IDEM -- INTERFACE INSTRUCTABILITY DEMONSTRATION

William Bricken
August 1984

CONTENTS

1. OVERVIEW

IDEM is an Interface Instructability Demonstration. This program is instructable; it learns procedures though dialogue, rather than being programmed by changing code.

Interaction with IDEM takes the form of a negotiated dialogue in restricted English. IDEM is highly task-oriented, and quite intolerant of confusion. Whenever it fails to understand the purpose of a communication, or whenever message is confused or ambiguous, IDEM will stubbornly insist upon clarification until it understands. (The unimplemented TRUST command can temporarily override this persistence.) In return, IDEM will act intelligently and with assistance when the user expresses a desire clearly.

NOTE: IDEM is a demonstration of a principle; it does not actually implement its capabilities. In describing IDEM, most claims are observable as output within a restricted domain. The underlying mechanisms that generate this output are not generalizable in the current demonstration-of-a-concept mode.

Section 9, for the practical minded, outlines the course of some demonstrations of IDEM. Sections 2 through 7 are descriptive, while Section 8 contains comments.

2. AVAILABLE STATIC KNOWLEDGE

IDEM knows a very small world that consists of two kinds of objects: persons and events. Events are sub-classified into meetings and demonstrations. There are four instances of PERSON, three instances of MEETING, and three of DEMONSTRATION.

In order to reflect intelligently upon its static knowledge, IDEM also knows about what it does and does not know. This meta-knowledge can be divided into:

2.1 KNOWLEDGE ABOUT THE WORLD

IDEM keeps a list of known instances of categories, known categories, known slots within each category, and known relations.

2.2 KNOWLEDGE ABOUT FUNCTIONS

IDEM knows what it can do by keeping a list of functions that it can perform. Each function can be interpreted as a capacity or as a command to act, depending upon the context. Capacities are extended via the LEARN command.

3. FRAMES AND SLOTS

Object concepts in IDEM are implemented as DEFSTRUCTs. Each category has a list of characteristics, or slots, that form the category frame. Each instance of a category has the slots filled with specific values.

When IDEM learns a new instance, it asks the user to fill in the slot values associated with the category of the instance. When IDEM learns a new category, it asks the user to modify the set of slots being inherited from the parent category (not fully implemented). IDEM obtains enough information to insert the new category in its category hierarchy.

Relations are treated as complex data objects, permitting the combination of existing categories under a new label.

4. AVAILABLE COMMANDS

Since IDEM exists in a symbolic world, it has the basic capacities for symbol manipulation. Its programming primitives are available to the user as a minimal vocabulary for instruction. Nominally, these are PUT, EQUATE, STORE, LOOP, and the logical connectives AND, OR, NOT, IF, THEN, OTHERWISE, ELSE, EQUIVALENT, ALL, SOME, and ONE. Sentences constructed of these elements and known objects as arguments are parsed into the appropriate control structure (not implemented).

The commands that are described below extend the programming primitives with options that help the user instruct the behavior of IDEM. TELL and REMIND update the user about the state of the instruction. LEARN and FORGET control the actual instruction process.

4.1 EXIT

EXIT terminates an instruction session.

4.2 TELL

TELL <concept> describes IDEM's knowledge of a concept.

Implementation: TELL works for all known categories, but only for known instances of PERSON. TELL can describe the commands LEARN, TELL, and (after learning) the commands SCHEDULE and NOTIFY.

4.3 REMIND

REMIND prints a trace of the current instruction session.

Implementation: REMIND is very limited in scope, and works only at some branches of LEARN a command. The branches are

a. What is the context (of the process)?
b. What is the type of effect?
c. Describe the effect.

4.4 FORGET

FORGET <concept> deletes knowledge of a concept. Forgetting may be local, global, or inherited (not implemented).

Implementation: A message is printed. If learned, SCHEDULE and NOTIFY can be forgotten. FORGET EVERYTHING reinitializes IDEM's knowledge base, causing IDEM to forget all items learned during the current instruction session.

4.5 LEARN

LEARN <concept> is the basic instructability mechanism. It initiates a series of knowledge directed questions that assure IDEM of a context and a set of facts sufficient to define the concept being learned.

4.5.1 *LEARN INSTANCES*

IDEM adds new elements, or atomic objects, by asking the user to specify a category and then fill in the slot values for the new instance of that category. If the category is recognized, IDEM confirms that it is correct by listing other elements in that category. Various other options also exist:

a. If the new category is currently a known element, the user can extend the category hierarchy;

b. If the category is unrecognized, the user can teach IDEM about it;

c. ? refers the user to the TELL option; and

d. X allows the user to exit immediately.

When the user exits without specifying a category for the new instance, IDEM attempts to find one by requesting a list of other elements in the same category. The user may reply with the following:

a. X or None exits this request;

b. ? reveals a No-help-available message;

c. The name of one other instance: IDEM responds that one instance is not sufficient to identify a category.

d. The names of two or more other instances:

Case 1: All are members of a known category (implemented for PERSON, MEETING, and DEMONSTRATION only). IDEM presents the category and tries to verify it by asking:

a) Is the category correct?

b) Is the category a superclass of the desired category? If so, can the user name the subcategory?

c) Are there other instances similar to the target instance that don't fit into this category? Regardless of the reply, IDEM learns the category by rote.

Case 2: All but one are members of a known category (PERSON, MEETING or DEMONSTRATION only): IDEM identifies the category and pursues the mismatch by asking:

a) Is the category correct for both the mismatch and the new instance?

b) Is the mismatch an instance that is somehow an exception?

c) Is the identified category a superclass, with the new instance and the mismatch representing different subclasses? If so, can the user identify a distinguishing predicate for these subcategories? (*Implementation:* when the predicate is INSTRUCTABLE, IDEM incorporates it.)

d) Is the identified category not relevant?

This extended exploration is intended to demonstrate IDEM's capability to explore data-bases in an attempt to learn by generalizing from instances. (*Evaluation:* This section is weak theoretically.)

4.5.2 LEARN CATEGORIES

When a new category is introduced, IDEM tries to find where it fits within its category hierarchy. Options include:

a. A known parent category: IDEM lists the subcategories and asks which will be children of the new category. (Implemented for EVENT only.)

b. An known element as a parent category: IDEM extends the category hierarchy.

c. An unknown parent category must be taught to IDEM.

d. ? refers to TELL.

e. X exits.

4.5.3 LEARN RELATIONS

IDEM requests the number and domain of the arguments for a new relation. Argument domains are recognized by the functions in "Learn Categories". The special case of an element as an argument is queried as being restrictive.

Implementation: if the arguments are PERSON and DEMONSTRATION, IDEM notices that it knows of two other relations with the same arguments and asks the user if the intended relation is different than these.

Finally IDEM asks the properties of the new relation (*Implementation:* only one is accepted). ? will list known (but not implemented) properties.

Known relations are accepted as data via the STORE command.

4.5.4 LEARN COMMANDS

Functional concepts are described to the user as plans that achieve a goal. A functional perspective is one in which a command changes the state of the system.

When asked to learn a command, IDEM requests the context (only one context is implemented: CALENDAR-WORLD) and the number and domain of the arguments to the command.

Then IDEM asks for the effect of the command. As many effects as desired can be entered, each is identified as a direct effect or as a side effect (*Implementation:* there is no difference between the two.). Sequences of effects build sequential plans, or procedures. When a description is entered, IDEM checks each word for recognition, and notifies the user accordingly. If all words are recognized, the command is learned. IDEM asks to learn all unrecognized words.

Implementation: If learning SCHEDULE, IDEM will display its SCHEDULE plan, which needs modification. Step 3 can be modified.

Implementation: If the arguments of the command are PERSON and MEETING, IDEM notices a similarity with known commands. The known similar command is INVITE; if SCHEDULE has been taught, it is also similar. IDEM asks if it would be easier to modify a known command, and provides plans of each for the user to evaluate for potential modification. SCHEDULE can be modified to form NOTIFY.

This part of the demonstration has a narrow response path (i.e. is very

canned). It is intended to show instructability by analogy. The message "Bottom of Demonstration" indicates that IDEM's capabilities has been over-stepped.

4.5.5 LEARN RULES

IDEM learns rules and behaves as if it were an expert system proto-typing environment. When rules are entered, their consequences are shown for approval.

Implementation: IDEM can learn one rule, SEE-THE-DEMO-RULE. INSTRUCTABILITY-DEMO, HAS-SEEN, and SCHEDULE must have been taught previously. The input sequence:

a. Type: rule
b. If-part: IF a PERSON HAS-NOT-SEEN the INSTRUCTABILITY-DEMO
c. Then-part: THEN SCHEDULE the PERSON for it.

IDEM will show the consequences and ask if that is what was wanted.

Implementation: IDEM will ask if the scope of the command SCHEDULE should be expanded from MEETING to EVENT in order to include DEMONSTRATIONs.

5. LEARNABLE COMMANDS

IDEM is implemented to learn and be able to use only two commands: SCHEDULE and NOTIFY. This restriction is due to

a. a severely limited knowledge base,
b. a non-generalizable learning mechanism, and
c. pre-programmed, as opposed to dynamic, intelligence.

5.1 SCHEDULE

IDEM does not know how to SCHEDULE until it is taught. After teaching, it is able to perform the SCHEDULE function.

Implementation: Since SCHEDULE is actually predefined, a sequence of predetermined responses to IDEM's questions is necessary:

a. Type of concept: command
b. Context of process: calendar-world (irrelevant)
c. Number of arguments: 2
d. First argument: person
e. Second argument: meeting
f. Type of effect: side (irrelevant)
g. Effect: Put the time-of the meeting on the calendar-of the person.

h. Changes: step 3 (no choice)

i. Change step 3 to: Put the meeting in the calendar-day slot that matches the day-of the meeting.

5.2 NOTIFY

LEARN NOTIFY demonstrates a capability to modify previous functions. NOTIFY is intended to be a modification of the SCHEDULE plan.

Implementation: The response sequence to LEARN NOTIFY is the same as that of LEARN SCHEDULE up until step f (above). At that point IDEM will notice the similarity of arguments between NOTIFY and two other commands, INVITE and (if learned) SCHEDULE. The user may examine the plans of these commands and modify one of them to define NOTIFY.

Implementation: Only step 3 of SCHEDULE can be modified, to "Put the name-of the meeting in the messages of the person". IDEM will ask if day and time should be included, similar to SCHEDULE, and then do it.

6. THE PARSER

Simple English sentences are scanned for key-words that are either imperative verbs or descriptive nouns. Verbs are treated as commands; nouns as arguments.

Implementation: Keywords can be known functions, categories, relations, or instances. The parser looks for a command word that it recognizes, followed by a noun which may be known or unknown. If it is unknown, the command that is expected is LEARN. Most common pronouns, prepositions, and articles are skipped over.

7. THE DISPLAY WINDOWS

IDEM uses seven display windows:

1. The background frame "Interface Instructability Demonstration".

2. The interaction window "Current Task": displays "==>" at top-level and expects a command sequence. Control is then transferred to "Your Replies:".

3. The interaction window "Your Replies:": accepts response input from the user to IDEM's questions.

4. The display window "Needed Information:": displays questions from IDEM as it attempts to learn.

5. The display window "Completed Information:": displays IDEM's comments whenever it understands, completes a task, or is transferring control.

6. The display window "Additional Information:": displays ancillary questions and information.

7. The display window "Instructability Technique:": describes the technique of instructability currently being demonstrated.

8. DISCUSSION

8.1 USER FRIENDLINESS

IDEM attempts to make its processes and knowledge accessible to the user. It is willing to negotiate definitions and actions so that both the system and the user can arrive at a common perspective.

IDEM is unfriendly in that its processes are inherently mathematical. Violation of mathematical convention, such as having a single symbol with two different meanings, or having a process with an ambiguous result, is intolerable.

8.2 MATHEMATICAL PERSPECTIVE

IDEM's intelligence is determined by the mathematics of its knowledge. For instance, if the process

SCHEDULE <person> for <meeting>

is declared to be commutative (explicitly or by default), then IDEM would have the sense to do the correct thing when instructed to

SCHEDULE <meeting> on the agenda of <person>.

A mathematical perspective is the same thing as explicit definition of terms and intolerance of ambiguity. A lot of the mechanics of this perspective can be hidden from the user by incorporating default behaviors and coercion of user intent. The user can be informed (and to some extent trained) by displaying every call to a default value and every coercion from informal description to formal description. The negotiation between human informal forms of reference and machine formal forms is the heart of intelligent interface.

8.3 TECHNICAL OBSERVATIONS

IDEM demonstrates both achievable and potentially unachievable behaviors within the scope of a three-to-five man-year research effort.

8.3.1 WHAT IS ACHIEVABLE

The following capacities, although difficult, should be within the reach of a concerted effort:

1. The restricted English interface language.

2. A limited, user-friendly interface system with description and explanation facilities.

3. Program/plan synthesis and compilation that make instructability relatively efficient.

4. Knowledge and process management with intelligent assistance.

8.3.2 WHAT IS UNACHIEVABLE

1. IDEM cannot conceptualize and engineer the knowledge in a domain.

2. Traditional representation problems (the axiomatization of time, causality, common-sense knowledge, and spatial reasoning) will not magically disappear.

3. New primitives, concepts, or strategies will not develop automatically.

4. Instructability is not a solution to machine learning problems; it is a naturalistic programming interface with a lot of automated programming support.

8.4 A POSTERIORI

Some ideas that became clear after IDEM was implemented should be considered in future work:

1. An alternative to the functional perspective, with functions and arguments, is the objective perspective, with actors, factors, results and relationships.

2. "Learn models" would provide a nice extension up the hierarchy of abstraction, and could start to embody a perspective.

3. IDEM could reflect upon the input of the user and its own replies,

in order to learn a user model or a model of the interaction process.

4. IDEM's intelligence is overtly connected to the amount of its knowledge. The efficiency of its performance will depend on the quality of the engineering of the knowledge of a domain, not on machine learning questions.

5. Critique of frame-based knowledge representation:
   Pro: fast search
        handles ambiguity well
        supports mixed-initiative
   Con: time-consuming to instantiate
        needs different processing functions for different frames

6. Hayes-Roth on the sequence of learning activities:
   request information
   interpret
   operationalize
   integrate into existing knowledge structures
   evaluate

IDEM follows this pretty closely, omitting the evaluation step.

9. DEMONSTRATION SCENARIOS

The following scripts demonstrate the demonstration, without blowing it up. Patter is omitted. I have also avoided personifying IDEM's responses.

"I:" refers to IDEM output comments

"Reply:" and other forms of "<Label>:" are IDEM requests for information. What follows each request is typed in by the user.

9.1 SCENE I: LEARNING NEW INSTANCES, CATEGORIES AND RELATIONS

STORY: Assume a need to keep track of people who have seen this demo.

The relation HAS-SEEN is unknown to IDEM. So is the instance INSTRUCTABILITY-DEMO.

0. (==>) LEARN RULE1 (demonstrate initial ignorance at top level)

I: *question sequence* --

Type: Rule
If-part: IF <new-person> HAS-SEEN the INSTRUCTABILITY-DEMO

I: can't, don't know

<new-person>
HAS-SEEN
INSTRUCTABILITY-DEMO

1. (==>) First LEARN about <new-person>

I: *question sequence* --

Type: Instance
Category: Person (demonstrate typo recovery)
Category: ? (demonstrate TELL)

Tell about: Person
Tell about: Haskell

Category: Person
OK?: yes
Fill-slots: now

```
            Name: <any-atom>
            Sex: <any-atom>
            Interests: <any-atom>
            Calendar-of: call-to-calendar       (procedural attachment)

      I:  done


2.  (==>  ) TELL us about the category PERSON

      I: known-instances and defining slots


3.  (==>  ) Now I want you to LEARN about INSTRUCTABILITY-DEMO

      I:  question sequence --

      Type:  Instance
      Category:  X                  (demonstrate learn from examples)

      Request for examples:  training-demo haskells-demo

      I:  all match DEMONSTRATION

      Reply:  2, is a subset
      Name:  demos-that-learn
      Alter slots?:  no
      Fill slots:  later

      I: OK


4.  (==>  )  TELL us about EVENT

      I:  known-instances and defining slots
      I:  known-subcategories

      Reply: DEMONSTRATION

      I:  known-instances and defining slots
      I:  known-subcategories                   (includes newly acquired
info)

      Reply:  DEMOS-THAT-LEARN

      I:  known-instances and defining slots
```

5. (==>) LEARN the relation HAS-SEEN

I: *question sequence* --

Type: Relation
Arguments: 2
First argument: person
Second argument: instructability-demo

I: do you want specific second argument?

Reply: 3 (no)

I: shows hierarchy
I: change?

Reply: change to EVENT

I: different than these similar relations?

Reply: yes

I: properties of relation?

Reply: ?

I: shows and re-asks

Props: irreflexive

I: OK

6. (==>) STORE <new-person> HAS-SEEN the INSTRUCTABILITY-DEMO

I: OK

9.2 SCENE II: LEARNING NEW COMMANDS

STORY: Assume the context of CALENDAR-WORLD, and the desire to SCHEDULE.

Imagine a new person, <new-person>, whom you would like to schedule for the finance-meeting. You would also like to notify his boss, HASKELL, of the meeting.

Neither SCHEDULE or NOTIFY are known to IDEM. However, IDEM does know HASKELL, and just learned <new-person> from Scene I.

1. (==>) SCHEDULE <new-person> for the FINANCE-MEETING.

I: can't.

2. (==>) I'd like you to LEARN how to SCHEDULE.

I: *question sequence* --

Type: Command
Context: Calendar-world
Arguments: 2
First argument: Person
Second argument: Meeting
Effect type: Side

Describe effect: PUT the DAY-OF the MEETING on the CALENDAR-OF the PERSON.

Effect type: REMIND (demonstrate REMIND)

Effect type: X to exit effects

I: OK plan for SCHEDULE?

Reply: Decide to change Step 3 of SCHEDULE.

(demonstrate default planning, effect description ambiguity and incompleteness, need for AI support)

Modify: PUT the NAME-OF the MEETING in the CALENDAR-DAY slot that matches the DAY-OF the MEETING

(demonstrate assumed knowledge of slot structure of the calendar, ellipsis of "of the calendar-of the person")

I: learns SCHEDULE

3. (==>) SCHEDULE <new-person> for the FINANCE-MEETING

I: OK

4. (==>) Now please LEARN how to NOTIFY

I: *question sequence* --

Type: Command
Context: Calendar-world
Arguments: 2
First argument: Person
Second argument: Meeting

I: Do you want to modify?

Reply: Yes

Look at: Invite
Look at: Schedule

Modify: Schedule step 3
Change to: PUT the NAME-OF the MEETING in the MESSAGES of the PERSON

I: Include time and day?

Reply: Yes

I: OK

5. (==>) NOTIFY HASKELL

I: of what?

What: the finance-meeting

I: OK

9.3 SCENE III: LEARN NEW RULES

Assume we want to teach IDEM this rule:

```
      IF <person> HAS-NOT-SEEN the INSTRUCTABILITY-DEMO,
            THEN SCHEDULE him for it.
```

```
1.  (==>  )  LEARN SEE-THE-DEMO-RULE

      I:  question sequence --

      Type:  Rule
      If-part:  IF a PERSON HAS-NOT-SEEN the INSTRUCTABILITY-DEMO

      I:  Interprets as
            (NOT (HAS-SEEN (LOOP OVER PERSON) INSTRUCTABILITY-DEMO))

            OK?

      Reply:  Yes

      Then-part:  THEN SCHEDULE the PERSON for it.

      I:  Interprets as
            (SCHEDULE (LOOP OVER PERSON) INSTRUCTABILITY-DEMO)

            OK?

      Reply:  Yes

      I:  Expand the scope of the second argument of SCHEDULE?

      Reply:  yes, to EVENT.

      I:  OK, shows forward-chained rule.

            OK?

      Reply:  Yes

      I:  Incorporates rule
```

9.4 SCENE IV: A COMPLEX COMMAND

NOTE: This is a conceptual demo; IDEM cannot do it because it lacks all the assumed knowledge.

Assume we want a command that sets up a presentation, ARRANGE.

Assume domain knowledge:

Categories of people:
available-speakers
staff-members
company-distribution-list

Category of event:
presentation

Relation:
<speaker> accepts <invitation>

Expanded command:
SCHEDULE <room> for an <event>, reserves a room by putting the time-of the event on the matching calendar-day of the room

Compose effects for ARRANGE:

1. LEARN INSTANCE: THIS-MONTHS-PRESENTATION
2. LOOP until <speaker> accepts <invitation>
   INVITE the <speaker> to THIS-MONTHS-PRESENTATION
3. SCHEDULE the staff-members for THIS-MONTHS-PRESENTATION
4. SCHEDULE the room for THIS-MONTHS-PRESENTATION
5. NOTIFY the company-distribution-list of THIS-MONTHS-PRESENTATION

Result:

==> ARRANGE THIS-MONTHS-PRESENTATION

I: Date?:
Place?:
Speaker?:

IDEM takes care of the technicalities, and notifies you if room or speaker need to be changed.