Reference: Schomaker, L., Hoenkamp, E. & Mayberry, M. (1998). Towards collaborative agents for automatic on-line handwriting recognition. Proceedings of the Third European Workshop on Handwriting Analysis and Recognition, 14-15 July, 1998, London: The Institution of Electrical Engineers, Digest Number 1998/440, (ISSN 0963-3308), pp. 13/1-13/6.

Towards collaborative agents for automatic on-line handwriting recognition

Lambert Schomaker (NICI)
Eduard Hoenkamp (NICI)
Marshall Mayberry (Univ. of Texas, Austin)

NICI, Nijmegen Institute for Cognition and Information
University of Nijmegen, P.O.Box 9104
6500 HE Nijmegen, The Netherlands
Tel: +31 24 3616029 / Fax: +31 24 3616066
schomaker@nici.kun.nl

projects in the Cognitive Engineering group at NICI :

- On-line recognition approaches
- Comparison of forensic handwriting systems
- UNIPEN
- Multimodal speech and handwriting input
- Information Retrieval/Information Filtering
- Content-based image retrieval
- $\boxed{\text{Hybrid (NN/AI) modeling}}$

- **Multi-level information integration**
- **Agents: old for new?**
- **A triple-agent system**

"context"

- Use of context: a panacea for limited bottom-up classification performance?
- It is difficult to realize efficient use of context:
  - in case of complex input (cf. OCR of newspaper page vs. OCR of mail envelope)
  - under dynamic and free input conditions (writing a letter on a pen computer)

”context”

- What? (... are the relevant context bits: the ”frame” problem, Pylyshyn)
- How?
- No elegant solutions for multi-level information integration exist, as yet

syntax

- **Earlier experiments with NLP & on-line recognition: disappointing**
- **Parser for Dutch, using sentences from office context**
- **Batch architecture (strokes → characters → words → sentence)**
  - **use of context postponed until last word of sentence .**
  - **was slow!**
  - **written input may be syntactically incorrect**
  - **writers don't write job applications or love letters in this way**

syntax, continued

Needed: interactive approach
(e.g., incremental parser)

- **probabilistic language models**

  (works: but large corpus needed, many parameters)

- **grammars**

  (concise & explicit: but may lack information)

How to make a system which is modular and dynamically configurable?

old wine in new bottles?

- **O.G. Selfridge (1958)**

  Pandemonium: a paradigm for learning in mechanisation of thought processes. Proceedings of a Symposium Held at the National Physical Laboratory, pages 513–526, London, November 1958. HMSO.

- **Daemons**

- **Critics gallery**

- **Multiple experts**

- **Society of mind**

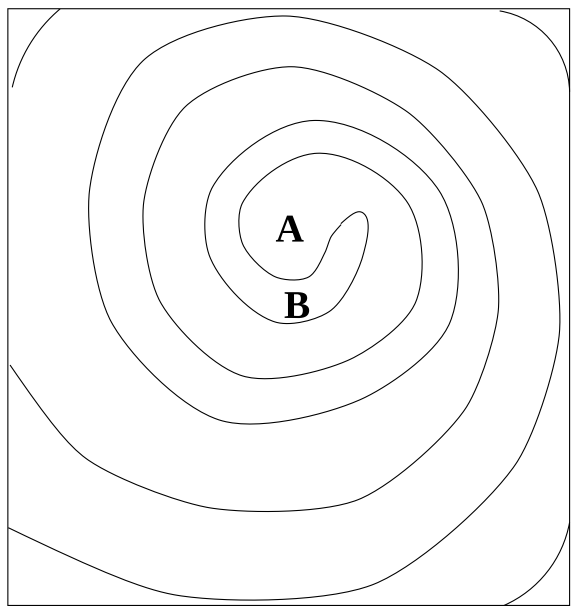
what's new?

- **Good definitions (Wooldridge & Jennings)**
- **Game theory, negotiation algorithms**
- **Multi-sensor fusion algorithms**
- **Learning**
  - **genetic algorithms**
  - **case-based reasoning**
- **Formalisms: Knowledge Interchange Format (KIF), Knowledge Query and Manipulation Language (KQML)**
- **Try:** `http://ontolingua.nici.kun.nl`

Potential for pattern recognition:

- **Realisation of complex decision boundaries**

again: the double spiral argument



- **Solve geometrically, e.g., with a MLP?**
  → Overfit!

- **Solve algorithmically, by search?**
  → More powerful!

experiment:

design a system

- **simple**
- **interactive (user is present & time is real)**
- **using bottom-up and top-down information**
- **using agent architecture**

→ **in order to see what the use of syntactic information may yield under simplified conditions**

design issues

- no natural language input but Scheme program input on a pen computer
- interactive:
  - no machine font substitution, leave ink 'as is'
  - use color for state feedback
  - give user full control, using virtual buttons, menus etc.
- bottom-up: Kohonen LVQ classifier of unistrokes
- top-down: Scheme parser (LR, incremental)

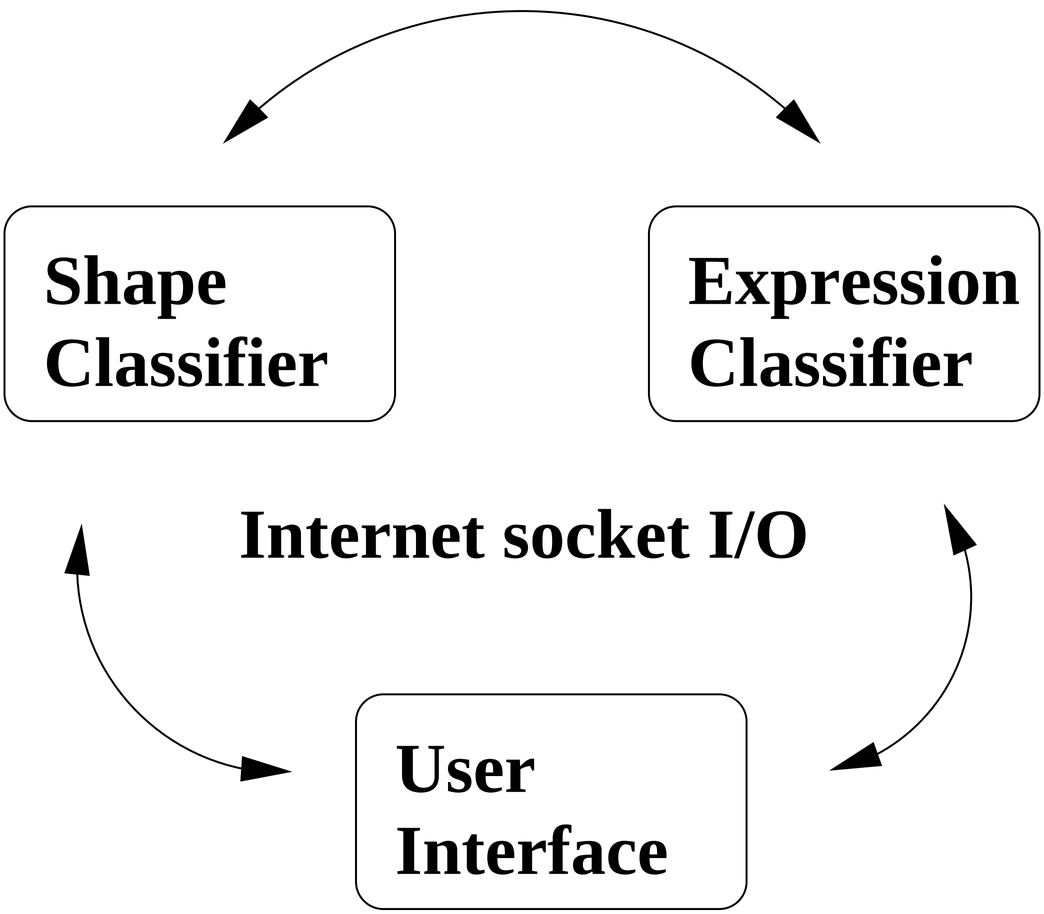
Scheme code example (towers of Hanoi)

```
hanoi.scm

(define ringlist
  (lambda (l n)
    (define mring
      (lambda (size)
        (cons 'ring size)))
    (if (= n 0)
        l
        (ringlist (cons (mring n) l) (- n 1)))))

(define mpole
  (lambda (ndisks)
    (cons 'pole (ringlist nil ndisks))))

(define disks
  (lambda (pole)
    (cdr pole)))
  .
  .
  .
  .
```

implementation

Agents:

1. shape classifier
2. expression classifier
3. user+user interface



Shape Classifier agent

- **Input: tokens of the Scheme language, written as unistrokes**
- **unistrokes, resampled to 60 samples**
- **Kohonen LVQ, nearest centroid match**
- **translate to** $\vec{\mu} = (0, 0)$
- **normalize rms radius to** $\sigma_r = 1$
- **feature vector:**
  - (x_k, y_k) **60 normalized coordinates**
  - $(cos(\phi_k), sin(\phi_k))$ **59 pairs**
  - **total 119x2=238**
- **training, 5-10 samples of a token**
- **learning rule** $f_j = \eta x_j + (1 - \eta) f_{j-1}$
- **token recognition rate** $\approx$ **85%**

Shape Classifier agent (pseudo code)

```
Init:
   init-communication
   read-table-with-token-templates
   ask-parser-for-type-of-each-token

while(true) {
   switch (read-request()) {
   case unistroke
      classify unistroke
      query-parser
      combine-parser-expectancy-and-shape-classification
      notify-user-agent

   case train
      update-token-shape-and-label
      notify-user-agent

      .
      .
   }
}
```

Expression Classifier for Scheme

- context-free grammar
- LR parser: incremental, no look ahead
- use lex/yacc (shift/reduce)
- tokens:

'	/
(	=
)	and
*	begin
+	BOOLEAN
-	case
.	CHAR
cond	let
define	let*
delay	NUMBER
do	or
else	set!
if	STRING
lambda	VAR

Expression Classifier for Scheme

- **Example of rule:**

```
state 29
        Def :  LPAR DEFINE_VAR Expr RPAR
        Def :  LPAR DEFINE_LPAR VAR RPAR Body RPAR
        Def :  LPAR DEFINE_LPAR VAR DefFormals RPAR Body RPAR

        VAR  shift 55
        LPAR  shift 56
        .  error
```

- **After each token: generate list of expected tokens and update state**

- **Requests to parser agent:**
  Accept_token
  Reset_state
  Delete_token
  Forward_token

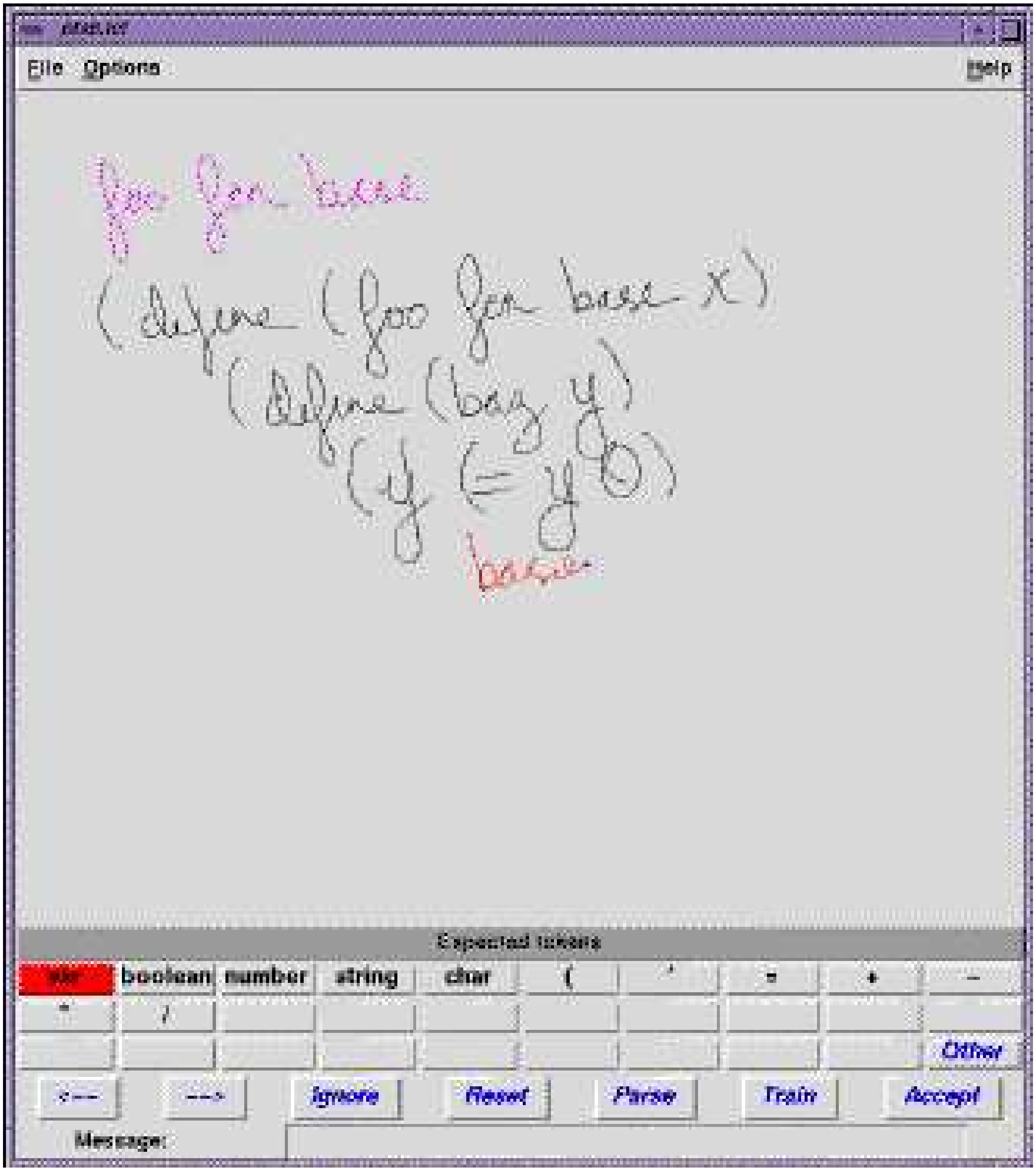
Expression Classifier agent (pseudo code)

```
Init:
   init-communication
   read-grammar

while(true) {
   switch (read-request()) {
   case token
       process-token
       update-parser-state
       return-expected-tokens

   case reset
       reset-parser-state
     .
     .
}
```

User Interface agent (pseudo code)

```
init-communication
start-parser
start-classifier

create-windows
create-event->task-bindings
      (ink events, parser events, classifier events)

wait-for-events(forever)
```

User Interface



interaction example

- **VAR expected**
- **token *dog* written**
- **token *dog* rejected → must be new token!**

results

good news:

- 100% 'recognition'
- users (Scheme programmers) like it!
- agent architecture is very convenient

results

bad news:

- individual information contributions by the agents must be analysed and quantified
- VAR becomes a problem in case of unconstrained scope
- NUMBER and STRING are open categories

results

Information content of Scheme source code

Symbols	$N_{alphabet}$	$^2log(N_{alphabet})$	*Entropy*	*Redundance*
Raw token stream	2003	11.0	6.3	4.7
Lumped token stream	28	4.8	2.4	2.4

(Based on corpus of N=27310 tokens.

Lumped means: use placeholders instead of actual instances of VAR.)

Entropy: $-\sum_{i=1}^{N_{alphabet}} p_i \, ^2log \, p_i$

results

Parser expectancy

Symbols	Avg. $N_{alternatives}$
Raw token stream (VAR scope=whole corpus)	1891.5
Raw token stream (VAR scope=single function)	97.4
Lumped token stream	16.0

(Scheme source-code corpus of 27310 tokens.

Lumped means: use placeholders instead of actual instances of VAR.)

→ If scope is not limited to a single function, the parser adds very little information. Reasons: users' naming creativity and the presence of constants (string, number).

- User actions are definitely needed!
- But their work can be made easier by using syntactical context
- The virtues of a grammar:
  "Look Ma' - No probabilities!"
- Beware of placeholders (name slots) in the grammar
- Just a first step towards the use of a multiple-agent architecture