSIGHT 2 +	TIMM	RULE-MASTER	KDS 3	1 st CLASS
C x	C Incremental	LISP x	C Incremental	C x	Prolog 2 Incremental	Turbo-PASCAL x	FORTRAN x	C x	8086 Assembly	PASCAL x
MSDOS	MSDOS	MSDOS VMS	MSDOS	MSDOS VMS	MSDOS	MSDOS	MSDOS	C compiler DOS 3.0 UNIX,VMS	MSDOS	MSDOS
			VMS UNIX				x	x		
x	AT	x	x	x	x	x	x	AT	x	
	x	x					x			x
								x		x
	x						x	x		
x	C,PASCAL DBASE III	LISP DBASE III	x x	x x	Prolog 2 Via "	x DBASE II	x	Most x	PASCAL BASIC Reads Assem. Lang.	ANY x
x	x	x x	x	x x	x x	x	x	x	x	x
	x x	e	e						x x	e
x x	x x	x x	x x	x x	x x	x x	x	x x	x x	x
x	x	x x	x	x x	x	x x		x	x x	x
	x x	x x	x x	x x	x		x		x x	x
x	x	x x	x x	x x		x x	x x		x x	x
Teknow-edge	Neuron Data	TI	Exsys. CA Intell	Human Edge	Exp Sys Inter	Level -5-R	GRC	Radian	KDS Dev Sys	Progr. in Motion
$5K	$5K PC-AT $3K MAC	$3K	$395 PC $5K VAX	$2500 Adv $5000 Pro	$895	$485	$1.9K PC $19K others	$995 PC $5K Wkst $17.5K Vax	$1495	$495
$50	$1K	$75 First Unit	One Time Fee	$50	By Agree-ment	Negotl.	No Fee	$100 PC $500 Unix/VMS	Based on Quality	No Fee

- proiectare;
- analiză;
- planificare;
- monitorizare;
- controlul proceselor;

- dimensiunea sistemului (respectiv numărul maxim de reguli, suportat de sistem);
- costul sistemului, costul de dezvoltare şi de întreţinere;
- viteza de dezvoltare a sistemului şi viteza lui de execuţie;
- uşurinţa învăţării utilizării sistemului (care depinde nu numai de claritarea conceptelor folosite, dar şi de calitatea documentaţiei);
- portabilitatea sistemului şi posibilitatea interfaţării cu software-ul existent;
- suportul din partea companiei producătoare, inclusiv formarea (pregătirea);
- gradul de satisfacţie al programatorului (dependent de interfeţele cu programatorul şi facilităţile grafice, inclusiv de depanare etc) şi al utilizatorului (dependent şi de interfeţele cu utilizatorul).

Nucleele pentru construit sisteme expert, deşi foarte variate din punctul de vedere al posibilităţilor (de la sistemele sofisticate, scrise în LISP şi bazate pe mai multe paradigme de programare, pînă la sistemele inductive, simple, scrise în C şi orientate mai mult către achiziţia cunoştinţelor) sînt încă la început. Tendinţele de perfecţionare sînt orientate către sisteme mai rapide, portabile, care să ruleze pe maşini de uz comun şi care să permită, totodată, o mai mare flexibilitate în programare şi întreţinere. Se pune accentul pe reprezentări din ce în ce mai sofisticate ale cunoştinţelor şi pe tehnici speciale, de inferenţă (de exemplu arhitecturi de tip "blackboard"). De asemenea, încep să apară sisteme expert, specializate pentru rezolvarea anumitor clase de probleme (sau taskuri generice) ca: diagnoză, planificare, controlul proceselor etc. Sînt considerate, de asemenea, foarte importante posibilitatea conectării la software-ul existent şi calitatea/interactivitatea interfeţelor cu programatorul şi cu utilizatorul.

Bibliografie

1. **GEVARTER, W.B.:** *The Nature and Evaluation of Commercial Expert System Building Tools.* In: Computer, May 1987, pp. 24-41;
2. **TELLO, E.:** *Object Oriented Programming for Artificial Intelligence*, Addison Wesley, 1989;
3. **SICHEL BEACH, SH.:** *An Analysis of High-end Expert System Tools*, In: IEEE Expert, 1987
4. *The Spang Robinson Report* - August 1986, November 1986, February 198', May 1987;
5. **FIKES, R., KEHLER, T.:** *The Role of Rule Based Representation in Reasoning.* In: CACM-September 1985, pp. 904-920;
6. **de KLEER, J.:** *An Assumption Based Truth Maintainance.* In: Artificial Intelligence, No. 28, 1987. p. 333;
7. **FILMAN, R.:** *Reasoning with Worlds and Truth Maintainance in a Knowledge Based Programming Environment.* In: CACM, No. 31 (4), April 1988, pp. 382-401.