REXX2001—CHOSEN LANGUAGE OF MAN AND MACHINE

MARC VINCENT IRVIN
MANAGEMENT VISIONS INSTITUTE

SPEAKER: Marc Vincent irvin

FROM: Management Visions Institute

EVENT: REXX Symposium 1992 (May 4th and 5th)

DATE: Copyright 5/92

OBJECTIVE: Show NL reality, and power NL can give REXX

TITLE: REXX2001 - Chosen Language of Man and Machine

THESIS: Subtle adjustments to REXX could make it the premier language for intelligent systems and occasional programmers.

PREMISE: If science fiction is any barometer, we are heading toward computers able to understand what we say, and say what they understand. Many feel that day is far away because any system that smart requires human intelligence; true human intelligence, according to AI experts, won't be in our life times. Those experts are wrong. A language that needs no programming class, and responds coherently to English could, according to my experiments with REXX, Expert Systems (ES), and Natural Language (NL), be on Personal Computers (PCs) in no time. The secret to achieving the goal ahead of schedule lies in purging all the unnecessary baggage left over from the evolution of computers.

When computers were first conceived they were expected to handle the same kinds of information people do: words, numbers, and symbols. After their invention reality set in with bits and bytes, disks and tapes, sequential and random files, relative and hierarchical databases, and on and on and on... REXX and cover functions can do away with most of the junk that waste programming time. Once purged REXX, ES, and NL can work together to produce a language that occasional programmers will love, and professional programmers can build intelligent systems with.

EMPOWERING CODERS FOR THE 21ST CENTURY

Hardware computing power has grown geometrically over the past twenty five years. Software computing power has grown very little. It is easy to see why this is. The hard problems in hardware have been bridged and standardized because hardware is directed by a predictable element - computer programs. On the other hand, software has no such luxury. The hard problems in software are precipitous and transient because software is directed by an unpredictable element - human programmers. Achieving similar gains is not impossible, however. The hard problems in software can be bridged and standardized if software is directed by predictable elements that cultivate the unpredictable natures of human programmers. Below is a list of road blocks programmers face, a proposed set of solutions, and a some recent experiments devoted to empowering programmers.

Road blocks to empowering programmers for the 21st Century.

1. Differing data, field, and integer types.
2. Differing call formats and complex command syntax.
3. Differing sub-system interfaces and data access methods.
4. Lack of real-time development and run-time features.
5. Lack of interactive and friendly development methods.
6. Lack of cognitive psych, decision support, and AI models.

Solutions to empowering programmers for the 21st Century.

1. Dynamic data typing by use on words, numbers and symbols.
2. Common call methods with toggling for special syntax use.
3. Cover functions whose inputs look the same to all users.
4. Add date/time based initiators and scripts.
5. Workspaces, smart debug features, and natural language.
6. Intergrate Rule, Case, Genetic, Object, & Neural Models.

My experiments with empowering programmers for 21st Century.

1. Using REXX as a platform gets around first road block.
2. RUN() uses interpret not CALL, and ES/NL options toggle.
3. FILECHNG, REXXRDR, and REXXWRTR will standardize all I/O.
4. CLKQUEUE gives temporal power needed for smart programs.
5. REXXCALC w/APL's online tools and PARACODE NL syntax ANS.
6. All empowerments put in CLKRULES' leave room for more.

FILECHNG is a file copy utility with options that:

1. Finds fields and replaces them with other fields.
2. Selects portions of a file or its records to work on.
3. Input can be from disk, reader, or VM command.
4. Sequence checks, purges dupes, and writes change reports.
5. Replace field can be used to select, purge, insert data.
6. One powerful option puts code wherever FINDS occur.

One problem, involving RACF based MVS security, required a list to be made of TSO users that were given IDs, but had never used their IDs. Only two passes of file change were done on an input file that contained all the multi-record RACF reports of the TSO IDs not used in the last 60 days. A sample set of records from RACF ID report follows...

```
USER=TSOUSR   NAME=TED BUNDY   OWNER=SYSTEM  CREATED=88.289
 DEFAULT-GROUP=SYS1    PASSDATE=00.000  PASS-INTERVAL=60
 ATTRIBUTES=NONE
 REVOKE DATE=NONE   RESUME DATE=NONE
 ...
 NO-MODEL-NAME
 LOGON ALLOWED   (DAYS)     (TIME)
```

Below is REXX code that 1) selects the records, and 2) formats them into single lines for examination and display.

```
/* MVI */
'FILECHNG LISTUSER ASOF0392 A PASSDATE WORK A',
    '*PICKRECS',    /* IF REPLACE FIELD = // THEN PICK REC */
    'USER=(1 15) //',     /* PICK RECORDS WITH USER'S ID */
    'SSDATE=00 //',/* PICK RECS FOR NEVER USED IDS */
    'NO-MODEL //', /* PICK REC THAT WILL ACT AS RPT END */
IF RC ^= 0 THEN EXIT 100
'FILECHNG PASSDATE WORK A = = =',
    '*RECDLM(NO-MODEL)', /* MAKE ONE REC OF MANY RECS */
    '*PICKRECS',    /* IF REPLACE FIELD // THEN WRITE REC */
    'SSDATE=00.000 //',  /* PICK NEVER USED RECORDS ONLY */
    /* NOTE, // CAN BE FOLLOWED BY AN EXITNAME TOO       */
    '*OUTEXIT(PASDAT:)'  /* TELL FILECHNG NAME OF CHK LGC*/
IF RC ^= 0 THEN EXIT 200
EXIT 000

PASDAT:  /* THIS ROUTINE IS READ/INTERPRETED BY FILECHNG */
PARSE VAR $REC 1 'USER=' UID ' ',
               1 'CREATED=' ADDAT ' ',
               1 'SSDATE=' PASDAT ' ',
               1 'INTERVAL=' PWINT ' '
IF PASDAT = '00.000' & ADDAT <= '92.004' & PWINT = '60',
   THEN SAY 'USERID ('UID') NEVER USED SINCE ADD ON' ADDAT
/* TO CHG O/P REC PUT VAL IN $REC, TO DEL PUT '' IN $REC */
DOC:
SAY 'REXXNAME: PASSDATE      '
EXIT 000
```

CLKQUEUE is a scheduling utility with options that:

1. Run VM commands based on "date" and/or "time" requested.
2. Requests may be run once or requeued every n days.
3. Commands can be rerun every n hours, minutes, or seconds.
4. Time scripts are possible as CLKQUEUE can call itself.
5. Runs have return codes useable by later clock requests.
6. Its powerful options execute all kinds of REXX code.

Below is a ad-hoc sampling of the many ways that REXX code can be invoked on a date and time basis.

```
CHKRTC:  92/03/22 1 . . 0 IF LIBSFND() THEN 'ERASE UID LIB'
LOVE:  93/02/13 09:00:00 1.H1*17:00.93/02/14 0 0 0,
   MSG * DON'T FORGET THE VALENTINES DAY FLOWERS.
VMUSERID 92/03/22 03:00:00 1 . . 0,
  RUN(VMUSERID:)   /* execute the command beneath EOF */
  IF RC = 0 THEN DO
    'STATE VMUSERID DATA A'
    IF RC ^= 0
      THEN SAY 'ERROR BUILDING VM USERID FILE.'
      ELSE SAY 'VM USERID FILE BUILT OK.'
    END
* RUN NEXT COMMAND ONCE EVERY WEEK...
  GRPPRTADMBKT: 92/04/30 10:00:00 07 92/04/23 10:04:34 0,
    IF GRPRPTADMBKT = 0,
       THEN DO
        'CMSQ RACFMVS GRPRPORT GRP ADMBKT'
        IF RC^=0 THEN SAY 'CLKQUEUE ERROR RUNNING RACF RPT'
        END
       ELSE SAY 'CLKQUEUE ERROR RUNNING GRPPRTADMBKT AT',
            RUNTIME'.'
* CHECK THE NETWORK EVERY 10 MINUTES...
CHKNET: 92/04/28 13:10:00 01.M10 92/04/28 13:00:09 0 CHKNET
CMD1 92/01/22 14:30:00 1 0 0 0 DIRLOG RSCS
CMD2RC 92/01/22 23:59:00 01.M10 0 0 0 CP QUERY RSCS
 92/01/22 23:59:00 01.M10 0 0 0,
    IF CMD2RC = 45 THEN MSG OP *** RSCS IS DOWN! ***
CMDX: 92/01/22 23:59:00 01.M10 0 0 0 ,
    IF CMD1 ^= 0 & CMD2RC ^= 0 THEN DO
        "MSG OP ****************************"
        "MSG OP UNABLE TO RECOVER RSCS..."
        END
* RUN SPECIAL SET OF CLOCK COMMANDS ON NEW YEARS DAY.
ENDOFYEAR: 93/01/01 20:00:00 0 0 0 CLKQ EOYCYCLE
EOF
VMUSERID:
/* BUILD THE VM DIRECTORY FROM DIRMAINT SEGMENTS */
'DIRBUILD'
'STATE USER DIRECT A'
IF RC = 0 THEN RUNRC = 0; ELSE RUNRC = RC
```

REXXCALC is a calculator/memory utility with options that:

1. Calculate variables in adding machine or formula modes.
2. Manages workspaces via SAVE, LOAD, DROP, & LIST commands.
3. Passes commands to VM when they are not calculations.
4. Executes REXX code from command line or saved variables.
5. Keyboard assistant via CLKQUEUE's intelligent scheduling.
6. Many powerful options give APL like capabilities to REXX.

Below is a sample session where the user has to:

1) Figure number of cylinders required for a new file.
2) Test how the SUBWORD command works as they are developing a new REXX program.
3) Edit a function named BENEFITS, change some of the formulas, and execute it.

```
REXXCALC   /* X; prompts the user for a response.     */
REXXCALC - RELOADED 9 VARIABLES FROM PROFILE.
REXX IS ACTIVATED INTERACTIVELY...
X; vars
/* INIT VARIABLES FOR REXXCALC EXEC */
$RSCS      = 'CP SMSG RSCS'
$SMART     = 'CP VMC SMART'
$AUTOLOG   = 'CP SMSG AUTOLOG1 AUTOLOG'
$ULOG      = $SMART 'D ULOG'
FMTDATE    = TRANSLATE('34756812',910522'//','12345678')
$OP        = 'CP MSGNOH OP'
UTC_BFRTX = 3746 + 530 + 0 + 0 + 242 + 36 + 1015
BEN_ALLOW = 4112
PER_CHECK = (UTC_BFRTX - BEN_ALLOW)/24
X; $op Peter please mount tape 3003 on 580, Thx mvi.
X; $rscs q sysprtx q
X; ben_allow       /* ask what co paid beny portion is? */
4112
X; rexx say subword('a b c',4)

X; rexx say subword('a b c',2)
B C
X; * next line does calculation within another workspace.
X; rexxcalc ofcspace my_area = (deska+grade6space-isle)
442
X; weekhours = 8.5 + 9.0 + 8.0 + 8.0 + 9.5
43
X; reccnt = 327000
X; blksize = 4096    /* no. of bytes per block */
X; bpc = 180         /* blocks per cylinder */
X; reqcyls = format(((reccnt*132)/blksize/bpc)*2,1,0)
X; save          /* will save prior 5 variable. */
REXXCALC - SAVED 14 VARIABLES IN PROFILE WORKSPACE.
X; xedit benefits calcrexx
X; run(benefits 5050 ben_allow)
X; quit
INTERACTIVE REXX IS CANCELLED BY USER.
```

NODELOAD is a tool for building natural language code that:

1. Compensates for user spelling errors based on context.
2. Maps input vocabulary and loads them into node words.
3. Values that follow keywords are put in its node word.
4. Node words once set are useable by REXX based rules.
5. REXX based rules can be coded as pseudo English.
6. After registration first node word represents call tag.

Below is a sample of the NODELOAD catagories, keywords, and basic vocabulary used in mapping pseudo English grammar. It was inspired by an article written by Richard Brooks titled "A Natural Language Interface to MVS" published in the October 1991 issue of the TECHNICAL SUPPORT JOURNAL.

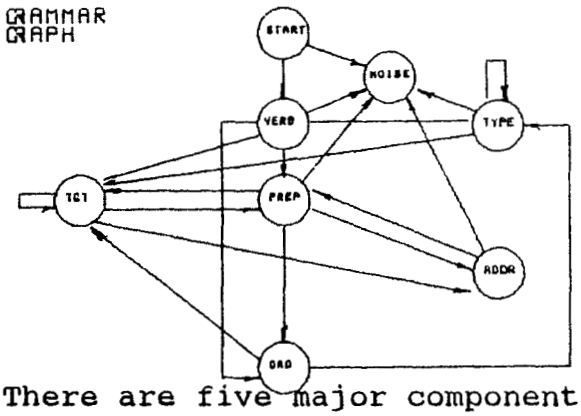
```
INITQUES: /* node type = node names allowed to follow it. */
TYP.START    = CMND NOISE
TYP.CMND     = PREP TYPE ORD TGT
TYP.PREP     = 'CHK_ADDR: ORD TGT ADDR NOISE'
TYP.TYPE     = TYPE PREP TGT NOISE
TYP.ADDR     = PREP
TYP.TGT      = 'CHK_ADDR: PREP ADDR TGT'
CHK.CMND     = SHOWME SHOW LIST GIVE PRINT DISPLAY
CHK.PREP     = AT ON IN TO FROM FOR OF
CHK.TYPE     = TAPE ALLOCATION RECORD UR UNIT ONLINE TSO
CHK.TGT      = DISK CPU TAPE STORAGE MEMORY PATH DRIVE
CHK.NOISE    = ME ADDRESS PLEASE THE A INFORMATION FOR AN
CHK_ADDR:
PARSE VAR RUNSTR $PS $ST
IF VERIFY(WORD($ST,$PS),'0123456789ABCDEF') > 0
   THEN RUNRSPNS = ''
   ELSE RUNRSPNS = 'ADDR'  /* TELL LGC WORD IS AN ADDRESS */
CMND: SELECT
      WHEN FIND('TAPE DISK DRIVE',TGT) > 0 THEN RUN(DODU:)
      WHEN FIND('MEMORY CPU PATH',TGT) > 0 THEN RUN(DODM:)
      OTHERWISE SAY TGT 'NOT RECOGNIZED AS A TARGET.
      END
RUN(ISSUECMD:)
```

```
Sample user input follows with diagnosis options turned on.
PLS SHOW ME THE ALLOC INFO ON DISK 6C1 TO 6C4!
(PLS) miswritten, (PLEASE) set instead.
(ALLOC) abbreviated, (ALLOCATION) set instead.
(INFO) abbreviated, (INFORMATION) set instead.
Attribute (NOISE) automatically set to (PLEASE).
Attribute (CMND) automatically set to (SHOW).
Attribute (TYPE) automatically set to (ALLOCATION).
Attribute (NOISE) automatically set to (INFORMATION).
Attribute (PREP) automatically set to (ON).
Attribute (TGT) automatically set to (DISK).
Attribute (ADDR) automatically set to (6C1).
Attribute (PREP) automatically set to (TO).
Attribute (ADDR) automatically set to (6C4).
FINALCMD = D U,DASD,ALLOCATION,6C1,4
```

NL THEORY FOR NODELOAD

The directed graph SHOW-ME grammar explained by Richard Brooks for developing natural language commands published in the October 1991 issue of the TECHNICAL SUPPORT JOURNAL has been automated.

Below are a list of sentences that the experimental NODELOAD logic can handle and the a sample grammar graph.



Lst al usrs. Dsply tape 001.
Gv usx dxsk 7a1 for 16.
Pls print me the cpu memory,
starting at address e000.

There are five major components in building the "Missouri, Show Me" natural language command parser. When working with rule based PARACODE all of the following steps are automatic, and require no direct coding by the user. The NODELOAD example shows how natural language is done without resorting to rule based code.

1. A Grammar Graph is made to depict how the parts of each sentence will interact. For example, what type of word can begin the sentence. The basic words in the vocabulary are going to be loaded into these words so they should be descriptive. Nouns, verbs, and modifiers are basic parts of speech common in SVO grammars. TYP variables are used to fully represent grammar graphs like the one shown above.
   TYP.attribute = attribute names that can follow
2. A Vocabulary Definition is done using the CHK variable where each attribute get attached to it all the valid words that may be loaded into it.
   CHK.attribute = list of valid words or symbols.
3. Spelling Verification is done first by context then against all attributes. For example, if the unrecognized word follows a verb and then only the attributes valid after verbs are checked for transposed letter and the like.
4. Registration, when a parsed word is successfully found in a valid CHK.attribute list and the word is loaded into its corresponding attribute for later use.
5. Construction, when all the words have been successfully registered into attributes the name of the first attribute registered is used in a RUN() statement. Thus if a VERB like "Show" was the first word in the NL command then the user would code get control via routine called:
   VERB: /* process all registered words via REXX */
6. Execution, when the user has fully constructed the command he must then execute it in such a way that the user may customize or override its use.

PARACODE is a natural language programming system that:

1. Allows users to code ES rules using pseudo English code.
2. Uses multi-word synonyms to give English flexibility.
3. Allows grammatical use of probabilities and fuzzy logic.
4. Allows user to converse logically with knowledge bases.
5. Has frame attributes like ask, why, how, check, & doc.
6. Many powerful options put code wherever users need it.

Below is a sample Expert System (ES) rule written in Paracode.

```
VCRADVSR: /* This rule advises what model VCR to purchase */
If the VCR type is VHS and heads is over three and FX wanted
   then the best purchase is probably a super VCX_1000
   else the best buy is likely to be a dumb Record_Mate99
INITGOAL: The maingoal is best buy and a three is a 3
INITQUES: /* These entries set synonyms and frame values. */
   syn(FX FX:is 'special:effects special_effects')
   syn(VCR CAM 'video:machine video:recorder')
   syn(buy purchase); syn(type model); and syn(heads tracks)
   syn(FX_wanted 'FX:wanted FX:needed') and syn(: super dumb)
   syn(VCR_type 'VCR:type') and syn(best_buy 'best:buy')
The ask.mainexit is 'The VCRADVSR says buy a' best buy'.'
The ask.VCR_type is 'Enter preference: VHS or BETA.'
The chk.VCR_type is VHS BETA  /* check allowed values  */
The dft.VCR_type is VHS       /* default value is VHS  */
The chk.FX_wanted is "CHK_YESNO:"   /* a dynamic call  */
The chk.heads is 2 3 4 5
The fmt.heads is 1 1 numeric  /* one byte numeric only */
The why.heads is "Because better VCR's have 4 or more."
```

Below is a sample dialog with VCRADVSR... (R; is user reply.)

```
Enter preference: VHS or BETA.
R; VHS
Please enter value for (HEADS).
R; which
Your input options are 2 3 4 5.
R; 4
Please enter value for (FX_wanted).
R; yes
BEST_BUY            = CNF(0.80)  VCX_1000
The VCRADVSR says buy a VCX_1000.
R; video machine model
VCR_TYPE = VHS
R; special effects needed
FX_WANTED = 1
R; reset
Enter preference: VHS or BETA.
R; The CAM model is VHS; tracks is 4; and FX needed is not true
BEST_BUY            = CNF(0.70)  Record_Mate99
The VCRADVSR says buy a Record_Mate99.
```

NL THEORY FOR PARACODE

The Subject, Verb, and Object AI paradigm works well with REXX.

REXX it could be expanded to enter NL mode if a switch is set.

```
RULENAME:   /*  basic clauses:  IF antecedent THEN consequent */
[ IF s v o [ conj s v o etc...] THEN] s v o [ conj s v o etc... ]
```

[ ]	Means fields within are optional. Only s v o required.
IF	Valid conditionals are IF and WHEN.
s v o	Subject, verb, object can be represented by a single symbolic such as with true/false values and task() executions, or as separate multi-word phrases beneath.
s	Subject can be up to three distinct words if the Computer Oriented Dialog SYN() synonyms have specified a single root word. They are the equivalent to nouns or noun phrases (NP) in English and basic ATN theory.
v	Verbs can be up to three distinct symbols and/or words if the SYN() synonyms have specified a root word. They imply the action to take against its subject and object, and are referred to as verb phrases in basic ATN theory. Normally verbs are compare symbols, but in advanced NL may be called tasks that have boolean RCs.
o	Objects can be a literal or symbol, or a phrase up to three words if the SYN() synonym's root is defined. It represents a noun and may be preceded or followed by a fuzzy logic or confidence factor (Ie. big/little or probably/definitely). If the object value, after synonym substitution is found to be unknown translation is interrupted, and the object is put into a queue for resolution. Resolution occurs by first looking for a RULE that has a consequent clause that sets the value. Next, framed variables are checked to see if their is a defined procedure to solve the rule. For example, doing a database retrieval. Finally, a previously specified text is used or a text is manufactured that asks the user to supply the value or choose the default.
conj	Conjunctions are limited to and, or, exclusive or, and their character equivalents. They can be three word phrases, but no logical use has been found for using multi-word phrases as conjunctions.

NOTES:

1. Advanced synonym substitution can handle antecedents separately from consequents.
2. Rules can have any number of consequent or antecedent consequent combinations.
3. The RUN() can dynamically invoke any task at will.
4. REXX and its NL equivalent can coexistent in PARACODE.
5. All s and o values, otherwise known as, nouns can be easily updated using NODELOAD directed graph substitution.

PARATALK is a natural way to show properties and relationships.

1. Users can easily encode semantic net diagrams of knowledge.
2. Encoding may consist of pseudo English facts, rules, and acts.
3. Acts invoke scripts, models, and step by step operations.
4. Backward reasoning will try to resolve unmatched phrases.
5. PROLOG deep structure sample: does_well_in(Student,Discipline)
6. Conditionals (ie. =<>) can now be relationals or operationals.

Below is a sample expert system rule written in Paratalk.

```
MAJORADV: /* This rule advises student in selecting a major  */
If a student is interested in a specific discipline,
   and student does well in the subject,
   and the subject is important in discipline,
   and the discipline is in demand,
   then student should major in the discipline
      and forward student transcript to Dean of discipline
   else student should not major in the discipline
INITGOAL: The maingoal is student should major in discipline
/* Examples below show how facts may be initially entered.  */
John does well in math and John is interested in business
Math is important in business and business is in demand
Bill does well in math and Bill is not interested in business
INITQUES: /* These entries set synonym and frame values.     */
   syn(is_interested_in 'is:interested:in')
   syn(does_well_in 'does:well:in') and syn(is_in 'is:in')
   syn(is_important_in 'is:important:in')
   syn(should_major_in 'should:major:in')
   syn(not:should should not) and syn(student 'name:of:student')
   syn(send transfer mail forward)  /* keyword for action logic */
   syn(discipline 'specific:discipline')
The unknowns are 'student discipline subject'
The variables are 'student'  /* if symbol not set than infer it */
The relations are 'is_interested_in does_well_in is_in'
The relations are relations 'is_important_in should_major_in'
The actions are 'send'
The ask.student is "Please enter the student's first name."
The ask.mainexit is "Enter 'reset' to get some fresh advice."
SEND: say "Transcript is being forwarded to" discipline "Dean."
```

Below is a sample dialog with MAJORADV... (R; is user's reply.)

```
Please enter the student's first name.
R; John
student should_major_in discipline = JOHN should_major_in BUSINESS
Transcript is being forwarded to BUSINESS Dean.
Enter 'reset' to get some fresh advice.
R; reset
Please enter the student's first name.
R; Bill
student should_major_in discipline = BILL should_major_in BUSINESS not
Enter 'reset' to get some fresh advice.
R; quit
```

ADVANCED NL THEORY IN PARATALK

In PARACODE a NL syntax was demonstrated that allowed a user to write conventional programming code in subject, verb, and object (SVO) based pseudo English. In NODELOAD a NL syntax was shown that facillitated the building of context free grammars. Using PARACODE and NODELOAD together it was suggested that pseudo English code and dialogs could be achieved as a side affect of Expert System development.

PARATALK takes the expression of NL code and queries to a higher plain by incorporating pattern matching, dynamic verb manipulation and execution, and dynamic variable entry and assignment.

Wherever, SVO phrases are allowed, so too are SRO (subject, relation, object) clauses and ARC (action request commands).

SRO Subject relation object clauses put pseudo English consequents into a pattern table for interrogation as antecedents. "John likes soda" is a typical SRO clause. Unlike PARACODE the central word does not have to get converted to "= > <" symbols. In PARATALK anything goes, just tell the interpreter how to recognize your SRO clauses by loading its relational word or predicate into a variable named RELATIONS. For example, to be able to say "If John likes soda then soda tastes good" you do the following in the INITQUES: section.

```
RELATIONS = 'likes tastes'
```

SRO clauses may be any length. When SROs are in an IF/WHEN (eg. antecedent) statement the associated symbols get looked up in a clause table. If found the IF/WHEN condition is set to true, otherwise it's set to false. If not part of an IF/WHEN condition the clause is put into a table. Matching clauses with previously stored clauses is called pattern matching. In the clause "John likes soda" all the words are taken literally. Now imagine having 10 "likes soda" clauses in the table, but for different people. To refer to all those people the following can be done in PARATALK.

```
RULE: Unknowns = 'Who'; and Relations = 'likes'
If who likes soda then do x is 1 for words(who)
   say word(who,x) 'likes soda'; end
```

In the above example all 10 names of people who like soda would be put into WHO. Sometimes a simple variable, set elsewhere in the logic, needs to be used. In that case enter the word within quotes. For instance, if soda were a variable filled with words like PEPSI or COKE then the PARATALK way to express it would be... "If anyone likes 'soda' then go buy soda".

ADVANCED NL THEORY IN PARATALK

In SRO PARATALK we were shown how to enter pseudo English assertions and interrogations like those that follow:

```
John likes soda
If John likes soda then soda tastes good
If who likes soda then do x is 1 for words(who)
   say word(who,x) 'likes soda'; end
If anyone likes 'soda' then go buy soda.
```

In addition to writing programmable code in English, and interrogating stored English clauses for truth there is another option. You can invoke special AI functions that carry out scripts or models of various scenes, events, speech, manual operations, and/or machine components. For instance, the consequent clause "go buy soda" is an imperative statement that requires a direct action.

ARC Action request command clauses have two parts. The first is the action part which corresponds to the program name used during CALLs from normal procedural code. The second is the request part which corresponds to the variables passed during normal procedural calls. However, for scene or model invokations to occur using pseudo English statements something must tell the PARATALK interpreter that this is an ARC phrase, rather than an SVO or SRO one. That something way is to load the primary action word (ie. verb) into a variable named ACTIONS. Since actions speak louder than words below is a sample of what I'm talking about, full blown PARATALK.

```
ACTIONS are 'go walk get fasten drive'
RELATIONS are 'likes';  and soda is PEPSI
GOBUYSODA: If anyone likes 'soda' then go buy soda
additional backward or forward chaining rules...
/* Basic script follows for going to the store */
GO: Parse var runstring whattodo withwhat .
If whattodo is 'buy' then do
     Item is withwhat  /* comments are allowed too */
     Walk to car; get in car; and fasten seat belt
     Drive to store and exit from car
     Walk into store and purchase store 'item'
     Drive back home
     end
/* The actions below can be external programs too.  */
WALK:  etc...
EXIT:  etc...
DRIVE: etc...
FASTEN: etc...
```

Basically, the above example neatly mixes all three NL methods: SVO, SRO, and ARC. It's pretty natural, wouldn't you say?

ADVANCED NL THEORY IN PARATALK

In SVO, SRO, and ARC we were shown how well PARATALK armed the Knowledge Engineer (KE) with the tools needed for building Conventional and Expert Systems using Pseudo English. Command clauses and phrases could be easily constructed that were declarative, interrogative, and imperative without requireing the KE to resort to arcane coding artifices. And there is much more...

External file data can handled be handled dynamicly using the LITERALIZER concept peculiar to the data driven pattern matching protocols of OPS5. With it files, sensors, and knowledge bases can be processed with 5th generation granularity using something resembling the well known object oriented paradigm. For instance, the clause "If cat weight is high and finickiness is extreme then type is Cheshire" is valid PARATALK terminology using LITERALIZERS.

OAV Object attribute value conditions can be employed in conjuction with SVO clause rules to provide name tags to fields in records. An example below builds a literizer for the CAT clause shown above. The basic format for entering a literalizer follows...
OPS(filename,objectname,attribute1 attribute2 etc...)

Actual sample...
OPS('FELINE DAT',CAT,'WEIGHT FINICKINESS TYPE')

Note, full power of the parse command is available.

OPS('RACF DATA A',PROFILE,'24 PW 32 1 'DATE=' DATE)

Also, some basic assumptions are now possible pertaining to context. For example, that CAT is the object for the attributes weight, finickiness, or type is easily implied. What's more cases of ambiguity (more than one literalized "object" contains the same attribute name) are easily resolved to most KE satisfaction by understanding that ununique attributes will get the object from one most recently used. Next, pronoun usages like "it" or "they" are possible and can be substituted, easily again with the most recently used object with a matching context definition (TYP.) An example of that kind of clause is shown beneath.

CAT_TYPE: If its weight is high and finickiness is extreme then its type is a Cheshire

In the above rule the value of "its" will be taken from whatever the last object happened to be that contained the attribute "weight".