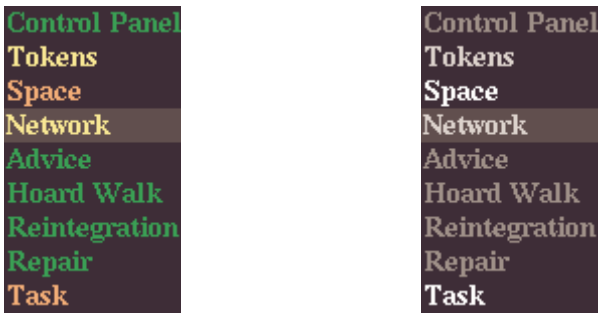
These nine indicators appear in a small window that measures about 1" x 2" on the user's display. These indicators require a modest amount of screen real estate, a resource that is precious to users, during normal operation. Because the expected benefit of the indicators outweighs its cost in screen real estate, I expect the user will keep this indicator window visible at all times. Note, however, that while the user investigates a problem indicated by the interface, the screen real estate requirements increase. This change in resource consumption is appropriate during periods when the user is actively engaged with the interface [8].

Each indicator light can signal one of four states about the subsystem it monitors:

- Operating normally
- Developing a problem or experiencing a noncritical problem
- Experiencing a critical problem
- Indicator not operational or status unknown

The first three of these states are color coded to one of three urgency levels: normal, warning, and critical. By default, the colors associated with these urgency levels are green, yellow, and red. I chose this color scheme to make it easy for users to remember since green, yellow, and red correspond to the colors used to signify similar meanings in many situations in the United States—most notably traffic signs and signals. This color scheme is not without its own difficulties. For instance, some users may be using a monochrome monitor, others may be color blind, and not all cultures use green, yellow, and red for these meanings. For these reasons, the user can easily customize the color scheme used to represent these three urgency levels. In fact, the user can forego the use of color and instead use different grayscale levels to signify the different urgency levels. Figure 4.1 shows two views of the indicator window: one in the default green-yellow-red color scheme and one in a monochrome scheme.[8]

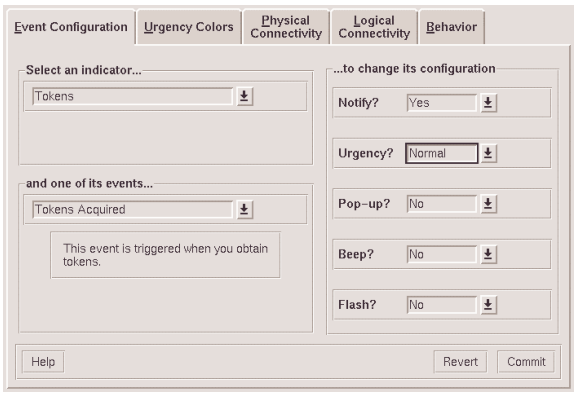


(a) Color Scheme (b) Monochrome Scheme

Figure 4.1: Indicator Lights

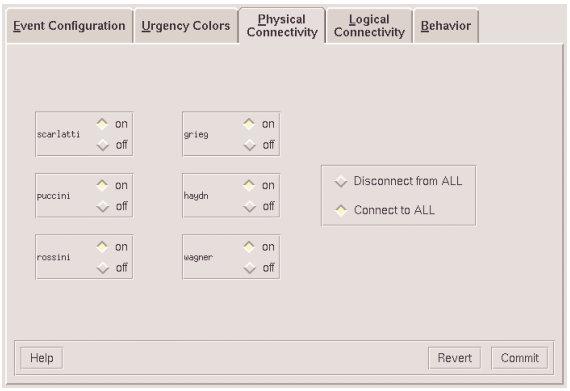
This figure shows two views of the indicator lights. The indicator lights give users a peripheral awareness of system state. View (a) shows the default green-yellow-red color scheme. View (b) shows a monochrome scheme. In both views, the *Tokens* and *Network* indicators are shown with a warning urgency level and the *Space* and *Task* indicators are shown with a critical urgency level. All other indicators are normal.

[8] A color supplement to this thesis is available at http://www.cs.cmu.edu/Reports

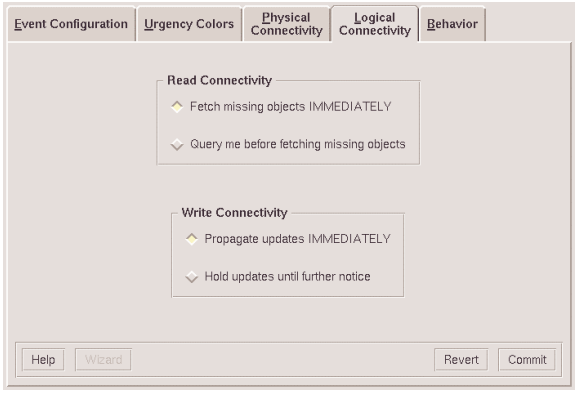


(a) Event Configuration Tab

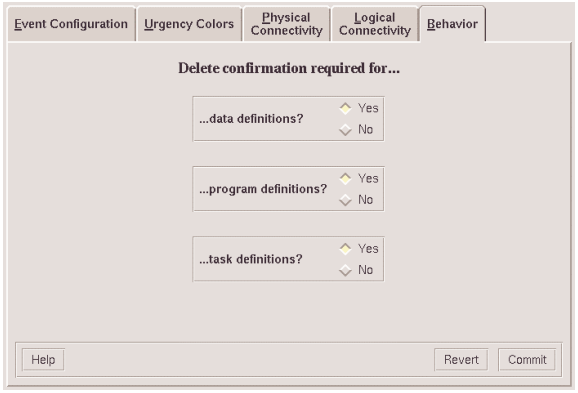
(b) Urgency Colors Tab



(c) Physical Connectivity Tab



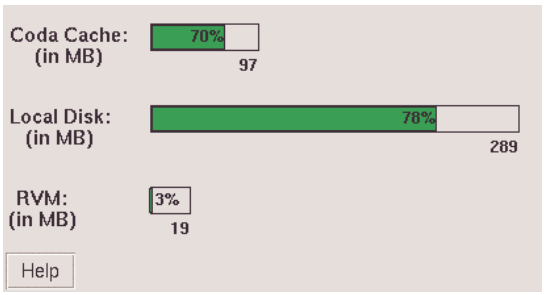
(d) Logical Connectivity Tab



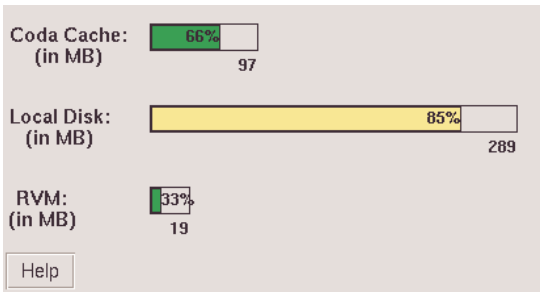
(e) Behavior Tab

Figure 4.2: The Tabs of the Control Panel

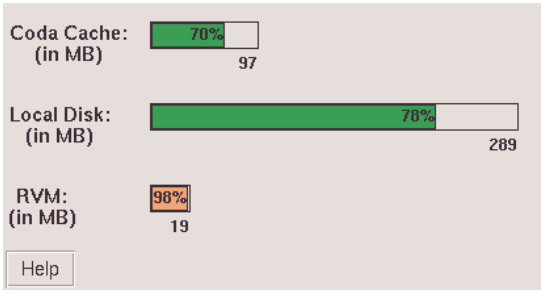
This figure shows the supporting windows that appear when the *Control Panel* indicator is double-clicked. From these windows, the user can control various aspects of the interface. The *Event Configuration* tab, view (a), allows the user to control the way events are notified. The *Urgency Colors* tab, view (b), allows the user to customize the color scheme used to indicate the three levels of urgency. The *Physical Connectivity* tab, view (c), allows the user to control connectivity to each Coda server. The *Logical Connectivity* tab, view (d), allows the user to control the servicing of cache misses and the propagation of updates while remaining fully connected to the network. The *Behavior* tab, view (e), allows the user to disable confirmation dialogue boxes.



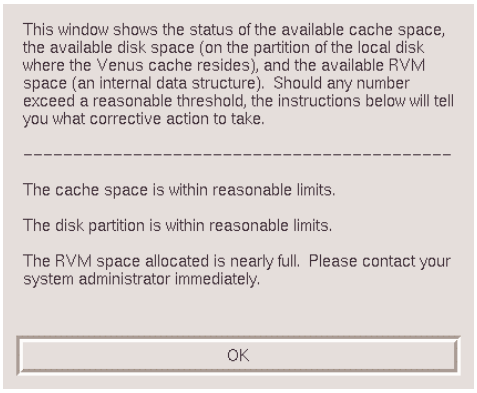
(a) All Systems Normal



(b) Local Disk is Filling Up



(c) RVM Space Almost Full

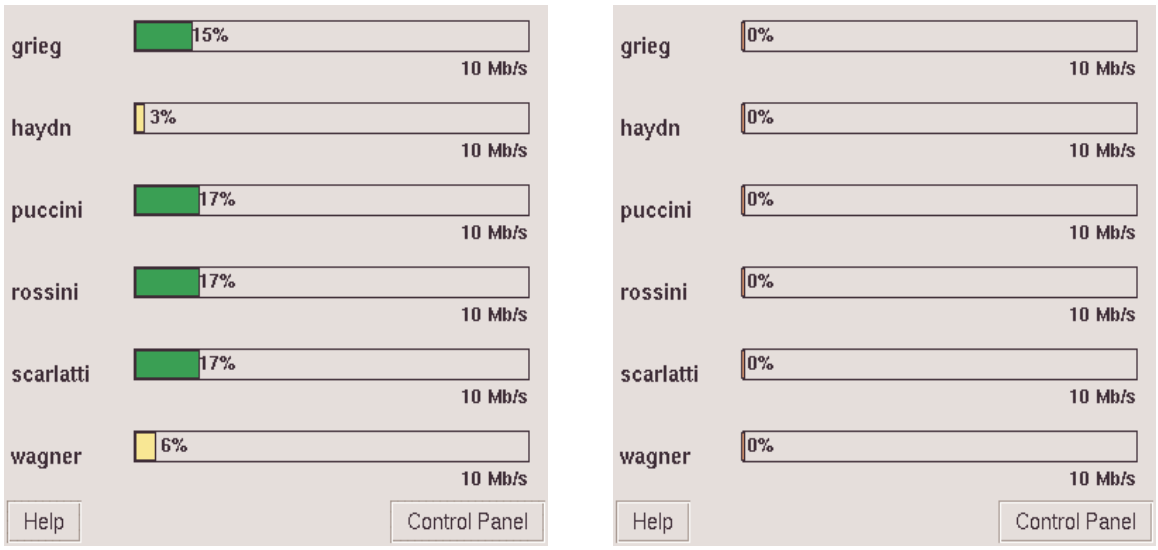


(d) Help Window for (c)

Figure 4.4: Space Information Window

This figure shows three views of the *Space Information* window as well as its help window. View (a) shows the *Space Information* window when all three areas are within normal limits. The three gauges in this view are all green. View (b) shows the window when the local disk is beginning to fill up. Because there is still space available, the local disk gauge is yellow (as would be the *Space* indicator light). View (c) shows the window when the RVM space is almost full. Because there is almost no RVM space available, its gauge (and the indicator light) would be red. View (d) shows the help window that would appear if the user were to click on the *Help* button of view (c). This window explains that the user should contact their system administrator to resolve this problem.

Currently, this window shows one gauge for each server of which the client is aware. As the number of Coda servers increases, the content of this window becomes lost to the user. This window should be modified to list only those servers the user is actively accessing. If the user is accessing more servers than can fit on the window, the interface should allow them to be scrolled. Neither of these extensions would be difficult. Further, the gauge shows the network bandwidth as a percentage of the maximum Ethernet bandwidth, a metric that is not meaningful to the user. Rather than showing the relative percentage of Ethernet bandwidth (e.g., 3%), the interface could show the absolute bandwidth estimate (350 Kb/s). Neither of these modifications to the interface would be difficult.



(a) Client operating weakly connected (b) Client operating disconnected

Figure 4.5: Network Information Window

This figure shows two views of the *Network Information* window. View (a) shows that this client is operating weakly connected to `haydn` and `wagner`, but strongly connected to all the other Coda servers. The gauges for `haydn` and `wagner` are yellow; those for the other servers are green. The *Network* indicator light would be yellow in this situation. View (b) shows that this client is operating disconnected to all servers. The gauges show a thin sliver of red. The *Network* indicator would also be red.

The distinction between strong and weak connectivity is made based upon comparing the current network bandwidth estimate to a threshold. If the current bandwidth is above the threshold, the system considers itself to be strongly connected; if it is below, the system considers itself weakly connected. Ideally, users should have the ability to control the value of this threshold, though novice users should not be required to manipulate it. Such an extension is discussed in Section 9.2.2.1.

The behavior of this indicator, as described here, is probably not what I would design today. The problem is that the indicator changes state before it is really appropriate to do so. For example, as soon as the system realizes that a server has crashed, the indicator turns red. However, because most Coda files are replicated, the fact that a single server has crashed is not terribly important to the user. It would be more appropriate to change the state of the indicator when a volume has transitioned into the disconnected state (of Figure 2.1). The informational window would then need to expose the pathnames of volumes operating disconnected (and weakly-connected) as well.

4.6 Advice

The fourth indicator light, labeled *Advice*, alerts the user to pending requests for advice from Venus and of hoard hints. At critical decision points during operation, Venus may request advice from the user. These requests frequently relate to network usage. At other times, Venus may notice anomalies in the user's hoard database and may alert the user to them by way of a hint.

When the user double-clicks on this indicator light, the window shown in Figure 4.6 appears. It has two sections. The top section, labeled Advice Needed, contains a list of advice requests. The bottom section, labeled Advice Offered, contains a list of hoard hints. Each type of request and each type of hint are marked with either a warning or critical urgency level. By default, requests are marked as critical only when a thread is blocked waiting for a response. All other requests and all hints are alerted at the warning level. The color of the indicator light is red if any critical requests are pending user attention, and yellow if any noncritical requests or hints are pending attention. It is green if there are no requests or hints outstanding.

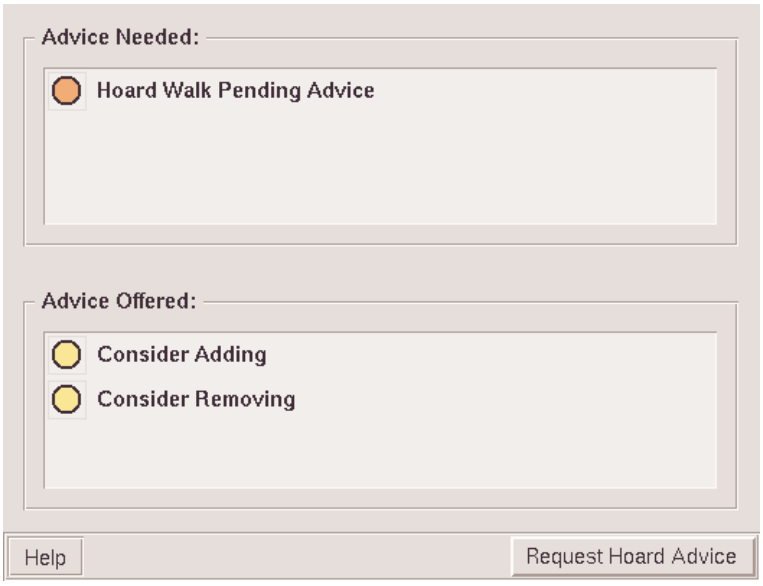
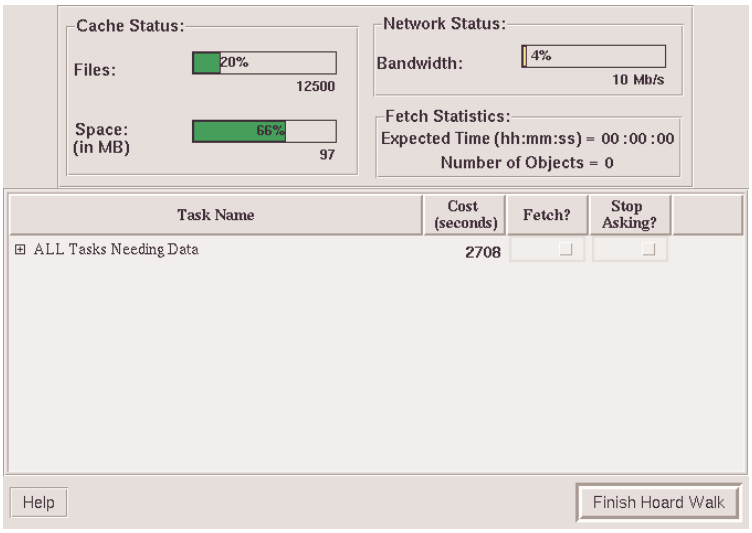
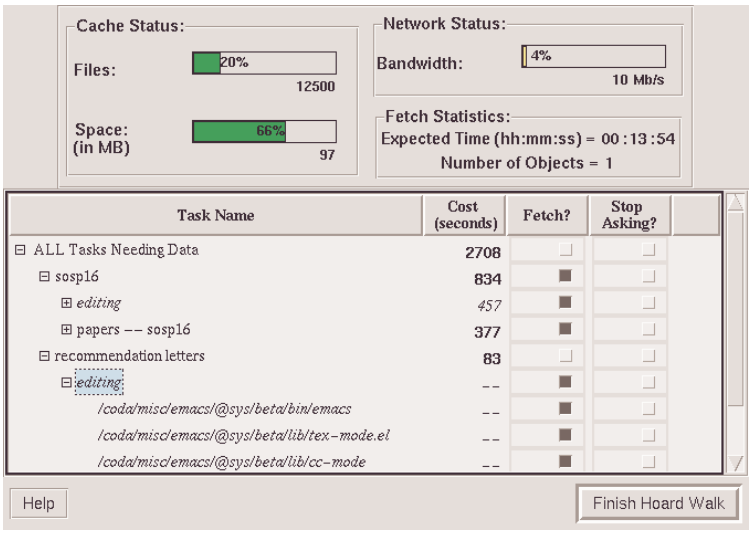


Figure 4.6: Advice Information Window

This figure shows the *Advice Information* window. It is displayed whenever the user double-clicks on the *Advice* indicator light. The top portion of the window, labeled Advice Needed, contains queries posed by Venus. By answering these questions, the user may influence the behavior of Venus. The bottom portion of the window, labeled Advice Offered, contains hints offered to the user regarding the content of the cache and of the hoard database. The circles to the left of each entry indicate the urgency of the request.



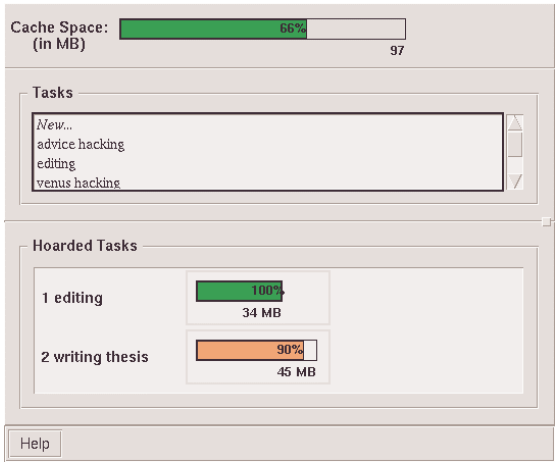
(a) Initial View



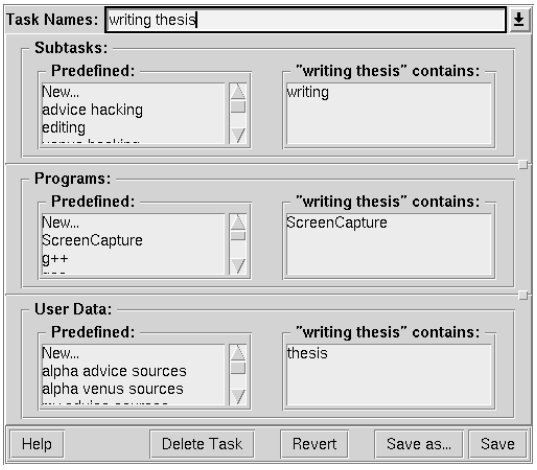
(b) Expanded View

Figure 4.10: Hoard Walk Advice

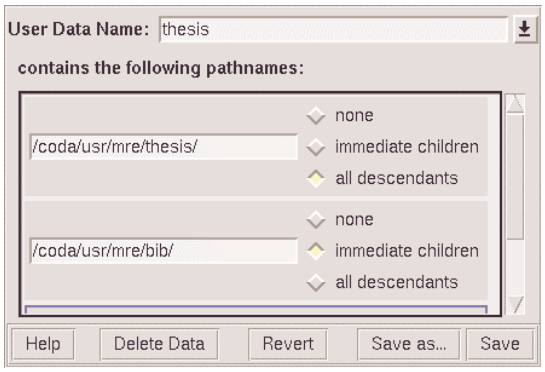
This figure shows a hoard walk advice request. View (a) is displayed initially. The upper left corner shows the status of the cache: the percentage of cache container files and cache blocks dedicated to hoarded objects. The upper right corner shows the current average bandwidth to the servers. The expected time to complete the hoard walk is shown below the network status. The main area of the window shows a list of the tasks that need data to be fetched. Initially, only the "All Tasks Needing Data" task is shown. If the user expands this task as well as the tasks one level below it (by clicking on the "+" signs or double-clicking their names) and then selects the `sosp16` task, the window shown in view (b) will be displayed. Italicized text names indicate a task that is contained in more than one task definition. For each element of this hierarchical list, the expected cost (currently shown only in units of time) is listed to the right of the task name. Further to the right are two checkboxes. The left checkbox allows the user to select the item to be fetched. The one to the right allows the user to instruct the system to not fetch the item during the current hoard walk and, furthermore, to not ask about this item in future hoard walks during this weakly connected session.



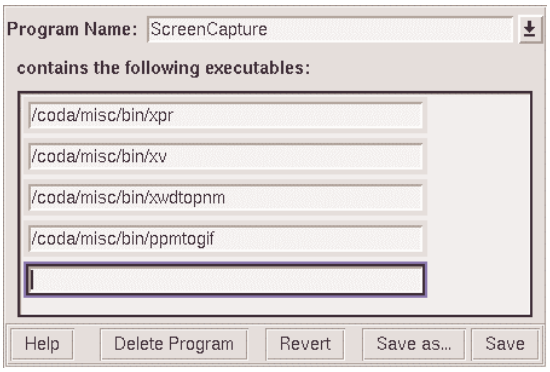
(a) Task Information Window



(b) Task Definition Window



(c) Data Definition Window



(d) Program Definition Window

Figure 4.15: The Windows Associated with the Task Indicator

This figure shows the four primary windows associated with the *Task* indicator light. View (a) shows the window displayed after clicking on the indicator. The top section of this window shows the status of the cache; the middle section shows a list of all defined tasks; and the bottom section shows those tasks that have been hoarded, their priorities, and their availability. View (b) shows the definition for the "writing thesis" task. This definition contains the "writing" task, the programs for "ScreenCapture", and the "thesis" user data set. View (c) shows the definition for the "thesis" user data. This definition includes the `/coda/usr/mre/thesis` directory and all its descendants as well as the `/coda/usr/mre/bib` directory and its immediate children. View (d) shows the definition for the "ScreenCapture" programs, including `xpr`, `xv`, `xwdtopnm`, and `ppmtogif`.

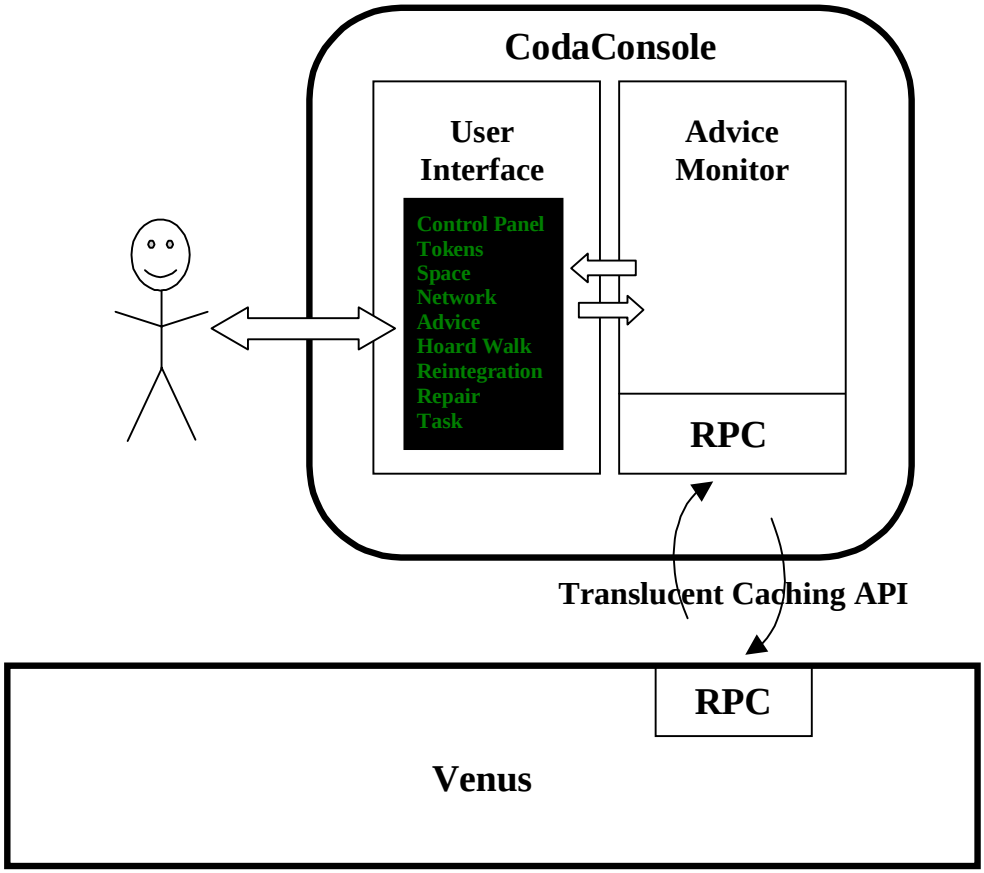


Figure 5.1: Architecture

This diagram shows the architecture of the system. The CodaConsole consists of two parts: the user interface and the Advice Monitor. The user interface implements the graphical interface described in the previous chapter. The Advice Monitor acts as a liaison between Venus and this interface. Venus notifies the Advice Monitor of various events via the RPC interface implementing the Translucent Caching API.

These three pieces communicate using two different mechanisms. The user interface communicates with the Advice Monitor via a pair of unidirectional pipes, as shown in the figure. The Advice Monitor communicates with Venus via a pair of RPC2 connections, also shown in the figure.

5.2 Details of Implementation

This section describes the implementation of each of the three components described above. I begin by describing the implementation of the user interface, focusing on the finite state machines that drive the indicator lights. I then present a brief description of the Advice Monitor. I conclude with a summary of the modifications necessary to implement the Translucent Caching API within Venus.

5.2.1 User Interface

Chapter 1 described the design of the user interface component of the CodaConsole. The user interface, which is implemented in Tcl/Tk [39, 56] using the Tix widget library [29], assumes the user is running a windowing system[14]. It is entirely event driven. As events arrive, the interface notifies the user via the indicator lights. As the user requests information, it displays the appropriate information windows. If it needs data from Venus, it contacts the Advice Monitor to request that data.

The interface presented to the user contains a set of indicator lights. These indicator lights are implemented as small, event-driven, finite state machines. The arrival of an event potentially causes a transition from one state to another. These state changes may then cause the appearance of the indicator lights to change. I will now describe three of these state machines. The remaining ones are similar in spirit, though the details differ.

5.2.1.1 Tokens Indicator

The finite state machine for the *Tokens* indicator has three states: Valid, Invalid, and Expired&Pending. The user is required to have tokens before the advice monitor begins executing so the finite state machine will start in the Valid state. When the interface receives notification that the user's tokens have expired, the machine transitions into the Invalid state. If the interface then receives a notification that an activity, a reintegration for example, is pending tokens, the machine transitions into the Expired&Pending state. The machine transitions to the Valid state from either of the other two states upon receiving notification that the user has obtained valid tokens. These states and the transitions between them are shown in Figure 5.2.

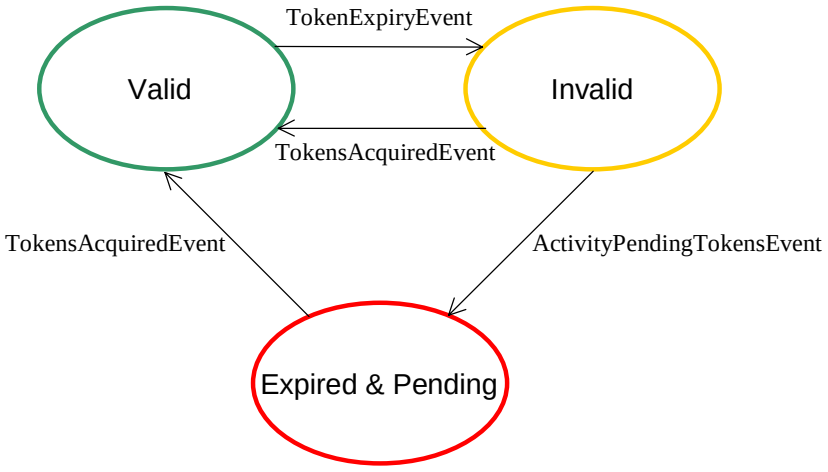


Figure 5.2: State Machine of the Tokens Indicator

[14] Currently, the system works under X windows, though it does not require that windowing system.

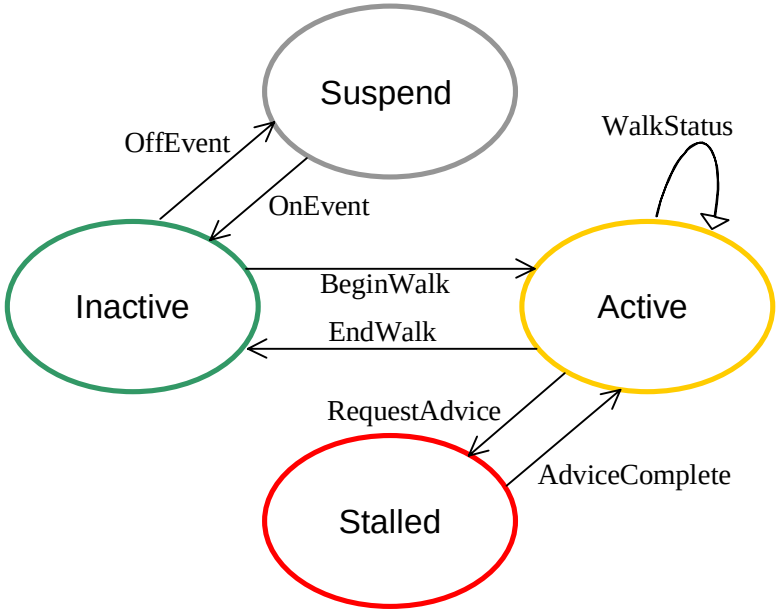


Figure 5.3: State Machine of the Hoard Walk Indicator

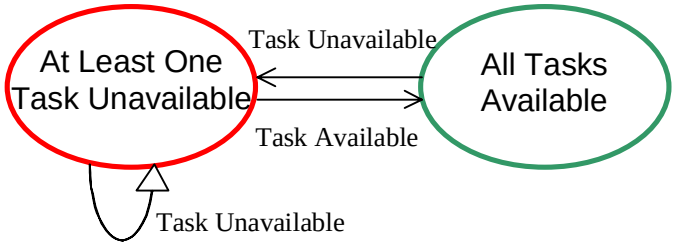


Figure 5.4: State Machine of the Task Indicator

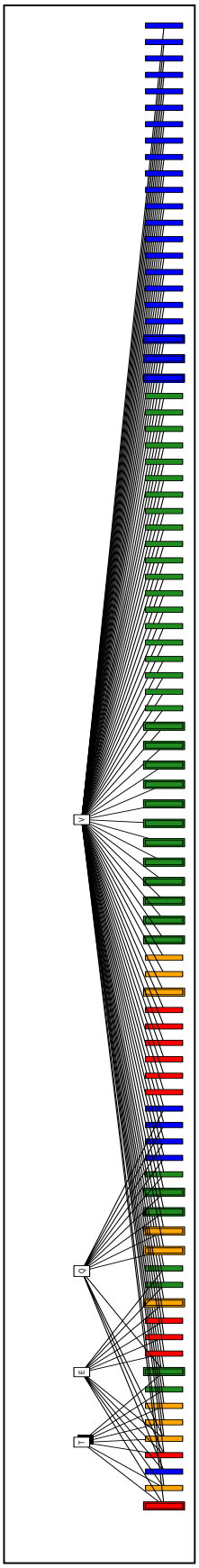


Figure 7.33: Finding Originations
This figure shows where evidence for each finding of Appendix F originated. The four top-level nodes correspond to the tutorial (T), exercises (E), questionnaire (Q), and transcripts of verbal protocol (V). Each leaf in the tree represents a single finding. The color of the leaves represents the severity rating of the given finding. Red indicates a Level 1 severity rating; yellow represents a Level 2 finding; green represents a Level 3 finding; and, blue represents a Level 4 finding. The size of the leaf represents the scope of the finding. Wide leaves correspond to global findings; narrow leaves correspond to local findings.

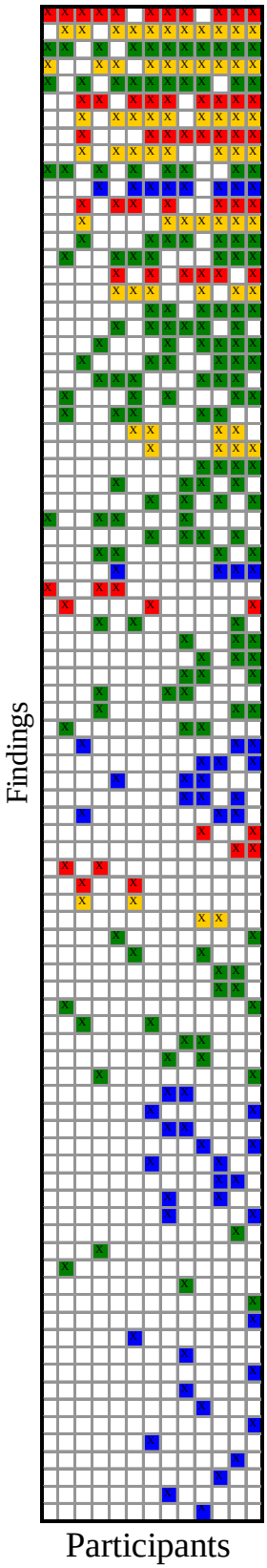


Figure 7.34: Frequency of Reporting
This matrix indicates which users contributed evidence to which findings. Each box indicates that the given user contributed evidence to the given finding. The color of the box indicates the severity of the finding: red represents level 1 problems, yellow represents level 2, green represents level 3, and blue represents level 4. The matrix has been sorted so that more productive users appear in the rightmost columns and more frequently reported problems appear towards the top. The order of users from left to right is: P2, S1, N4, P3, P5, C1, N1, E2, P4, T1, N2, E1, E3.

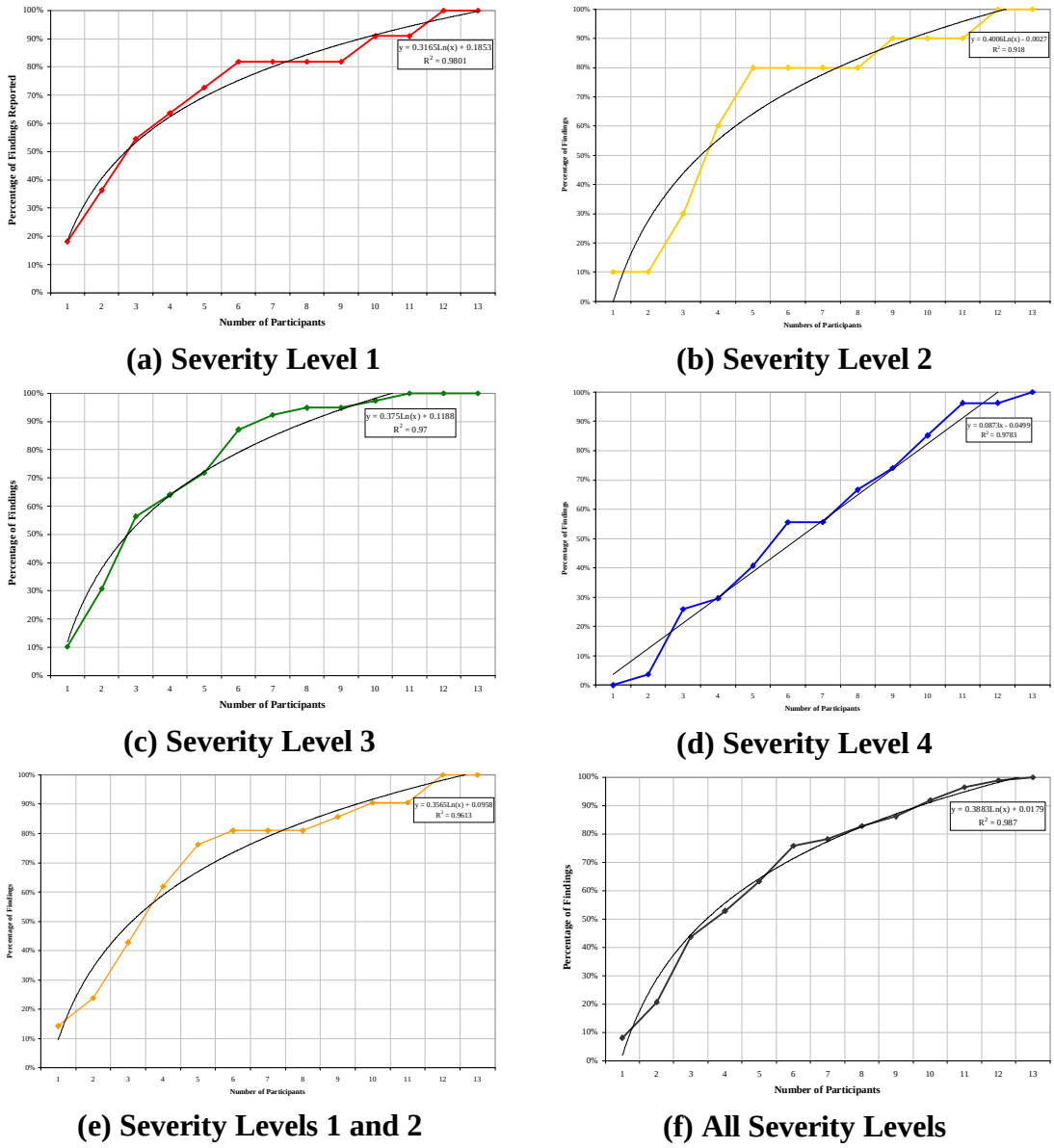
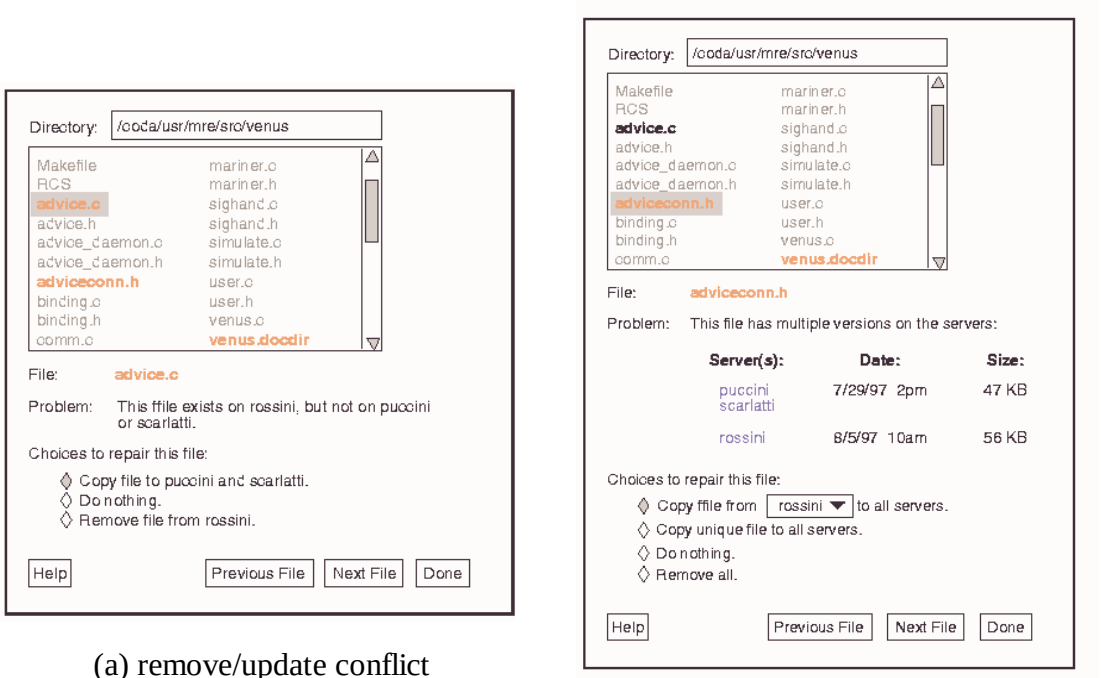


Figure 7.35: Number Users Required to Cover Most Severe Findings
This figure shows the number of participants necessary to cover the usability findings. In Views (a)-(d), I show just the findings with severity levels 1-4 respectively. In View (e), I show the most critical findings, those with a severity level of 1 or 2. In View (f), I show all findings. In each view, I show the closest function that approximates the data. All but one graph is most closely approximated by a log function. View (d), however, is more closely approximated by a linear function.



(a) remove/update conflict

(b) update/update conflict

Figure 9.1: Repairing Objects in Conflict

This figure shows the preliminary design of the repair module for the CodaConsole interface. Windows similar to these would appear when the user double-clicks on the name of an object in conflict in the *Repair Information* window of Figure 4.14. View (a) shows the screen that would allow users to manually repair a directory with a remove/update conflict. In this case, the file exists on `rossini`, but not on `puccini` or `scarlatti`—perhaps one user deleted it while communicating with `puccini` and `scarlatti` but another user updated it while communicating only with `rossini`. View (b) shows the screen that would allow users to repair an update/update conflict manually. In this case, users who were operating partitioned from one or more servers both updated the file. Thus, the version of it stored on `puccini` and `scarlatti` differs from the one on `rossini`. In these drawings, objects whose names appear in red are in conflict, those whose names appear in black have fixes pending, and those whose names appear in gray are not in conflict. Servers whose names appear in blue represent hyperlinks that allow the user to examine the content of individual replicas. The next and previous object buttons take the user to the next or previous object in conflict. The done button begins the actual repair.