15.3 TASK DECOMPOSITION

The example above of vacuum cleaning showed how a task, ‘clean the house’, was decomposed into several subtasks: ‘get the vacuum cleaner out’ and so on. Most task analysis techniques involve some form of task decomposition to express this sort of
0. in order to clean the house
1. get the vacuum cleaner out
2. fix the appropriate attachment
3. clean the rooms
   3.1. clean the hall
   3.2. clean the living rooms
   3.3. clean the bedrooms
4. empty the dust bag
5. put the vacuum cleaner and attachments away

Plan 0: do 1 – 2 – 3 – 5 in that order.
        when the dust bag gets full do 4

Plan 3: do any of 3.1, 3.2 or 3.3 in any order
        depending on which rooms need cleaning

Figure 15.1 How to clean a house

behavior. Hierarchical task analysis (HTA) is typical of such an approach [15, 318]. The outputs of HTA are a hierarchy of tasks and subtasks and also plans describing in what order and under what conditions subtasks are performed.

For example, we could express the house-cleaning example as in Figure 15.1, further decomposing the subtask 'clean rooms'. Indentation is used to denote the levels in the task hierarchy, and the tasks are also numbered to emphasize this hierarchy. The plans are labeled by the task to which they correspond. So plan 0 refers to the way in which we perform the subtasks 1–5 of task 0. Similarly plan 3 refers to the way in which we perform 3.1–3.3. There are no plans for subtasks 1, 2, 4 and 5 as these have not been decomposed.

Reading the plans, we see that not all the subtasks need be performed, and not necessarily in the order presented. Looking first at plan 0, subtask 4 'empty the dust bag' need only be performed when the dust bag is found to be full. As this is put in plan 0, we assume that we may empty the dust bag at any stage including when we first get the vacuum cleaner out or when we put it away. If we know that we only ever notice the bag is full when we are actively using the machine, we might choose to put this subtask within 3 'clean the rooms'. This sort of restructuring, finding the appropriate and meaningful hierarchy, is part of the process of HTA.

Looking now at plan 3, how to clean the rooms, we see that we are allowed to clean the rooms in any order. If the task had been varnishing the floors rather than cleaning them, we would presumably do the hall after the rest of the rooms! Furthermore, we only clean those rooms which need vacuuming. The bedrooms will not get dirty as fast as the hall, so we need not clean them so often. If we wanted to be more precise about when the rooms are cleaned, we could produce a more specific plan:

Plan 3: do 3.1 every day
3.2 once a week
   when visitors are due 3.3
How does one produce such a hierarchy with attendant rules? The process is iterative. Assume for the moment that we have some overall task in mind, such as house cleaning. We then ask, what subtasks must be accomplished in order to perform the main task? To answer this question we refer to various sources: direct observation, expert opinion, documentation and so on. These sources will be discussed later in Section 15.6. We then look at each subtask and seek to subdivide it, and so on.

One could go on with this process indefinitely, so one applies some form of stopping rule in order to decide when the tasks are basic enough. The level at which we do this will, of course, depend on the purpose of the task analysis. For example, imagine we were looking at a chemical plant and had produced a first-level decomposition of what to do in an emergency:

0. in an emergency
   1. read the alarms
   2. work out appropriate corrective action
   3. perform corrective action

If our ultimate aim is to install computer monitoring of the plant, we would be interested in expanding tasks 1 and 3. On the other hand, if the aim is to produce online operations manuals, then it is task 2 which would require expansion. In fact, at this high level of task description the analyst would probably expand all the subtasks as she ought to take a somewhat larger view. However, one would obviously put more effort into those subtasks which are directly relevant to the intended purpose.

A rule, which is particularly appropriate when the aim is to design training materials, is the P × C rule. This says that if the probability of making a mistake in the task (P) multiplied by the cost of the mistake (C) is below a threshold, then stop expanding. That is, simple tasks need not be expanded (because no one needs training), unless they are critical.

Another obvious stopping point is where the task contains complex motor responses (like mouse movement) or where it involves internal decision making. In the first case, decomposition would not be productive; explaining how such actions are performed is unlikely to be either accurate or useful. In the second case, we would expand if the decision making were related to external actions, such as looking up documentation or reading instruments, but not where the activity is purely cognitive. A possible exception to this would be if we were planning to build a decision support system, in which case we may want to understand the way someone thought about a problem in order to build tools to help. However, it is debatable whether HTA is the appropriate technique in this case.

The task hierarchy can be represented diagrammatically as well as textually. Figure 15.2 shows a task hierarchy for making a cup of tea. The main task, 'make a cup of tea', is decomposed into six subtasks. Of these only the first, 'boil water', is expanded further. The remaining tasks 2–6 and the subtasks of 1.1–1.4 are underlined showing that the analysis has been deliberately stopped at that point. This obviously denotes the same information as the textual form, but may be more accessible at a glance.
Having produced a first stab at a task hierarchy, one would examine it for errors or omissions. One way of approaching this would be to describe the steps in the task hierarchy to a domain expert. This would quickly show that the plan for making tea has a significant error — it forgets to warm the pot. This has to be added between tasks 2 and 3.

It is the nature of expert knowledge that obvious things get missed for a task description. One way the analyst can search for such omissions is by examining the form of the subtasks. For example, 1.4 says ‘turn off gas’, but nowhere does it say to turn the gas on! Probably, this was implicit in ‘put kettle on hob’, but it should be added between tasks 1.2 and 1.3. At this point we might notice that the task hierarchy is a little unbalanced. This might be right, but we may have included too many detailed tasks at the highest level. We choose to add a new top-level node ‘make pot’ which would encompass the tasks 3 and 4 and also the new ‘warm pot’ task.

The top-level tasks would now be

0. make a cup of tea
   1. boil water
   2. empty pot
   3. make pot
   4. wait 4 or 5 minutes
   5. pour tea
0. make a cup of tea

plan 0.
do 1
at the same time, if the pot is full 2
then 3 - 4
after four or five minutes do 6

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>boil water</td>
<td>2.</td>
<td>empty pot</td>
<td>3.</td>
</tr>
</tbody>
</table>

plan 1.
1.1 - 1.2 - 1.3
when kettle boils 1.4

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>fill kettle</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Figure 15.2 Hierarchical task analysis: making a cup of tea

Having produced a first stab at a task hierarchy, one would examine it for errors or omissions. One way of approaching this would be to describe the steps in the task hierarchy to a domain expert. This would quickly show that the plan for making tea has a significant error – it forgets to warm the pot. This has to be added between tasks 2 and 3.

It is the nature of expert knowledge that obvious things get missed for a task description. One way the analyst can search for such omissions is by examining the form of the subtasks. For example, 1.4 says ‘turn off gas’, but nowhere does it say to turn the gas on! Probably, this was implicit in ‘put kettle on hob’, but it should be added between tasks 1.2 and 1.3. At this point we might notice that the task hierarchy is a little unbalanced. This might be right, but we may have included too many detailed tasks at the highest level. We choose to add a new top-level node ‘make pot’ which would encompass the tasks 3 and 4 and also the new ‘warm pot’ task.

The top-level tasks would now be

0. make a cup of tea
   1. boil water
   2. empty pot
   3. make pot
   4. wait 4 or 5 minutes
   5. pour tea
Plan 0.

do 1
at the same time, if the pot is full 2
then 3 – 4
after four or five minutes 5

This is almost there: the actions ‘empty pot’ and ‘wait 4 or 5 minutes’ are pretty basic and clearly do not need expansion. Neither do they need to be included within one of the other tasks. We might think that ‘empty pot’ should be in with ‘make pot’, but we can empty the pot whilst the kettle is boiling whereas we have to wait for the kettle to boil to do any of the other tea-making tasks. Similarly, the ‘wait’ node belongs at the top as the pouring of the tea depends on it.

The ‘pour tea’ node is a little anomalous. Is it really so much simpler than, say, making the pot? Perhaps we should expand this node too. We could decompose it into three parts:

5. pour tea
   5.1. put milk in cup
   5.2. fill cup with tea
   5.3. add sugar to taste

Plan 5. 5.1 – 5.2
           if desired 5.3

However, the mention of cups makes us wonder: do we really only want to describe the making of a single cup of tea? Perhaps we ought to allow several cups of tea to be made. To do this we modify the plan to allow repetitions of steps 1–3 for each cup. We could describe this plan in words, or use a simple diagram as in Figure 15.3.

The analyst can choose to use a more formal method of describing the plans, such as one of the dialog notations described in Chapter 16, a simple self-explanatory diagram, or plain text. The choice is very much a matter of taste, except that it would be unwise to use too formal a representation until late in the process.

The modified HTA after all this analysis is given in Figure 15.4. In addition, adding the sugar has been expanded to include asking the guests whether or not sugar is required. Also note that the main goal has been altered from ‘make a cup of tea’ to ‘make cups of tea’.

Plan 5.

\[
\begin{array}{c}
5.1 \rightarrow 5.2 \rightarrow \text{empty cups?} \rightarrow \text{NO for each guest 5.3} \\
\end{array}
\]

Figure 15.3 Plan for pouring tea
We have now seen all the types of plan that are commonly found, most of them in Figure 15.4:

**Fixed sequence**  In plan 3, we always do the same sequence of subtasks.

**Optional tasks**  In plan 0 'empty pot' and in plan 5.3 'add sugar' may or may not be performed depending on circumstances. Sometimes, there will be a choice between several options.

**Waiting for events**  In plan 1, we had to wait for the kettle to boil, and in plan 0 we waited 4 or 5 minutes. The latter, waiting a certain time, is probably more common in real-world tasks, such as process control or office procedures, than in the use of computer software.
Cycles  In plan 5, we repeatedly perform tasks 5.1 and 5.2 until a condition is reached (no more empty cups).

Time sharing  Tasks 1 and 2 could be done at the same time (or at the very least they can be intermingled).

Discretionary  For this we have to go back to the vacuum-cleaning example in Figure 15.1. The person is allowed to clean the rooms (plan 3), in any order and whether or not they need it. Basically, you can keep your house as clean or as dirty as you like!

Mixtures  Most plans are a mixture of these elements. For instance, plan 1 for ‘boiling water’ is largely a fixed sequence but split by a wait.

As we can see, the process is far from straightforward. In common with other task analysis techniques, the quality and form of the final output depends very much on the skill of the analyst. Furthermore, different analysts are likely to produce different results, especially as regards the level of detail. Remember also that there is no single ‘right’ answer – the output of the task analysis should reflect the purpose to which the analysis will be put.

Waiting . . .

In the HTAs in Figures 15.2 and 15.4, there are tasks labeled ‘wait 4 or 5 minutes’, but also in the plan ‘after 4 or 5 minutes’. This is clearly repetitive. The explicit subtask was added because the waiting during tea making is often a ‘busy wait’, perhaps chatting while the tea brews. If the task had been more like sending an email and waiting for the reply, we would not have included the waiting as an explicit subtask and only had ‘when reply arrives’ in the plan. Arguably for the tea making we could have left the ‘wait 4 or 5 minutes’ out of the plan. However, the issue of the timing seems critical for the task sequence, hence belonged also in the plan – task analysis is not an exact science! The fact that tasks often have gaps in them is something we will return to in Chapter 19.

WORKED EXERCISE

PRODUCE A HIGH-LEVEL HIERARCHICAL TASK ANALYSIS SHOWING HOW YOU WOULD FIND INFORMATION ON A WEBSITE. ASSUME THE SITE HAS A SEARCH FACILITY AS WELL AS NORMAL LINKS.

ANSWER  This HTA just shows the main stages. Subtask 1.1 only works if the page needed is one level below the top page. Really, one would like to add a task 1.1.3 to say something like ‘if the information required is not on the new page found through the link then repeat the steps of 1.1 on the current page’.
Can you fix this? See our solution on the web at /e3/exercises/ch15/
Notice how the dialog with the user is mixed up with the rest of the program; calculations are interspersed with input-output. Some of the if then statements represent the system’s choices (amount > max_loan), others represent choices of the user (answer == "yes"). If you are a programmer, you will also have noticed the poor program structure and the use of goto. This is not because the authors are bad programmers. It would be possible to rewrite the program using only structured programming constructs. However, the resulting program would be equally obtuse, and, in general, programs which have to parse are full of nasty structures. In this program, the biggest complication is the check that the answer is either ‘yes’ or ‘no’. Error checking and correction often dominate interactive programs.

Imagine now you have been asked to analyze the dialog in some way: for instance, to list all the possible sequences of user inputs and system responses, or to tell the user how to get the repayments on a 15-year loan. The mixing of user and system choices and the convoluted nature of the program structure make this surprisingly difficult. Alternatively, you may be asked to change the interface style or fit the program with a mouse- and window-based interface — is this difficult? Remember, this is a short program which is almost all interaction with the user and should be relatively easy. Imagine a program of 10,000 or 100,000 lines. Various commercial applications began their life on traditional text-based terminals, but are now available on Windows or Macintosh platforms. The ancestry of such programs is often all too obvious — not really surprising.

This gives us two reasons for using a separate dialog description notation: ease of analysis and separation of the interface elements of the program from the actual calculations (semantics). These reasons both presuppose the program exists already — a third reason for using a special notation is to write down the dialog before a program is written. This allows the designer to analyze the proposed structure, or perhaps use a prototyping tool to execute the dialog. A dialog notation is also a way for members of a design team to talk about the design and eventually for the designer to pass on the intended dialog to the programmer of the actual application. Thus dialog notations often form an integral part of prototyping methodologies and tools (which were discussed in Chapter 6).

### 16.3 DIAGRAMMATIC NOTATIONS

Diagrammatic notations are heavily used in dialog design. At their best they allow the designer to see at a glance the structure of the dialog. However, they often have trouble coping with more extensive or complex dialog structures. Sections 16.3.1—16.3.4 describe variants of state transition networks, which are the most heavily used diagrammatic notation. As part of this description, several issues will be discussed which are shared by other diagrammatic and textual notations, in particular the treatment of concurrent dialogs and pre-emptive features. Sections 16.3.5—16.3.8 describe other diagrammatic notations: Petri nets, Harel’s state charts, traditional flow diagrams and JSD diagrams.
16.3.1 State transition networks

State transition networks (STNs) have long been used for dialog description, the first uses for specification dating back to the late 1960s [258, 279] with executable tools developed from the late 1970s on [366, 367].

Consider a simple mouse-based drawing tool. It has a menu with two options, 'circle' and 'line', and a drawing surface. If you select circle you are allowed to click on two further points on the drawing surface. The first of these is the circle’s center and the second any point on the circumference. After the first point is selected, the system draws a ‘rubber band’ line between the center and the current mouse position. After the second point is chosen, the circle is drawn.

The 'line' option in the menu is to draw a polyline. That is, the user can select any number of points on the drawing surface which the system connects with straight lines. The last point is denoted by a double click on the mouse. Again the system ‘rubber bands’ between successive mouse positions.

Figure 16.1 shows an STN describing the tool. Each circle denotes a ‘state’ the system can be in. For example, Menu is the state where the system is waiting for the user to select either ‘circle’ or ‘line’ from the menu, and Circle 2 is the state after the user has entered the circle center and is waiting for the point on the circumference.

Between the states are arrows, the transitions. These are labeled with the user actions that triggered the particular transition and the response the system makes. For instance, state Circle 1 is where the system is waiting for the user to select the circle’s center. If the user clicks on a point, the system moves into state Circle 2 and responds by drawing the rubber band between the point and the current mouse position. From this state, the user can click on another point, upon which the system draws the circle and then moves into the special Finish state. We can see from this that the STN is able to represent a sequence of user actions and system responses.

When in state Circle 1, the user has no other options: there is only one arc coming from it, corresponding to selecting a point. In other states, the user has several options. For example, from state Menu the user can select ‘circle’ from the menu, upon which the system moves into state Circle 1 and highlights the ‘circle’ option on

Figure 16.1 State transition network for menu-driven drawing tool
Figure 16.2 Portion of STN allowing multiple circles to be drawn

the menu, or alternatively, the user can select 'line' from where the system moves into state Line 1. That is, the STN is able to describe user choice.

There is a choice from state Line 2 also: the user can double click on a point and finish the polyline, moving to the Finish state, or he can single click, which adds a new point to the polyline. In the latter case, the transition points back into state Line 2. This represents iteration – the system stays in state Line 2 accepting any number of points on the polyline, until the user double clicks on a point.

Iterations need not involve just one state. The dialog as it stands only allows you to draw one circle. You presumably have to go through the menu selection again for each circle drawn. We could imagine altering the dialog to allow any number of circles to be drawn. To do this, we would make the arc from state Circle 2 loop back to state Circle 1. This is shown in Figure 16.2. There are some problems with this arrangement as it stands which we will discuss later. However, note that already we are using the STN to discuss different dialog options.

16.3.2 Hierarchical state transition nets

The Start and Finish states are not real states, but are there merely to let us glue this bit of dialog into a bigger dialog. For example, the drawing tool may have a main menu, from which we can select one of three submenus: a graphics menu (as described for circles and lines), a text menu (for adding labels) and a paint menu (for freehand drawing). We could describe this complete system using the hierarchical STN in Figure 16.3. This is like the previous STNs, but has additional composite states represented as rectangles with a picture of a little STN in them. Each of these rectangles denotes the whole STN for the relevant submenu. We assume that the STN in Figure 16.1 is the Graphics submenu. In a large specification this may be represented by a caption for the STN or by putting the label Graphics submenu in the Start state.

To read this diagram, we start in state Main menu and follow the relevant transition from it as before. Imagine the user has selected 'graphics' from the main menu. The system responds by 'popping' the graphics submenu and then going into state Graphics submenu. However, this is not really a single state, but corresponds to the STN described in Figure 16.1. We therefore enter this subdialog at the state pointed to
by its special Start state, that is the Menu state. We then follow through the graphics menu STN, drawing either a circle or a polyline. When we get to a Finish state we revert to the main diagram in Figure 16.3 and follow the (single) arrow from the Graphics submenu state which leads us back to the Main menu.

The use of hierarchical elements does not change the power of the basic notation as one can simply imagine gluing the subdiagrams into the main diagram. However, it makes it far simpler to specify large systems; it would not be unreasonable to specify a whole system dialog in this fashion, from the highest level down to individual keystrokes and mouse clicks.

**DESIGN FOCUS**

**Using STNs in prototyping**

Producing a state transition network can be a good start for prototyping. In the simplest case you can use paper-based prototypes. For each state in the chart, draw and label a representative screen on paper, either by hand or printed from a computer drawing package. You can then run through example scenarios with a potential user or client. If the user asks what happens if a particular button or key is pressed, you can simply consult the STN, look up the current state on the chart, see what the next state ought to be and then show the relevant piece of paper.
Alternatively, you can get the computer to do the work for you! Using a multimedia authoring tool or prototyping tool such as HyperCard or Macromedia Director draw each state as a separate screen, but leave a blank area for annotation and additional controls (see Figure 16.4). Name the screens using the same labels that you used in the STN. Now add buttons or active areas to each screen corresponding to the buttons on the intended final system and simply link them to the corresponding screen. (This is particularly easy to do in HyperCard.) Some user actions do not correspond to clicking areas of the screen. For these add extra buttons on the blank area and label them suitably, for example 'user types some text'.

Now you give this to your prospective users; they can click on buttons and see for themselves what will happen. Remember to warn them that this is a series of fixed screens not a functioning prototype. In particular, if you return to the same point in the dialog, the screen will be the same as the first time you were there.

Figure 16.4 shows an example of this sort of prototype based on the STN in Figure 16.1. It is built using HyperCard and can be downloaded from our website.

There are other ways of making STNs hierarchical. Each variant has its own rules for tying the high-level STN with the detailed STNs. In addition, the conditions which enable a transition and the system’s responses may be attached to a low-level STN. Generalized transition networks are probably the most well known of such variants as they are used to describe the computer system’s behavior in CCT (Chapter 12, Section 12.2.2).
16.3.3 Concurrent dialogs and combinatorial explosion of states

We have seen that STNs can be very good at representing the sequential, choice and iterative parts of a dialog. Where they fail, quite dismally, is in describing a dialog consisting of several concurrent parts. Take, for example, a simple dialog box for describing text style as one might find in a word processor (Figure 16.5). The box contains three toggles, one each for bold, italic and underline styles. A piece of text can be emboldened, italicized, underlined or any combination of these three. To select, say, emboldening, the user clicks over the bold toggle. To deselect it, the user simply clicks again.

If we look at each toggle individually, we have simple two-state STNs as in Figure 16.6. The arrows have been drawn with two heads, as the same user action moves you in either direction between the states. We have also omitted the system responses, which would be to invert the highlighting of the toggle, and possibly to change the style of any currently selected text in the document.

![Text Style Diagram](image)

**Figure 16.5** Simple dialog box with three toggles

![Diagram](image)

**Figure 16.6** Individual bold, italic and underline state transition diagrams
However, this does not tell us what happens if, say, the user clicks over the italic toggle and then the bold one. To do this we need to combine the diagrams. We’ll do this first for just the bold and italic options. This is shown in Figure 16.7. This has four states: one with neither style selected, one for bold only, one for italic only and one for both. You can verify that each user action performs as expected: for example, clicking over ‘italic’ whilst in Bold Italic would get you to the Bold Only state.

Finally, we add in the underline style in Figure 16.8. This time we have written the user actions simply as ‘B’ for ‘click over “bold”’, and so on, as the diagram has
become cluttered enough as it is! Again, you can verify that the user actions perform as expected: for example, 'U' in state _Bold Italic_ takes you to state _Underlined Bold Italic_.

The STN with two toggles had four states, the STN with three toggles had eight states and, in general, if we had had _n_ toggles, we would have had a diagram with $2^n$ states in it – not particularly easy to read! The problem is that the user is effectively operating the toggles concurrently, he can perform an action on any of them, and the actions on one are independent of the actions on the others. If we have two STNs with _m_ and _n_ states respectively, then the STN representing the two acting concurrently will have _m_ × _n_ states. Furthermore, the resulting diagram would hide the regularity of the interface.

This inability of STNs to handle concurrent dialogs is particularly a problem with direct manipulation interfaces. These are often full of toggles, option switches, style sheets, etc., all of which can be operated independently of one another. This seriously calls into question their usefulness under these circumstances.

One suggestion, particularly associated with Jacob [191], is that STNs should be used to model the microdialog of direct manipulation systems. That is, each interface element (menu, toggle, dialog box) would have an associated STN. However, the way that these are put together would use some alternative notation. Thus, for example, the above dialog box would be represented as consisting of three STNs, as we originally had in Figure 16.6.

### 16.3.4 Escapes and help

Escapes and help systems pose problems that are similar to the combinatorial explosion from concurrent dialogs. Imagine that we have been observing the use of the drawing tool. We have noticed that users often find they have wrongly selected some option and want to get back to the menu. As the dialog is currently specified, once they select, say, the circle option, they must select two points before they are allowed to continue. As the current system does not have any deletion at present, this was found to be particularly irksome.

As a solution to this problem, we want to add an escape key, which, wherever you are, cancels what you are doing and returns you to the main menu. This seems quite a simple addition – it only took a sentence to say. However, to add it to the STN describing the system would require an arc from _every_ state back to the main menu. Furthermore, this would make a complete mess of the hierarchical structure of the dialog description.

Some forms of hierarchical STN explicitly cater for this by saying that if a composite state has a labeled arc coming from it, then this acts as an escape from the subdialog. For example, we could redraw the overall system description as in Figure 16.9. Each submenu state now has two arcs coming from it. One arc is labeled 'normal Finish' and represents the path taken when the subdialog reaches its _Finish_ state. This arc has a little state circle added to it in order to emphasize that it is tied to the _Finish_ state within the subdialog. The other arc from the submenu state is
Figure 16.9  STN for drawing tool with escapes

labeled ‘ESC’ and this represents the user hitting the escape key. The difference is
that this arc is ‘active’ at all times during the subdialog. Even when the user is in the
middle of drawing a circle or a line, if the escape key is pressed, the subdialog is
immediately aborted and the arc labeled ‘ESC’ is taken. In this case, both the ‘ESC’
and the ‘normal Finish’ arcs go to the same place. In general, this need not be so, and
one may have several escapes activated by different user actions.

Help systems are similar to escapes in some ways, in that they may be invoked at
any stage during the dialog. However, unlike escapes, when you have finished using
the help system, you expect to return to the same point in the dialog that you left.
That is, you can think of the help system as being a little subdialog hanging off every
state in the network. Figure 16.10 shows this for two states of the dialog: as you can
imagine, it would get a little tedious for the whole thing!

The case of a help system is very similar to concurrent dialogs (in fact, it is an
embedded dialog), and the total number of states in the full diagram is again the pro-
duct of the number of states in the help system times the number in the original system.

16.3.5 Petri nets

One of the oldest formalisms in computing science is the Petri net. It is a graphical
formalism designed for reasoning about concurrent activities. In recent years it has
been used by several researchers to specify aspects of single-user [276] and multi-
user systems [277]. In an STN the system is always at exactly one state. Indeed, you
can simulate the behavior of the system by moving a counter around the STN following arcs. A Petri net is similar except that the system has several 'states' at once. These are depicted as several black counters in Figure 16.11.

The figure shows a Petri net for a system having two near independent bold/italic toggles. The circles are called places (like states) and the thin rectangles are called transitions. There are three counters: one in the 'Bold On' place, showing that the bold toggle is currently on; one in the 'Italic Off' place showing that the italic toggle is currently off; and one in the slightly elliptic 'user presses Bold' place showing that the user has just pressed the bold toggle. The slightly elliptic places represent those where user input can occur. The rule is that if all the places with arcs going into a transition have a counter then the transition can fire and all the counters from the input places are removed and new counters placed on the outputs of the transition. So, in Figure 16.11, there are counters in both the input places of transition 'T1' which will therefore fire leaving a single counter in place 'Bold Off'.

The italic side is similar except it has an extra arc coming from the 'Bold On' place to the transition 'T3', but with a small circle rather than an arrowhead. This is an inhibition arc and means that the transition 'T3' cannot fire if there is a counter in