Interface Design Simulation

Objectives:

Experience HCI design using a detailed task specification. Integrate suggestions for design in the text into a task-oriented design context. Provides a context to discuss design methodology and choices.

Task:

A programming team in your organization has developed a new deductive engine which allows application programmers to manipulate data for expert systems. Your job is to design a prototype interface for this engine.

The engine provides functions for a knowledge engineer to restructure, optimize, verify, and in general manipulate the components of a knowledge base of logical and arithmetic constraints. What is neat about this engine is that it maintains a graphic, network description of the logical transformation processes, and like a circuit, distributes logic over many network nodes.

However, different departments in your organization have different formats for their knowledge-bases, want to do different things to their data, and require different outputs and views of their data. Furthermore, some users want automated functionality and some want fine-grain interactivity with transformations.

Due to organizational preferences, the interface is to be constructed by three separate teams, one team for each of the following aspects:

Aspect 1: function calls to the interface	(programming, API)
Aspect 2: screen layout and interactivity	(interface, dialog)
Aspect 3: hardware i/o devices and functionalities	(architecture)

Fortunately, some members of each team can cross development boundaries and work with the other teams as advocates of their design process.

You are to add appropriate interface controls for things like opening and closing the system, trapping and notifying about input errors, and improper control configurations. Also, you should select appropriate names, labels, and displays for both naive and sophisticated users. You are not responsible for

explaining how the engine works,
the help system, or
the editors which allow databases to be constructed,

although you should include interface hooks to all three subsystems.

You are free to modify and enhance interface specifications to make the engine easy to use, so long as the requested functionality is available.

Below is a (loosely) structured listing of some of the requirements for your interface.

* Backend processor assignment: single, distributed

* Input language: logic, Prolog, Lisp

* Input form: files, keyboard

* Display:
  - linear (textual) view in any input syntax,
  - graphic (network) view,
  - internal computational view if requested

* Simple transformations: absorb, clarify, extract, coalesce

* Compound transformations: subsume, cancel, collect

* High-level transformations:
  - optimize relative to specified time and complexity parameters
  - identify contradictions, verify consistency
  - delete irrelevant facts
  - cluster facts in groupings relative to a particular set of variables

* Network display controls:
  - select an active subnetwork to perform transformation on
  - rearrange network by hand
  - rearrange network using energy minimization algorithm
    - parameterized by spring coefficient, spring divisor,
    - repulsion coefficient and relaxation stepsize
  - labels on or off

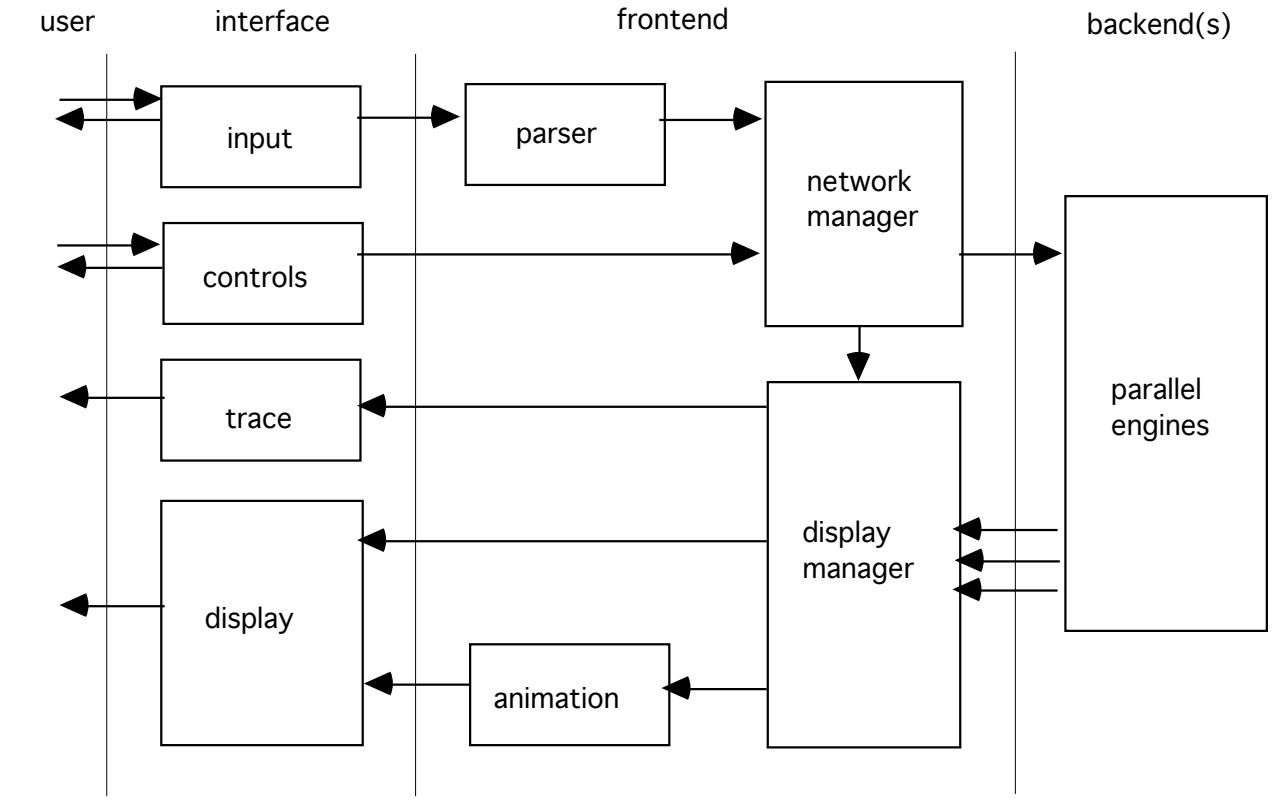
* Network animation:
  - show animation in forward or reverse order
  - stop and start animation freely
  - show active network components and their activity
  - specify rate of animation by
    - frames per second
    - transformations per second
    - specific transformations per second
    - activity indications per second

* Trace:
  - show engine activity by transformations performed and
  - by animation instructions performed

* Users also want to be able to :
  - focus on any display with full screen, especially the network display
  - reset display at any point
  - refresh display
  - load and display new logical databases
  - select textual parts of a database for analysis

The engine transformations can be applied individually or in any grouping. The backend engine(s) are much faster than the display, so the display manager collects engine activity and structures a display which makes sense to a person. It is important that the users of the engine understand the logic of the transformations. The engine transformations use an internal coding that is not easy to understand.

Here is the functional architecture of the system:



Backend(s):

Computational machinery on which transformations are done; can be a parallel, distributed array of processors

Frontend:

Manages the interface and coordinates assignment of and communication with backend. Backend coordination can be organized by synchronous or asynchronous message-passing or by shared memory.

Interface:

The information and controls seen by the user.