COURSE INFORMATION

Class structure:

I would like to organize this class as a practicum/workshop. The class will decide upon a group project and develop a software product by the end of the quarter. The goals of this class are

1) to develop an HCI interactive software product,
2) to learn to manage a group programming process,
3) to understand the technical details of developing software for HCI.

The project orientation is similar to the capstone project, except that we will focus on designing and writing software, and *not* on requirements, statement of work, project management, scheduling, and other planning documents. We will use whatever software tools are necessary to achieve our goals (Java, XML, C, Common LISP Interface Manager, Visual Basic, etc).

Evaluation:

Available grades:

non-completion: Incomplete, Withdraw, etc.

completion: A A- B+ B B- C

A:	reserved for superior performance
A- or B+:	expected grade for conscientious performance
B:	adequate work
B-:	barely adequate
C:	equivalent to failing

Grading Options:

1. Grading Contract: specify a set of behaviors and an associated grade.
2. Performance Quality: attendance, participation, assigned exercises
3. Self-determined: negotiate with instructor

Discussion:

If you already understand the field, if you plan to excel, or if you need clear performance goals for motivation, then **Option 1** is a good idea. If you prefer a clearly defined agenda, if you do well with concrete task assignments, or if you need a schedule of activities for motivation, then **Option 2** is a good idea. If you are not concerned about grades, if you intend to do what you choose anyway, or if you are self-motivated, then **Option 3** is a good idea.

I will notify any student who is not on a trajectory for personal success.

SOME PROJECT IDEAS

1. **Virtual University Course**
   Develop a short course to be presented on the web. A clever twist would be to develop this course as a web-based remote course.

2. **Web-page Refinement**
   Select a web-site from a moderately large company. Review and critique their design and performance. Then rebuild the entire site to correct the identified problems. A clever grounding for this work would be to send our results to the company.

3. **Internet Software Tool**
   Select, design, and implement a software tool which facilitates some aspect of web-interaction. Ideas may include intelligent agents, search engines, web-page builders, site mapping and layout, download analysis, etc.

4. **Customized GUI Interface**
   Evolve an existing interface toolkit with some customized refinements, such as a new type of widget, customization tools, dialog management, or interaction traking.

5. **Virtual Reality System**
   Develop existing C code for parts of a VR system, including 3D interaction devices, smart terrain, multiple participants in a single environment, new virtual bodies, etc.

6. **Finite State Machine Emulators**
   Software to emulate an interactive system, such as an ATM, a soda machine, a telephone answering system, airline ticket booking, etc.

7. **Java-based Gaming**
   Implement a fun game for the internet. Could be a version of a standard adventure and fighting game, or a strategy game such as Diplomacy or Stratego, or a graphics/visual game such as Life or Centipede.

8. **Mathematics Visualization**
   Develop a visual interface to some abstract mathematical structure such as an N-dimensional cube, 3D knots, or Fractals

9. **Spatial Arithmetic and Algebra**
   Develop a graphic interface for 7th grade math, using manipulative structures rather than equations and symbols.

10. **Manipulable Logic**
    Develop an interactive interface for Boundary Mathematics

11. **Innovation Prototyping**
    Select a yet-to-be commercialized application, such as wearable computers or TV-wristwatchs, and develop a prototype functional design for the interface.

12. **Ideas Provided by the Class**

PROJECT ORGANIZATION

Our class project will be **Knowledge-based Hyperlinks (KBHL)**. (The name is suggestive, not final.) The general idea is to develop a demonstration example (simulation of) traversing a network of hyperlinks based on content or semantic information, rather than on syntactic structures such as keywords.

The component tasks/roles for this project include:

0. Research: have others tried this approach? What did they learn? What topics partially address our project (eg: knowledge engineering, hypertext, web search, expert systems, interactive interface, software agents, etc.)?

1. Develop a sample database of content-carrying web-pages, with hyperlinks across various content components.

2. Develop a knowledge-based data-structure which attaches to each hyperlink. The knowledge-base will contain conventional expert system-like assertions of facts and relations.

3. Develop an inference strategy for traversing the knowledge of various hyperlinks. This may include pattern-matching, Baysian probabilities, various types of inference, and other semantic-like structures. See below for more ideas.

4. Develop an interaction plan which permits the user to understand and traverse knowledge-based links.

5. Develop an interface prototype for using KBHLs. The interface should require minimal learning, and have a "natural" feel.

6. Discuss and roughly design extensions to the KBHL, including user-extensibility, automated documentation, and software agency.

Discussion

The content web-pages require careful selection and design, to maximize the (apparent) utility of the KBHL tool. Research from Ontology Engineering (ie what the folks at Yahoo do to organize their weblinks) will guide this effort.

The available types of *intelligent traversal* require careful design, so that 1) the tool will be useful for finding information, and 2) the tool will be understandable to normal folks.

How semantics is captured and accessed is of critical importance. How do we know what the user is looking for? How will users be able to say what they are looking for? What types of intelligent traversal are useful?

A given links will usually contain may intelligent branches. How will the user know which to select? That is, the organization of information is not only, or even necessarily, logical.

What type(s) of organizational structure do we what our smart links to expose? Inferential techniques address implicit or embedded information. Other types of smart links expose different structures. For example,

a *refinement link* leads to more detail on a topic.
a *classification link* provides a property inheritance context.
a *chronological link* tells you what happened before or after.
a *spatial link* navigates through locations.
a *dependency link* identified prerequisites, requirements and causal structures.
a *structure link* decomposes an object into its component parts.
a *decision link* traces choices and their consequences.
an *analogy link* identifies thnig that are similar but not necessarily related.

Our problem maps onto a classic graph problem: what kind of nodes and vertices make sense? How many types of links can be used at one time? Should nodes or links provide consistency?

Observations

The simulation web-pages need to accurately reflect the data-structures underlying actual web-pages. We will need to figure out how additional information can be easily and portably attached to links.

Specialized types of inference are needed for different fields of knowledge. Only some kinds of knowledge are reducable to knowledge-based encoding.

Knowledge may not be about "content", it can also be about structure (the form of the link), about possibilities, about grouping, about proximity, etc.

We will need to show *critical functionality*. What does our tool do that other tools do not do? How is the advantage measured? Where are the strong. weak, and failure points?

There may be no solution for information overload. We can be overwhelmed by too many windows, by too many nodes and links, by too much scrolling. Perhaps links should filter and refine rather than enhance access.

Understandable structure may need to be designed and written into the website itself, rather than put into links.

Techniques for structuring and filtering:

Labeling: clear, concise labels and concepts
Chunking: relatively small, related hunks of information
Relevance: all information pertains to the content of the page or the goal of the user
Consistency: similar items are treated in similar ways

Hyperlinks may simply increase the desire for better content structure and more efficient linking models. That is, smart links may expose the greater weaknesses of hypertext systems.

Bottom line is that the information itself must have a structure for a smart link to expose.

COURSE SYLLABUS

This is a project-oriented course. All grades will be based on successful completion of the class project. A single group grade is recommended. The following schedule is very likely to change, and is intended as a general guideline for the pace and timing of the class.

Class Meeting	Topic
1)	Introduction
2)	Project discussion
3)	Project topic refinement, hypertext
4)	Project finalization and planning
5)	knowledge representation, modeling
6)	design review
7)	final design decisions
8)	content development
9)	coding specification
10)	interactivity simulation
11)	possible class holiday
12)	possible class holiday
13)	review of all components
14)	closure of iteration 1
15)	integration issues
16)	closure of iteration 2, integration
17)	revisions and refinements
18)	closure of iteration 3
19)	simulation of complete system
20)	discussion and evaluation

INFORMATION MAPPING (R. Horn)

Paper metaphors for hypertext

- library card catalogues
- footnotes
- cross-reference
- sticky notes
- commentaries
- indexes
- quotes
- anthologies

Computer metaphors for hypertext

- linked note cards
- popup notes
- linked screens or windows
- stretch text and outlines
- semantic nets
- branching stories
- relational databases
- simulations

Hypertext Links

- *system-supplied*
  - command and control pathways
  - table of contents
  - history tracking
  - automated profiling
- *user-created*
  - detours and shortcuts
  - notes, commentary, reminders
  - analogical links
  - new text
  - links to other knowledge bases
- *author-created*
  - links to prerequisite knowledge
  - hierarchical classification
  - chronological structures

Kinds of links

hierarchical	building a tree
keyword	building an array
referential	building a pointer list
cluster	building a struct

Wayfinding in cyberspace (these don't work very well)

show all connections
go back to the beginning
show history of behavior

Node sizes

one sentence
text of arbitrary size (article, monograph)
index card size
screen size
scroll of any length
variable record sizing
variable size, precisely and flexibly chunked

Information types

structure
concept
procedure
process
classification
principle
fact

Information Blocks

chunking	small, manageable hunks (blocks, maps)
relevance	one main point per chunk, based on purpose or function to reader
consistency	similar words, labels, formats, organization
labeling	label every chunk based on specific criteria

Common types of information blocks

analogy
block diagram
checklist
classification table
classification tree
comment
cycle chart
decision table
definition
description
diagram
example
fact
flow chart
flow diagram
formula
input-procedure-output
non-example
notation
objectives
outlines
parts-function table
parts table
prerequisite
principle
procedure table
purpose
rule
stage
synonym
theorem
when to use
worksheet

Types of hypertrail, path

- prerequisite
- classification
- chronological
  - sequence of events
  - storyline
  - natural development
- geographic
- project
- structural
- decision
- definition
- example

How readers behave

- novices stop reading too soon
- novices are mislead by superficial features
- novices rarely seek non-linear information
- readers construct a hierarchical mental representation
- readers remember the top level of information better
- readers depend on repetition of keywords

Reading cues

- hierarchical text organization
- explicit transitions
- sequence signals
- contrast and similarity cues
- pronouns as cohesiveness cues
- metaphors
- content schemas

Document titles

- just right: not too general, too specific, too long, too short
- common language for the intended audience
- itemize all possible readers and use lowest common denominator
- no cuteness or silliness
- no vague, mislabeled topic headers
- same words in contents, titles, pages, and references

FORMAL KNOWLEDGE

A **conceptual model** consists of
discrete objects, presumed to exist: the *Universe of Discourse*
interrelations between objects
functions: compound names for objects and for unnamed objects
relations: truth statements about objects

No matter how the world is conceptualized, there are other conceptualizations that are just as useful.

Declarative Style

An **knowledge-based program** consists of
a set of objects
a set of functions (names for compound objects)
a set of relations (facts)
a set of permissible transformations

State Space

The collection of facts (the database) at one given time defines the **state** of the world.
All possible state configurations define the **state space**.
To move from one state to another, apply a permitted **transformation rule**.
The state space and the moves between states from a **graph**.
Algorithms explore/search the state space.
Programmers control the search.

Knowledge Representation Labels

Constants:

names of specific objects: John, Tuesday, My-Phone-Number
names of specific functions: House-of[x], Phone-of[x], Truth-of[p]
names of specific relations: Likes[Mary, Tom], Phone-Number[Tom, x]

Variables:

refer to sets/classes/domains of objects
always scoped/introduced by a quantifier

Knowledge Representation Atoms

Named objects	(object constants)
Indirect/compound named objects	(functions)
Relations between objects	(facts)

Logical connectives (and, or, not, if, iff) connect atoms. They cannot be used inside atoms.

yes: eyes-of[John] AND hair-of[John}
no: (eyes-of AND hair-of)[John]
no: hair-of[John AND Mary]
yes: hair-of[John] AND hair-of[Mary]

Example of a RELATIONAL KNOWLEDGE-BASE

Part of a knowledge-base about family relationships.

Vocabulary:

```
(father X Y)
(mother X Y)
(male Y)
(female Y)
(parent X Y)
(sibling X Y)
(brother X Y)
(sister X Y)
(uncle X Y)
(aunt X Y)
(gfather X Y)
(gmother X Y)
(ancestor X Y)
(cousin X Y)
```

Knowledge Base:

```
(if (father A B) (parent A B))
(if (mother A B) (parent A B))
(if (and (parent A C) (parent A B) (not (= B C))) (sibling B C))
(if (and (sibling A B) (male A)) (brother A B))
(if (and (sibling A B) (female A)) (sister A B))
(if (and (parent B C) (brother A B)) (uncle A C))
(if (and (parent B C) (sister A B)) (aunt A C))
(if (and (parent B C) (father A B)) (gfather A C))
(if (and (parent B C) (mother A B)) (gmother A C))
(if (parent A B) (ancestor A B))
(if (and (parent A B) (ancestor B C)) (ancestor A C))
(if (and (parent A C) (parent B D) (sibling A B)) (cousin C D))
(if (father A B) (male A))
(if (mother A B) (female A))
```

Facts:

```
(father arthur bertram)
(father arthur bailey)
(father bertram cornish)
(father bertram carey)
(mother beatrice cornish)
(mother beatrice carey)
```

(father bailey carleton)
(father bailey cassandra)
(mother bessie carleton)
(mother bessie cassandra)
(male cornish)
(male carey)
(male carleton)
(female cassandra)

Example questions:
(gfather arthur ?)
(cousin ? cassandra)

Technical Difficulties in Modeling and Knowledge Representaiton

(using blocks world in LISP as an example)

1. What is important to describe?
   Build little theories of little worlds.
   (Block A) (OnTable A) (Hand Empty)

2. How should descriptions be partitioned?
   Functions or Relations, special or general objects?
   (OnTable A) (On A Table) (not (OnTable Table))

3. How do we talk about groups and classes of objects?
   Quantification and abstraction
   (All (x) (Block x))

4. How do we address things with no names?
   Functions as indirect, compound names.
   (House-of John)

5. How do we handle things with more than one name?
   unique name hypothesis, unification
   (Uncle John) = (Brother (Father John)) = Bob

6. How do we make general rules which define the structure of relations?
   quantification
   (All (x) (iff (Uncle x) (Brother (Father x))))

7. How are typing and filters on domains represented?
   predicates in conjunction
   (All (x) (and (Person x) (Father x y)))

8. How do we join more than one fact?
   conjunction
   (and (F x) (G x))

9. How do we compute with logic?
   inference as natural deduction and as resolution
   (if (and (P x) (if (P x) (Q x))) (Q x))

10. How do we compute with quantifiers and classes of objects?
    implicit universal quantification, Skolemization
    (Exist (x) (P x)) ==> (P (Sk-1 x))

11. What is the difference between a fact and a query?
    query combination rules

    A. conjunction with negated query
       (and (P x) (not (Q ?)))

    B. Skolemization of query variables
       (Q ?) ==> (Q Sk-1)

    C. Facts imply Query
       (if (P x) (Q ?))

    D. The answer predicate
       (if (P x) (Answer x))

12. What kinds of rules do we need for query answering?

    A. definitions
       (iff (P x) (Q (R x)))

    B. mathematical structures (symmetry, transitivity, etc)
       (if (and (if (P x) (Q x)) (if (Q x) (R x))) (R x))

    C. permissible state transformations
       (Pick-up x) = (Assert (not (onTable x)))

13. How can we control the inference/search procedure?

    A. Pre and Post conditions
    B. Compound queries
    C. Searching databases of rules and facts

14. How do we steer the resolution process?

    A. set of support
    B. ordered resoultion
    C. static vs dynamic approaches (compiled vs run-time)
    D. lookahead, cheapest first, dependency directed search

15. How do we express meta-level reasoning (rules about rules)
    measure the savings vs brute force

LECTURE NOTES

Design

Conceptual: requirements, system expectation, needed information
Physical: how to achieve objectives

Requirements

Functional: what the interface must do
Data: what needs to be available for processing
Usability: user performance and satisfaction

System Models

Dataflow: data the passes between processes
- rectangle: source or destination of data
- circle: process which transforms data
- named link: transacted data
- bucket: database or store

Entity Relationship (ER)
- entities: aggregate of data elements with a meaning
- attributes: specific types of data
- relationships: connections between entities

User Interface "Programming" Tools

command processors, scripting languages (SQL, UNIX shell, HTML)
menu systems (Mac, Windows)
form fill-in systems (Netscape, databases)
user interface toolkits (SUIT, NeXTStep, Visual Basic)
window managers (spreadsheets, MacOS, Win95)
user interface management systems (CLIM, JAVA)

Decision Types

structural: end user's conceptual model
functional: user actions and operations
dialog: content and sequence of information exchange
- semantics, units of meaning
- messages, units of content
- sequences, flow of content

presentation: interaction objects and processes (widgets)
pragmatic: use of hardware and physical space

Desirable Properties of a Conceptual and Implementation Model

sufficiency: all the needed information
necessity: only the needed information
understandability: easy to learn, easy to use
independence: modify constructs with minimal interaction
reusability: generic and general
consistency: same activity in same manner
minimality: no overlapping definitions and actions
orthogonality: each object accomplishes a different objective
compatibility: all models use similar concepts
implementability: easy to build

Usability requirements

learnability: time and effort to reach a level of proficiency
throughput: speed of execution and number of errors
flexibility: accommodation to changes in task and environment
attitude: satisfaction and acceptance

Task analysis techniques

Goals, tasks, actions
Hierarchical task analysis
Goals, operations, methods, selection rules (GOMS)
Task, semantic, syntactic, interaction

Usability testing techniques

direct observation
indirect observation (video recording)
verbal protocols (thinking aloud)
software logs
interviews (structured or flexible)
questionnaires
checklist, rating, semantic differential, ranking

Potential measurement criteria

time to complete task
percentage of task completed
speed (percentage of task per unit time)
ratio of success to failure

time spent on errors
number of commands used
frequency of use of help or documentation
time spent using help
percentage of favorable or unfavorable user comments
number of repetitions of failed commands
number of runs of success or failure
number of times the interface misleads the user
number of good and bad features recalled by users
number of available commands not invoked
number of regressive behaviors
number fo users choosing or preferring system
number of times users have to work around a problem
number of times user is disrupted from task
number of times user loses control of system
number of times user expressed frustration or satisfaction

Notes on the Programming Language JAVA

Features

- simple
- object-oriented (relatively pure oo, not procedural + oo extensions)
- distributed
- both interpreted and compiled instruction sets
- robust
- secure
- architecture neutral
- portable
- high-performance
- multi-threaded
- dynamic

Object Orientation

- class = abstraction
  - class variables
  - class functions
- instance
  - fields are instance variables
  - methods are functions
- hierarchy
- subclasses = design by difference
- inheritance
- overloading
- constructors
- accessors
- encapsulation (public, package, protected, private)

Implementation Features

- virtual machine
  - byte-code = machine instructions for a virtual machine (VM)
  - VM maps closely to most native hardware machine
- call-by-value parameter passing (compare to call-by-name, call-by-need)
  - the value of an object is its reference
  - copies binding into parameter field of method
- automatic garbage collection
- streams
- type-safe references (strong typing)
- exception handling
- multiple threads (multitasking, lightweight)
- simultaneous processes and shared objects
  - locks; user provided deadlock avoidance
  - automatic switching, scheduling, synchronization

Language Features

base data-types are not objects
first-class strings, read-only
international Unicode character set
first-class exceptions, checked by compiler
HTML inline interface
first-class network interface (URL, TCP, sockets)
protection and security model
class Object is root
interface concept for limited multiple inheritance

no pointers (use references instead)
no global variables (use root classes)
no goto (use catch/throw and labels)
no operator overloading (static basic operators)
no delete

Language Keyword Features

final:	constants, unforgable classes, non-overridden methods
this:	reference to self object
new:	constructs a new object or class
. :	accessor function
[ ] :	arrays
{ } :	sequential block
super:	references things from the superclass(es)
try-catch-finally:	exception handling
labelled break:	for skipping sequences and exiting loops

Packages

class libraries
functionality groups
user interface code provided
user provide application specific abstract data types

Provided Java API Packages

java.lang	the language
java.net	networking
java.io	streams and files
java.util	utilities, higher-order data-structures (enumeration, vector, stack, dictionary, hashtable)
java.awt	Abstract Window Toolkit
java.awt.image	image processing
java.awt.peer	interface with native interfaces
java.applet	basic applets

plus plenty more on the net and by vendors

Interfaces

unique in Java
separate design inheritance from implementation inheritance
can inherit a contract without inheriting an implementation
tie together dissimilar classes for object reference
subclasses provide code for all interface methods
multiple inheritance (classes can implement multiple interfaces)
no root, does not default to Object root-class
constrained to:
- abstract class (no instances, only subclasses)
- no code, only abstract method declarations
- static and final variables
- public methods

Exceptions

catch and throw handlers
programmer declared compile-time errors
cleanly checks for errors without cluttering code
try/catch/throw environment
finally clean-up

Protection

runtime system does not permit memory access

public	full access by all classes
package	access by classes in common library
protected	access by subclasses only
private	no access by other classes

Streams

usually paired as InputStream, OutputStream
Piped, Filter, Buffered
StreamTokenizer

System Programming Classes

Runtime	(state of Java at runtime)
Process	(running java process)
System	(state of environment)
Math	(standard computations)
Native	(foreign function interface)

Multimedia

MediaTracker	image maintenance
Sound	AudioClip
Animation	sprites

Abstract Window Toolkit (AWT)

- embedding within the local browser
- standard component set
  - button, checkbox, choice, label, list
  - scrollbar, textarea, textfield,
  - windows, menus, dialog boxes
- containers
  - graphical collections of components
- layout management
- event handling
  - mouse clicks and movements
  - keyboard
- graphics
  - drawing, color, fonts, clipping, image handling

Sample HTML Applet Call

```
<HTML>
<HEAD>
 <TITLE>Applet Page</TITLE>
</HEAD>
<BODY>
<H4>This is an example of a Java applet:</H4>
<HR> <APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=50> </APPLET> <HR>
</BODY>
</HTML>
```

Sample Applet

```
import java.applet.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet
        {public void paint(Graphics g)
                { g.drawString("Hello world.", 5, 10); } }
```

Web Resources (1997)

http://java.sun.com/	...from the Source
http://www.rpi.edu/~decemj/works/java.html/	a Java book author
http://www.gamelan.com/	registry of programs
http://sunsite.unc.edu/javafaq/javafaq.html	FAQs
http://www.well.com/user/yimmit/	links to resources
http://www.natural.com/	major developer
http://www.io.org/~mentor/J__Notes.html	more resources
http://www.acm.org/~ops/java.html	ACM resources
http://www.yahoo.com/Computers/Languages/Java/	search engine resources
http://rendezvous.com/Java/hierarchy	class diagrams

A Complete User Interface System

Primary Examples:
MacOS, Visual Basic, NeXTStep, Java, Common Lisp Interface Manager

- **windowing abstraction**
  containers, views

- **display components**
  button, checkbox, choice box, label, list, table,
  scrollbar, textarea, textfield, window, menu, dialog box

- **display tools**
  fonts and points
  color
  graphics system (drawing, clipping, 3D)
  image handling
  layout management

- **temporal data tools**
  time and synchronization model
  sound manager
  video manager
  animation manager

- **interactivity tools**
  event handling and management (mouse, keyboard, arbitraty input devices)
  streams and buffers
  scripting language

- **programming interface tools**
  object-oriented class, instance, and message system (initialize-, make-)
  load, compile, link, and evaluate
  language-specific text editor
  interface construction toolkit
  debugging and exception handling
  namespaces and packages
  foreign function interface

- **operating system tools**
  threads and multitasking
  concurrency, switching, scheduling, and synchronization
  memory management
  file system interface
  network interface and security
  low level: internal data structures, pointers, memory blocks, traps

Extended Examples (Java, CLIM)
using widget interactions as a simple example

Generic object operators/functions:

constructors: make-, initialize-, set-
assessors: get-
queries: ? -
functions: act-on-
relations: constrain-

Turnkey dialog boxes

throw-cancel and catch-cancel <aborts>
message-dialog
yes-or-no-dialog
get-string-from-user-dialog
select-item-from-list-dialog

Windows

nested-views, size, position, scroller, click-handler
title, font, color, active?, layer, zoom, grow, drag

Mac Common Lisp Menu Class structure

menu-element
- menubar (class, variable, function)
  - set-menubar
  - find-menu
  - <color-functions>
  - *default-menubar*
- menu
  - initialize-, set-
    - menu-title, menu-items, menu-colors
    - update-function
    - help-spec (balloon-help system)
  - install, deinstall, installed?
  - enable, disable, enabled?
  - font-style, <color-functions>
  - add-menu-item, remove-menu-item, get-menu-item, find-menu-item
  - menu-item
    - initialize-, set-, get-, query?-
      - owner, title
      - command-key, checked
      - action, action-function (call vs get)
      - disabled?
      - colors, font-style
      - update-function, help-spec
  - window-menu-item
    - close, save, save-as, save-copy-as, revert, hardcopy
    - cut, copy, paste, clear, select-all, undo, undo-more
    - load/evaluate-selection, load/evaluate-whole-buffer

Mac Common Lisp Dialog-items

initialize-, set-, get-, make-
view-size, view-container, view-position, view-nickname, view-font
dialog-item-text, dialog-item-handle, dialog-item-enabled?
part-color-list, dialog-item-action, help-spec, window-pointer
install, activate, activate-event-handler, default

button-dialog-item
press-button, default-button-dialog-item (make-, get-, set-, ?-)
static-text-dialog-item
editable-text-dialog-item
<key-stroke-handlers>
check-box-dialog-item (check-box-check, -uncheck, -checked?)
radio-button-dialog-item (radio-button-cluster, -push, -unpush, -pushed?)
table-dialog-item
<table-constructors>, <cell-contents-handlers>, sequence-dialog-item
pop-up-menu (<handlers>)
scroll-bar (<handlers>)

Interface Toolkit

The toolkit provides a drag-and-drop interface for constructing display interfaces. After selecting and positioning the interface, the toolkit writes the appropriate source code for that interface. Toolkit components:

Menubar Editor
Add Menu
Add Menu Item
Command key, Disabled, Check Mark
Menu Item Action (provide function), Menu Item Colors
Menu Colors
Print Menu Source
Rotate Menubars
Add New Menubar
Delete Menubar
Menubar Colors
Print Menubar Source
Use Dialogs (toggle with Design Dialog)
Design Dialogs
Document
Document with Grow
Document with Zoom
Tool (with title bar and close button)
Single Edge Box
Double Edge Box
Shadow Edge Box
Design Dialog Methods
Include Close Box
Color Window
Add Dialog Item

Static Text
Editable Text Field (Allow Returns, Allow Tabs, Draw Outline)
Button (Default Button)
Radio Button (Radio Button Pushed, Set Item Cluster)
Checkbox (Checkbox Checked)
Table (Set Cell Size, Horizontal Scroll Bar, Vertical Scroll Bar
Set Table Sequence, Set Wrap Length, Orientation)

Add Dialog Item Methods
Dialog Item Text
Enabled/Disabled
Set Item Action
Set Item Font
Set item Name
Set Color
Print Item Source

New Dialog
Add Horizontal Guide (for alignment during editing)
Add Vertical Guide
Edit Dialog
Print Dialog Source

Java Code for constructing some widgets

Named Button:

```
public void okButton() {
        Button b = new button("OK");
        add(b);  }
```

Unnamed button:

```
add(new Button("OK"))
```

Label:

```
add(new Label("Look at me"))
```

Checkbox:

```
add(new Checkbox("Check here if hungry"))
```

Checkbox Methods:
getLabel(), setLabel(String), getState(), setState(boolean)

Choice Menu:

```
{Choice myClassesMenu = new Choice;
  myClassesMenu.addItem("SE101");
  myClassesMenu.addItem("SE561");
  myClassesMenu.addItem("Special Project");
  add(myClassesMenu); }
```

Choice Menu Methods:
getItem(int), countItems(), getSelectedIndex(),
getSelectedItem, select(int), select(String)

PRODUCTION LISP CODE for a WINDOWING SYSTEM

Unedited, little documentation, good style.
This code is what you would have to write if you were developing an application windowing system without a toolkit or a class library.
Redundant code templates are omitted.

First the **class structure** for the windowing environment,
next the **menu system** with its corresponding **action functions**,
then the **control panel** with its corresponding action functions,
finally the **event handler** for text entry into the control window.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PARENT-WINDOW

(defclass parent-window (window)
  ((children :accessor children :initarg :children :initform nil)
   (common-data :accessor common-data :initarg :common-data :initform nil)))

(defmethod initialize-instance ((self parent-window) &rest rest)
  (apply #'call-next-method self rest)
  (map-children self #'set-child-parent self))

(defmethod find-parent-child ((self parent-window) type)
  (car (member type (children self) :key #'type)))

(defmethod add-parent-children ((self parent-window) &rest children)
  (setf (children self) (append children (children self))))

(defmethod remove-parent-children ((self parent-window) &rest children)
  (setf (children self) (set-difference (children self) children)))

(defmethod parent-children ((self parent-window) &rest children)
  (apply #'add-parent-children self children)
  (mapcar #'(lambda (child) (set-child-parent child self)) children))

(defmethod map-children ((self parent-window) func &rest args)
  (mapcar #'(lambda (child) (apply func child args)) (children self)))

(defmethod open-child ((self parent-window) type &rest rest)
  (cond
   ((eq type 'entry) self)
   ((find-parent-child self type))
   ((eq type 'database)
    (apply #'make-instance 'database-window :parent self rest))
   (T
    (apply #'make-instance 'display-window
           :type type :parent self rest))))
```

```
(defmethod window-close ((self parent-window))
  (call-next-method self)
  (map-children self #'window-close))

(defmethod set-window-title ((self parent-window) new-title)
  (map-children self #'set-window-title new-title)
  (call-next-method self new-title))

;;;six window subclasses and methods omitted here

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; DISPLAY-WINDOW

(defclass display-window (child-window)
  ((display-view :accessor display-view :initform nil)
   (title :accessor title :initform "Display"))
  (:default-initargs
    :window-type :document-with-zoom
    :view-font '("Monaco" 9)
    :view-size #@(300 150) ))

(defmethod initialize-instance
      ((self display-window) &rest rest &key (type 'display-view))
  (declare (dynamic-extent rest))
  (apply #'call-next-method self :type type rest)
  (let ((view (make-instance type
                :view-container self
                :view-size (subtract-points (view-size self) #@(15 15))
                :view-position #@(0 0)
                :draw-scroller-outline nil)))
    (setf (display-view self) view)
    (setf (title self) (title view))
    (when (parent self)
      (set-window-title self (window-title (parent self))))
    (mapcar #'(lambda (x) (setf (scroll-bar-scroll-size x) 12))
            (view-scroll-bars view))
    (set-common-data view (common-data self))))

(defmethod set-view-size ((self display-window) h &optional v)
  (declare (ignore h v))
  (without-interrupts
   (call-next-method)
   (let* ((new-size (subtract-points (view-size self) #@(15 15))))
       (set-view-size (display-view self) new-size))))

(defmethod window-zoom-event-handler ((self display-window) message)
  (declare (ignore message))
  (without-interrupts
   (call-next-method)
   (let* ((new-size (subtract-points (view-size self) #@(15 15))))
       (set-view-size (display-view self) new-size))))

(defmethod clear ((self display-window))
  (call-next-method self))
```

```
(defmethod save-to-eval ((self display-window))
  `(make-instance 'display-window
     :type ',(type self)
     :window-title ,(window-title self)
     :view-position ,(view-position self)
     :view-size ,(view-size self)  ))

(defmethod window-close ((self display-window))
  (when (parent self)
    (remove-parent-children (parent self) self))
 (call-next-method self))

(defun make-trace-output-window (parent)
  (make-instance 'display-window
    :type 'trace
    :parent parent
    :close-box-p    nil
    :window-title   "Trace Output Window"  ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; LOSP-MENU

(defvar *losp-menu* nil)
(defvar *db-edit-menu* nil)

(defun initialize-losp-menu ()
  (menu-install (setq *losp-menu* (make-losp-menu)))
  (menu-install (setq *db-edit-menu* (make-db-edit-menu))))

(defun make-losp-menu ()
  (MAKE-INSTANCE 'MENU
    :MENU-TITLE  "Losp"
    :MENU-ITEMS
    (LIST (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "About..."
            :MENU-ITEM-ACTION   #'make-losp-ABOUT-WINDOW)
          ;(MAKE-INSTANCE 'MENU-ITEM
          ;  :MENU-ITEM-TITLE    "Load"
          ;  :MENU-ITEM-ACTION   #'menu-load-losp)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Entry Window"
            :MENU-ITEM-ACTION   #'menu-make-entry-window)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Control Panel"
            :MENU-ITEM-ACTION   #'make-losp-CONTROL-PANEL
            :COMMAND-KEY        #\=
            :MENU-ITEM-CHECKED  nil)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Test Minimizer"
            :MENU-ITEM-ACTION   #'run-min-test)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Quit Losp"
            :MENU-ITEM-ACTION   #'close-LOSP
            :command-key        #\Q)))        )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; MENU ACTION FUNCTIONS
;;;
;;;   Menu items:             Activation function:
;;;     About...                 make-losp-about-window
;;;     Load                     menu-load-losp    [in initialize file]
;;;     Entry Window             make-entry-window
;;;     Control Panel            make-losp-control-panel
;;;     Test Minimizer           run-min-test
;;;     Quit Losp                close-losp

(defun menu-make-entry-window ()
  (setq *current-entry-window* (make-entry-window))
  ;(make-losp-control-panel))

(defun close-losp ()
  (if *current-entry-window* (window-close *current-entry-window*))
  (menu-deinstall *db-edit-menu*)
  (menu-deinstall *losp-menu*))

;;;several menu functions omitted here

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ABOUT-LOSP

(defun make-losp-about-window ()
  (modal-dialog
   (MAKE-INSTANCE 'COLOR-DIALOG
     :WINDOW-TYPE      :DOUBLE-EDGE-BOX
     :WINDOW-TITLE     "about-losp"
     :VIEW-POSITION    #@(426 60)
     :VIEW-SIZE        #@(370 185)
     :CLOSE-BOX-P      NIL
     :VIEW-FONT        '("Chicago" 12 :SRCOR :PLAIN)
     :VIEW-SUBVIEWS
     (LIST (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(58 8)  #@(260 16)
            "Losp Boolean Minimization Engine 1.0"   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(146 31)  #@(73 16)
            "May 1995"   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(8 58)  #@(345 32)
            "Copyright  (C)   1995,    OZ...International, Ltd.  and Interval
Research Corporation, All Rights Reserved."   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(20 102)  #@(345 16)
            "Authored by William Bricken and Jeffrey James."   'NIL)
           (MAKE-DIALOG-ITEM
            'BUTTON-DIALOG-ITEM   #@(130 140)  #@(114 23)
            "OK"
            #'(lambda (item) (declare (ignore item))
               (return-from-modal-dialog t))
            :DEFAULT-BUTTON    T)))   ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CONTROL-PANEL

(defun make-losp-control-panel ()
  (setq *problem-number-comtab* (make-comtab))
  (comtab-set-key *problem-number-comtab*
                  '(#\Newline) 'accept-problem-number-text-entry)
  (setq *isolate-variable-comtab* (make-comtab))
  (comtab-set-key *isolate-variable-comtab*
                  '(#\Newline) 'accept-isolate-variable-text-entry)
  (setq *losp-control-panel*
        (MAKE-INSTANCE 'control-panel-window
          :WINDOW-TYPE         :TOOL
          :WINDOW-TITLE        (format nil "Losp Control Panel")
          :VIEW-POSITION       '(:TOP 208)
          :VIEW-SIZE           #@(230 254)
          :VIEW-FONT           '("Chicago" 12 :SRCOR :PLAIN)
          :parent              *current-entry-window*
          :VIEW-SUBVIEWS       (losp-control-panel-subviews)))
  (set-radio-buttons-when-opened)
  (set-logic-check-box-when-opened)
  (set-circuit-check-box-when-opened)
  (set-trace-check-box-when-opened)
  (set-database-check-box-when-opened))

(defun losp-control-panel-subviews ()
  (LIST (MAKE-DIALOG-ITEM
         'STATIC-TEXT-DIALOG-ITEM   #@(10 5)  #@(56 16)
         "Analysis"
         'NIL)
        (MAKE-DIALOG-ITEM
         'BUTTON-DIALOG-ITEM   #@(70 3)  #@(60 18)
         "Apply"
         #'(LAMBDA (ITEM) (apply-button-action item))
         :VIEW-FONT                  '("Courier" 12 :SRCOR :PLAIN)
         :view-nick-name             'apply-button
         :DEFAULT-BUTTON              NIL)
        (MAKE-DIALOG-ITEM
         'RADIO-BUTTON-DIALOG-ITEM   #@(10 28)  #@(110 16)
         "Transcribe"
         #'(LAMBDA (ITEM) (transcribe-radio-button-action item))
         :VIEW-FONT                   '("Geneva" 12 :SRCOR :PLAIN)
         :view-nick-name              'transcribe-radio-button
         :RADIO-BUTTON-PUSHED-P       nil)
        (MAKE-DIALOG-ITEM
         'EDITABLE-TEXT-DIALOG-ITEM   #@(160 222)  #@(52 15)
         ""
         #'(LAMBDA (ITEM) (case-variable-text-action item))
         :VIEW-FONT              '("Geneva" 12 :SRCOR :PLAIN)
         :view-nick-name         'isolate-variable-text-box
         :comtab                 *isolate-variable-comtab*
         :ALLOW-RETURNS          T)  ))

;;;18 other dialog-item specifications omitted here
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ACTIONS
;;;
;; see process file for usage of the globals
;;   *valid-analysis-levels*  *current-analysis-level*
;;   *active-displays*  *most-recent-analysis-result*
;;   *current-entry-window*

(defun set-radio-buttons-when-opened ()
  (cond
   ((eq *current-analysis-level* '*TRANSCRIBE*)
    (radio-button-push
     (view-named 'transcribe-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*CLEAN*)
    (radio-button-push
     (view-named 'clean-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*SORT*)
    (radio-button-push
     (view-named 'sort-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*EXTRACT-LITERALS*)
    (radio-button-push
     (view-named 'extract-literals-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*CANCEL-BOUNDS*)
    (radio-button-push
     (view-named 'cancel-bounds-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*INSERT-BOUNDS*)
    (radio-button-push
     (view-named 'insert-bounds-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*MINIMIZE*)
    (radio-button-push
     (view-named 'minimize-radio-button *losp-control-panel*)))
   (T nil)))

(defun transcribe-radio-button-action (self)
  (setq *current-analysis-level* '*TRANSCRIBE*)
  self)

(defun clean-radio-button-action (self)
  (setq *current-analysis-level* '*CLEAN*)
  self)

(defun sort-radio-button-action (self)
  (setq *current-analysis-level* '*SORT*)
  self)

(defun database-display-box-action (self)
  (let ((win (when *current-entry-window*
               (find-parent-child *current-entry-window* 'database))))
    (if win
      (window-close win)
      (make-database-window *current-entry-window*)))
  self)

;;;20 other action specifications omitted here
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ENTER LOSP-ENTRY-WINDOW
;;;
;; Controls the behavior of the 'return' key in the entry buffer.
;; If the cursor is not on the last line of the buffer, 'return' copies
;; the current line (without the prompt) to the end and moves the cursor
;; there too.  No error handling is provided here.

(defun accept-entry (entry-window &optional force)
  (let* ((bmark (fred-buffer entry-window))
         (end (buffer-line-end bmark))
         (eob (buffer-size bmark))
         (entry (string-left-trim *entry-prompt*
                  (buffer-substring bmark (buffer-line-start bmark) end)))
         (symbol-entry (string2symbol-boxed entry)))
    (cond                              ;; accept input
     ((or force (= end eob))
      (set-mark bmark eob)
      (ed-insert-char entry-window #\Newline)
      (if (null symbol-entry)
        (buffer-insert-at-end bmark "return")
        (let ((logic-type
               (intersection (flat symbol-entry) *valid-logic-functions*))
              (assertion-type (member *assertion-token* symbol-entry)))
          (cond
           (logic-type
            (buffer-insert-at-end bmark *multiple-form-message*))
           (assertion-type
            (buffer-insert-at-end bmark "Assert:  ")
            (buffer-insert-at-end bmark
              (assert-entry entry-window (remove-assert-mark symbol-entry))))
           (T (let ((result (process-entry entry-window symbol-entry)))
                (buffer-insert-at-end
                 bmark (prepare-text-out result)))))))
      (ed-insert-char entry-window #\Newline)
      (buffer-insert-at-end bmark *entry-prompt*))
     ;; otherwise copy entry to the end of the buffer
     (T (buffer-insert-at-end bmark entry)))))

(defun force-accept (entry-window) (accept-entry entry-window T))

(defun buffer-insert-at-end (bmark string)
  (buffer-insert bmark string (buffer-size bmark))
  (set-mark bmark (buffer-size bmark)))

(defun prepare-text-out (form)
  (cond
   ((null form)  "")
   ((marked form) "()")
   (T (let ((string-form (symbol2string form)))
      (remove #\) (remove #\( string-form :count 1)
              :count 1 :from-end t)))))

;;;many other handlers and functions omitted here
```

PROJECT REFINEMENT

For our class project, we will be developing a single website which demonstrates ***knowledge-based hyperlinks*** *(perhaps just* ***SmartLinks****)*. Active items (words, graphics, diagrams, sections, etc) will permit traversal of the website based on semantic rather than syntactic references.

Discussions and decisions:

1. Review of relevant class handouts, and research into possible approaches.

2. Identify the task that the potential user of the website will be trying to accomplish. Develop several scenarios which capture the semantic need and intent.

3. Identify the types of traversal available to the user. Rough out the engine functionality and the system architecture.

4. Select a content area for the site which facilitates task accomplishment. The types of smart links will depend directly on the content and functionality of the site.

5. Identify the requisite languages, skills and roles for the project. (content and site development, link definition, engine development, interaction design,...)

6. Discuss the issue of novice vs expert users.

7. Brainstorm possible models of interactivity and interface displays. How will the user:
   1) know what is possible? 2) know what to do? 3) communicate their needs?

8. Assignment of tasks to individuals.

Recall

- Our links will be more useful if they are filters rather than generators.

- We may use several types of traversal, but each type will have a separate underlying traversal graph. We will eliminate interaction between semantic components.

Individual assignment:

Construct a graph of a possible site, with nodes being content chunks and links being traversals. You will need to

1. make up some *rough* content chunks in a content area (the class should have decided the content area tonight),
2. imagine some tasks, queries and traversals,
3. identify the type of connection being traversed,
4. specify in detail some content containing the link and some content being traversed to, with emphasis on the semantic connection between the two, and
5. be prepared to show this graph to the class.

PROJECT REFINEMENT

The content of our project is **Childhood Ailment Diagnosis**. The rough architecture is to use **SmartLinks** to connect the HTML-page-structure to a semantic-network, traverse the semantic-network, and then return to the HTML location which correcsponds to the end of the semantic-network path.

So we'll be constructing two (or more) databases: the **structural-HTML-database** and the **semantic-net-database**. Then we'll be constructing a **linking database**, which contains the connectivity between syntactic and semantic elements. We will also need a **traversal engine** as the back-end, and a **query interface** as the front-end.

Issues:

1. Review class assignments; make a rough pass at the data structures (the HTML structure graph and the semantic network graph).

2. Refine content area.

3. Refine semantic nets, and identify what we can do semantically. Types of traversal. Identify the language of the semantic-net (ie what types of nodes and links)

4. Discuss the types of structural (HTML) forms. Grainsize of information units; grainsize of textual and paragraph HTML; types of knowledge units. Other types of syntactic organization. Other types of semantic queries.

5. Rough pass at the control structure architecture.

6. Discuss the implications of the semantic-net modleling approach, the notion of ontology and the mappings from semantic-nets to relational calculus.

7. Discuss the limitations and traps in the proposed control architecture.

8. Identify roles, esp. who will be writing what code. Identify the requisite languages, skills and roles for the project. (content and site development, link definition, engine development, interaction design,...)

9. Continue to develop usage scenerios.

10. Consider the interaction and user-interface issues.

11. Discuss the issue of novice vs expert users.

12. Discuss the issue of partial vs. complete knowledge. Hypothesis testing in diagnosis.

Evolving the Interface

The WIMP metaphor (windows, icons, menus, pointer) appeared in public in 1984, designed for personal computers with naive users, narrow applications, weak processors, impoverished bandwidth and i/o, and stand alone usage. Isn't it time for a change?

	WIMP	BUFF
•	Metaphor	Reality, virtuality

The book, desktop, office room, etc are all weak metaphorical maps. We should be interacting with a strong mapping of the task itself.

•	Direct Manipulation	Delegation

Drag and drop makes drudgery easy but it provides no abstractions. Wouldn't you prefer to delegate those repetitive jobs to the system?

•	See and Point	Describe and Command

It's nice to see what you are manipulating, but this is a regression to first grade. We need tools that are driven by language and abstraction, not by touch

•	Consistency	Diversity

Consistency reduces the need for thought, but the world is actually complex and diversified. The pencil and paper suggest an ideal flexible, easy-to-use tool.

•	WYSIWYG	Represent meaning

Wysiwyg is a mapping to output that ignores the meaning of the output. We need semantics included at the display level, so that the interface knows why a phrase is in italics.

•	User Control	Shared Control

Letting the user steer the process gives a feeling of control, but is far too much work. We don't write essays by a single button push, so we must recognize that control is difficult and we should welcome help from agents and others.

•	Feedback and Dialog	System handles details

Clear, consistent feedback is like having your boss always looking over your shoulder. We should hide most processes; do you really want feedback from the garbage collection algorithm?

•	Forgiveness	Model User Actions

Reversible actions permit a user to make and revoke errors. With a little bit of contextual understanding, however, the system can forbid letting those errors from happening directly.

- Aesthetic Integrity — Graphic Variation

Simple, clean interfaces are also limited in capability, drab and boring. We need help navigating large spaces; variation and diversity are appropriate roadsigns.

- Modelessness — Richer Cues

It is idealistic and foolish to expect to do anything at any time. Modes are task specific, lets learn to identify contexts rather than to blur our vision.

ISSUE	WIMP	BUFF
users	naive	post-nintendo
applications	productivity	ubiquitous
power	weak	humungous
communication	impoverished	rich
connect	standalone	deep and dynamic
language	icons	English language
objects	weak and big	many, small, rich
origin	finder/files	personal information
travel	surf	push to you
image	be your best	don't work hard

What is suggested is a paradigm change, not an incremental improvement. The components of a paradigm are all mutually reinforcing, so that the desktop metaphor does not readily adapt to changes of the parts.

Main Points:

- Language must play a central role at the interface. Language is abstract, negotiable, contextual, multimodal, and ambiguous, it is not a physical metaphor.

- Objects need richer representations, multiple views for multiple uses. Objects need to include some notion of their meaning.

- The interface needs more expressive power and diversification to handle information complexity.

- There are more expert users and more agents and proxy users. The user base is smarter, networked, and dealing with too much information too readily accessible but not organized.