Chapter 5 MapTutor: Implementation and Trace

5.1 Introduction

This chapter is mostly dedicated to MapTutor's graphical interface. It also presents specific details of the implementation of MapTutor. **Section 5.2** presents the ideas that guided the design of the program. **Section 5.3** sets out a description of MapTutor's interface and its components. **Section 5.4** describes the strategies and data used by the program in order to keep track of the learner's actions. **Section 5.5** present the tutorial and help programs — two separate (but not independent) auxiliary programs used by MapTutor to present tutorial and help, respectively. **Section 5.6** describes how the learner should use MapTutor effectively. **Section 5.7** presents details about how the program intervenes and delivers feedback messages. **Section 5.8** discusses the utility of the end-of-session reports created by MapTutor. **Section 5.9** describe how the knowledge should be entered into MapTutor. Finally, **Section 5.10** presents some implementation details concerning the language and platform utilised.

5.2 Design Considerations

The construction of the interface should be the first step in the implementation of any non-trivial, highly interactive program because it helps the designer to have the first impressions about what the final product will look like, and also she will be able to anticipate which functions the rest of the program will have to provide to support its interface, and vice-versa. Anderson et al. (1992) suggest that the interface of a computer based tutor should be designed in such a way that it would allow the learner to resolve the problem posed to her without any help from the computer tutor itself. It could also be added that the interface must carry out some low-level functions, such as translating the states of the problem and doing bookkeeping of the objects, on behalf of other parts of the tutor. This latter function has been crucial in the design and implementation of MapTutor's interface.

The design of MapTutor consisted basically in determining which functions would be necessary to implement those capabilities specified in previous chapters and the support necessary for providing the learner with a friendly environment to resolve the problem without additional load. The latter means that the

program should be nearly as easy to use as its conventional counterpart (i.e., paper-and-pencil mapping). Moreover, the program should also be easy to learn to operate, and this would include providing facilities such as on-line help and tutorial. In MapTutor, these facilities constitute, in fact, separate (but not independent) applications which have been integrated seamlessly with the main program by means of an interapplication communication protocol (see details in **Section 5.5**).

MapTutor's interface carries out unnecessary details of the problem solution which the learner is expected not to have difficulties to learn (see Anderson et al., 1992). These details include, for example, drawing concepts in the mapping pane on behalf of the learner and highlighting a selected concept once the learner has clicked on a piece of text which contains the desired concept. Moreover, the interface provide mapping tools which make the program much easier to use than, for instance, by using menus. MapTutor's interface will be fully described next.

5.3 MapTutor's Interface

At the very beginning of this research project, it was determined that MapTutor' interface should have the following components:

- **Concepts Pane** — containing a set of pre-selected concepts extracted from the to-be-mapped text.
- **Links Pane** — containing a set of canonical links.
- **Map Pane** — where the maps are drawn.
- **Tutor Pane** — which would be responsible for delivering feedback messages.

Had MapTutor stuck to this design, it could face problems similar to those Feifer (1989) had with multiple interpretations of word-concepts (i.e., icons in his notations). Then, the *Concepts Pane* described above was replaced by a *Text Pane*. Surprisingly, it later became clear that include the text into the interface of MapTutor would also bring some advantages such as:

- it allows MapTutor to address the selection issue (see **Chapter 1**) in a very natural way, that is, it enables the learner to indicate concepts in their textual context; and
- when a certain tutorial intervention needs to refer to parts of the text at hand, it can easily do so by highlighting the respective piece of text in that pane.

Figure 5–1 shows the opening screen of MapTutor. It shows that the program has four active windows[1]:

- **Main window** — containing the **map pane**;
- **Text-and-links window** — containing the **map pane** mentioned above, and two panes related to the set of canonical links used by MapTutor: the **links pane** — containing a set of buttons corresponding to the set of links utilised, and the **links definitions pane** — containing definitions of each link in the set of links.
- **Tutor window** — which contains the **tutor pane** represented by an icon.
- **Tools window** — which contains two associated panes: the **tools pane** — containing a grid with the *mapping tools* plus a tool where the learner can ask for help, and the **tools definitions pane** — a small pane situated below the tools grid whose sole purpose is to provide definitions of each of these tools.

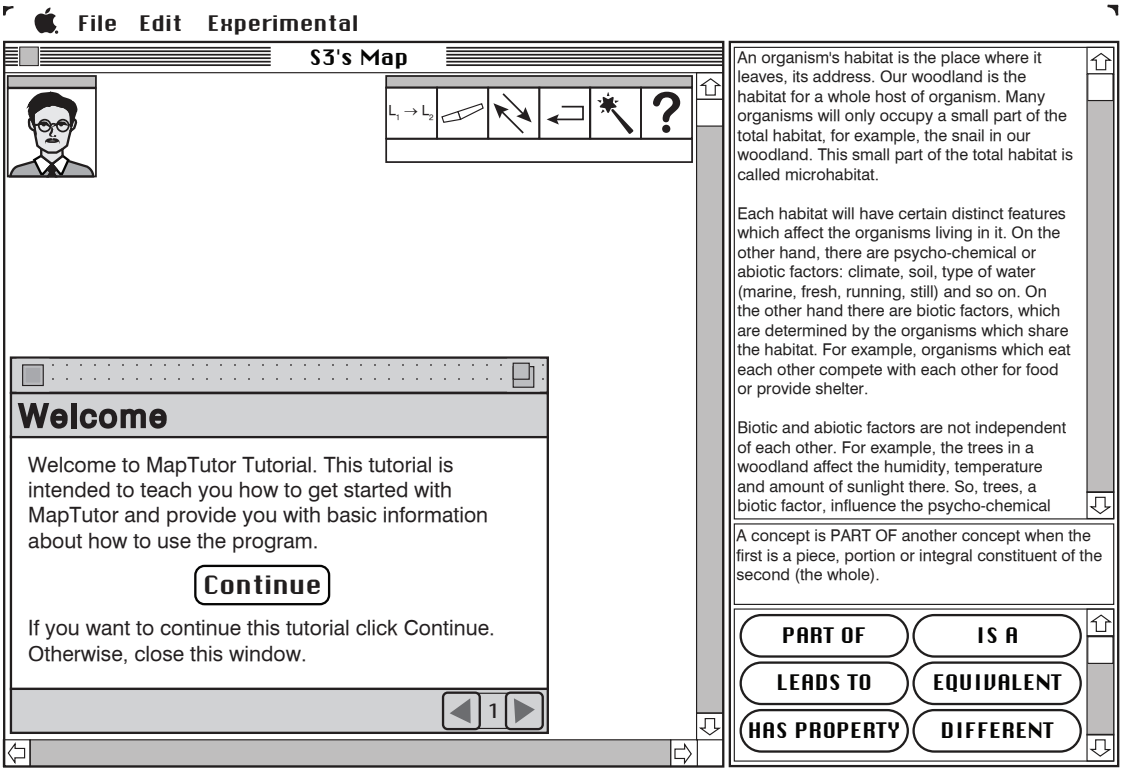

Figure 5–1: MapTutor's Opening Screen

In **Figure 5–1**, the text and links window is on the right hand side. The main window is the largest one. At its left-top side, is the tutor window, and at its

[1] Notice that the window over the left-bottom portion of the main window does not belong (directly) to MapTutor, but to its associated tutorial program.

right-top side is the tools window. The dialogue window at the bottom belongs to the tutorial program.

Figure 5–2 shows the hierarchy which holds among the interface elements (views) of MapTutor. This hierarchy should not be confused with the box named **Structured Interface** in **Figure 3–1**, in **Chapter 3**, which briefly describes how some of these views communicate with each other. The visual hierarchy provides a panoramic visualisation of all the visible objects (i.e., views) which MapTutor knows about. The visual hierarchy is based upon the notion of *enclosures*, so that a given view in the hierarchy encloses its descendants (if any) and is enclosed by its ancestors (if any). For example, in **Figure 5–2**, the **Links Pane** encloses the **Link1 Pane** — one of its descendants, and is enclosed by the **Text** and **Links Window** — its ancestor.

MapTutor's interface is event-driven, and almost all input events are mouse-based ones. This means that the program basically does not handle keystrokes because they are unnecessary for the learner to resolve the problem posed to her (i.e., drawing a graphical map)[2]. There are two types of mouse events handled by MapTutor:

1. **Mouse movements** — these events are responsible for changing the form of the cursor to provide visual clues which suggest the operation which can be carried out by clicking the mouse on the region the cursor is over. **Table 5–1** presents the several cursor shapes used by the program and when they are applicable.

2. **Mouse clicks** — these events are not only responsible for changing the state of solution of the problem at hand, but they can also invoke other auxiliary functions, such as asking for help. The effects of clicks on the several views of the interface are presented in **Table 5–2**.

When the cursor is over any part of the interface other than the ones listed in **Table 5–1**, the cursor is set to an arrow shape. This is the default shape and indicates that clicking on any interface view while the cursor has this shape has no effect for the solution of the problem.

Notice that the actions described in **Table 5–2** refers to good clicks, that is, clicks which comply with MapTutor's requirements. For example, if the learner

[2] As a matter of fact, all keystrokes (except, of course, when the program asks for the student's name at the beginning of a new session) handled by MapTutor are redundant and are still kept either for compatibility reasons or to offer keyboard short-cuts. The latter are intended to be used by advanced users.

attempts to draw a link which does not lead to any concept, the link will be disallowed and no action will be produced.

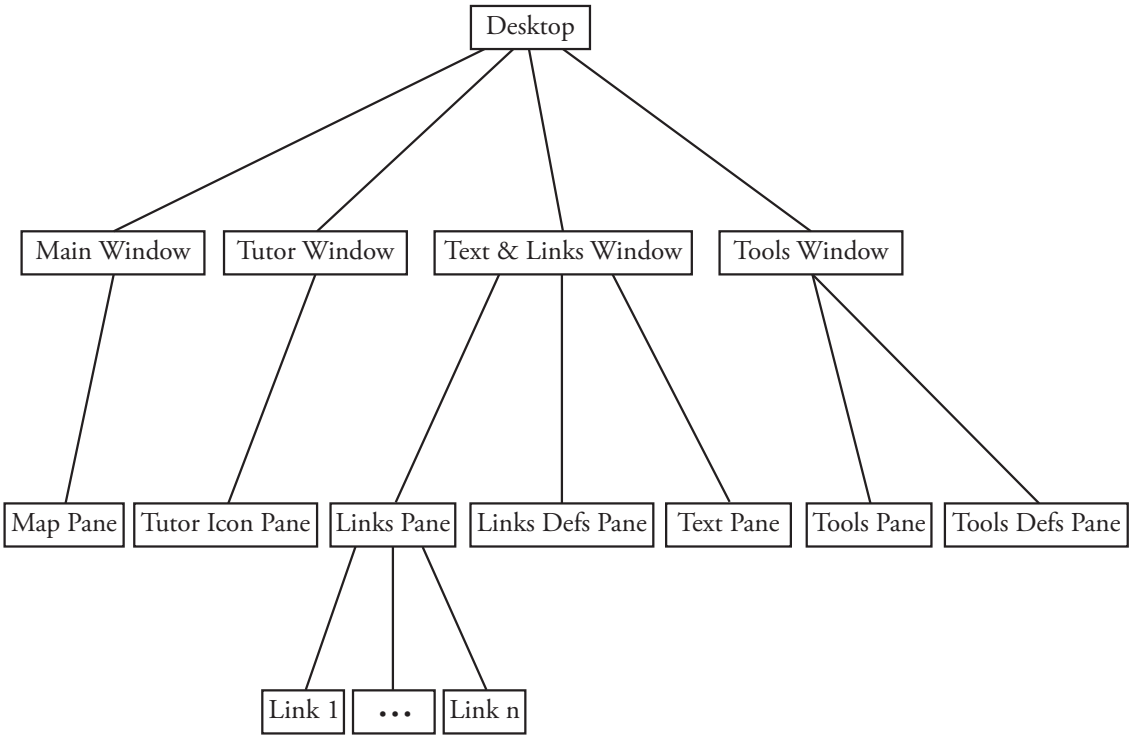

FIGURE 5–2: MAPTUTOR'S VISUAL HIERARCHY

In addition to changing cursor shapes, MAPTUTOR provides some other visual clues. For example, an animated marquee[3] around a selected concept in the map pane is a visual clue which indicates that the concept can be moved to another position in this pane (see details below). To have this marquee appear around a given concept the learner must click on the corresponding concept in the text. Likewise, the program draws a marquee around the name of a selected link in the map pane to indicate that subsequent actions invoked by using the mapping tools will affect this link (see details below).

When the cursor is over...	It changes its shape to...
The **Text Pane**	**finger cursor** — indicating that a concept can be selected in the text.
A button in the **Links Pane**	**finger cursor** — indicating that a link name can be selected by clicking on this button.

[3] This marquee 'is a rectangle with edges that appear to be sliding around the perimeter, like an old fashioned nightclub sign' (Knaster & Rollin, 1992, p. 237). It is also known as *marching ants* in the computer graphics jargon.

When the cursor is over...	It changes its shape to...
The **Tutor Pane**	**question-mark cursor** — indicating that some feedback message can be provided if this pane is clicked on. (Only when the tutor has some queued message to deliver.)
A selected concept in the **Map Pane**	**hand-grabber cursor** — indicating that this concept can be move around the map pane.
Close to a concept in the **Map Pane** (or inside a non-selected one)	**pencil cursor** — indicating that a link can be drawn to connect this concept to another.
An icon in the **Tools Pane**	**finger cursor**— indicating that an action can be executed by clicking on this icon.

TABLE 5–1: CURSOR SHAPES USED BY MAPTUTOR

When the click is on...	The result is...
A piece of text containing an important concept	the concept is drawn in the map pane, or if it has already been drawn, it is selected in that pane.
A button in the **Links Pane**	the link name becomes the active one, and all links subsequently drawn in the map pane will have this name until a new one is selected.
The **Tutor Pane**	if available, feedback is provided.
A selected concept in the **Map Pane**	the concept can be move around the map pane.
Close to a concept in the **Map Pane** (or inside a non-selected one)	a link originating at this concept can be drawn.
An icon in the **Tools Pane**	the action corresponding to the icon is executed.

TABLE 5–2: THE EFFECT OF MOUSE CLICKS

5.3.1 The Text Pane

In the text pane, the learner is expected to select concepts for including in her map in the map pane. Thus, this pane addresses the issue of text selection, which all learning strategies for reading comprehension require. The use of this pane also eliminates, or at least reduces to a minimum, the problem of multiple interpretations of concepts by tying boxes representing concepts in the learner's graphical map with word-concepts in the text. Physically, this is achieved by only allowing that concepts be moved or drawn in the map pane after one of its respective occurrences in the text has been selected (i.e., clicked on).

MapTutor determines when the learner has selected a concept in the text after clicking on somewhere in this pane by examining the slot *where in text* (see **Section 3.7.1**) for each concept. When the learner's click hits a given concept of interest, the program highlights it in the text, and either draws the concept in the map pane — if it has not already been drawn — or otherwise, just selects it in the map pane by drawing a marquee around it, as described above. Although the program does not allow the learner to draw a concept directly in the pane, selecting a concept for the first time in the text pane works as if she asked the program to draw the concept on her behalf. After all, drawing a box containing a concept's name is not a crucial issue in graphical mapping. If the learner's click on the text pane does not hit any concept relevant to the learning objectives, MapTutor will present a warning message (upon the learner's request only — see **Section 5.7**) saying that there was no interesting concept where the she just clicked.

An alternative way of selecting concepts in the text was initially tried out. It consisted of allowing the learner to make the desired selection by dragging the cursor (as most modern word processors allow), and then checking the selected piece of text to verify whether it is relevant to the learning objectives. Although this option looks quite intuitive and natural, since it resembles hand-made highlighting, it has three potential drawbacks in relation to the one finally chosen: (1) from a technical point-of-view, it is much more difficult to implement; (2) from a practical point-of-view, it is a bit harder for the learner to execute too; and (3) from a pedagogical standpoint, there is no guarantee that such a way of text selection will provide the learner with any cognitive gain. The alternative adopted does not guarantee any learning gain either, but at the very least it is both easier to implement and use.

The text pane is also used by MapTutor's teaching procedures to provide feedback to the learner. For example, when the program's feedback involves inferences

of a given relationship in the text, it will show (by highlighting it) where in the text the information can be found (see **Section 4.9.1**). **Figure 5–11** shows another example of text reference. There, MapTutor is asking the learner for confirmation about a suspicion raised by the diagnostic process, and it uses the text to create a context for the question.

5.3.2 The Links Pane

In the links pane, the learner can choose link names for use with the links she draws in the map pane. She is expected to choose a link by clicking on the respective button containing the link name. When a link name is selected it becomes the current link name and will stay so until a new name is selected. Each link drawn in the map pane gets its name from the current link name. The button corresponding to the current link name remains highlighted, so that the learner will know which link name is the current one.

The links pane has been designed to fit any number of links an instructional designer wishes to introduce into the system. Despite this, it is not expected (nor recommended) to have a large number of links (say, more than a dozen) input into the system, because typically, a canonical link system does not have such a large number of links (see **Section 2.4**).

5.3.3 The Links Definitions Pane

The only function of the links definitions pane is to provide the definitions of the links laid below (see **Figure 5–1**). The definition of a given canonical link appears automatically in this pane whenever the cursor is over the respective link. If the cursor is moved elsewhere, this pane will present the definition of the current link name (see above), if one has already been selected, or simply the heading *Definitions of Links* if there is none selected.

5.3.4 The Map Pane

The learner is expected to draw her graphical map in the map pane. Thus, it is in this pane where most interesting actions really take place. Each concept in this pane is represented by a rectangle containing the concept's name. A link between two concepts is represented by a line with an arrow at its end point — indicating the direction of the link, and the link's name situated about the middle of the line. MapTutor uses a seemingly natural notation for representing the problem states (i.e., map configurations) on the screen, but the learner is expected to follow some conventions and constraints. For example, the learner is not allowed to draw concepts by herself, but once a concept has been drawn, she can move

it wherever she wants to. Also, links drawn in a map only make sense when they connect two concepts. This means that links cannot ever exist without having a concept at each of its extremities. Moreover, when the learner moves a linked concept, all links attached to it also move along with the concept. These constraints simplify the programming task because they reduce the probability of spurious constructions, such as ambiguous links, and thus make it a bit easier to both interpret and keep track of objects drawn in the map.

MapTutor allows the learner carry out the following drawing tasks in the map pane:

- **Drawing a link** — that is, drawing a line starting at a point inside or close[4] a given concept and ending at another one in the map pane. MapTutor requires that each link has both an origin and a terminal concept at the time it is drawn. Whenever a link is drawn it is selected so that an immediate subsequent operation (e.g., renaming) can be performed over it.
- **Dragging a concept** — that is, moving a box containing a word-concept from a position to another in the map pane. This corresponds to two operations in paper-and-pencil mapping: (1) deleting the concept, and (2) then drawing it again in another place. If the concept being dragged has links originating or terminating at it, the operation will also make these links be moved together with the concept. MapTutor does not allow connected concepts to be dragged alone, because doing so would lead to map configurations containing dangling links and make bookkeeping much more difficult. This design decision also anticipated that this would not be a natural way of moving a concept. Thus, if the learner desires to move the concept without its connecting links, she will have to delete the connecting links before the operation.
- **Renaming a drawn link**. MapTutor allows the learner to rename a link she has drawn in her map by simply selecting the desired link, picking a new name and then clicking on the mapping tool (see **Section 5.3.5**) corresponding to renaming. In a paper-and-pencil map, this operation would correspond to (1) deleting the link's old name, and (2) then writing down a new name. The learner may, obviously, proceed in a similar

[4] A point is considered close to a concept if it is inside a threshold rectangle region around the concept's rectangle. The degree of closeness is given by constant PROXIMITY_THRESHOLD. The bigger this threshold the greater the probability of the point is close to a concept, but the greater the chances of a link being considered ambiguous too. Thus, this constant must be set experimentally so as to reduce the chances of ambiguity and without yet requiring that a link's extremity be necessarily inside a concept to be considered as attached to the concept.

way (i.e., deleting the undesired link and then drawing a new one), but the rename operation is intended to be a short-cut and indeed it seems to be much handier.

- **Inverting a drawn link**. This operation is somewhat similar to the previous one: to invert a link in the map pane the learner must (1) select the link, and then (2) click on the mapping tool corresponding to inversion. To perform this operation in a paper-and-pencil map, the learner would, for example, delete the link's arrow at one extremity and draw a new arrow at the other end in the opposite direction. Like renaming, this operation is intended to be a short-cut, that is, there are other ways of achieving the same result.

MapTutor allows any of the tasks above to be reversed (undone/redone) as many times as wished. Thus, there are altogether eight tasks available (the four ones above plus their reverses), which from the programmer's point-of-view, are rather different (however similar they may appear from a user perspective). Moreover, there are two other map tasks which have already been referred to above: (1) selecting a concept[5] and (2) selecting a link. In fact, these are intermediate tasks which do not have a direct influence over the map being constructed. These operations prepare the selected objects for the execution of one of the operations described above. Selection of a link in the map pane is carried out by clicking on the region around the link, whereas selecting a concept is done in the text as described above[6]. Only one object (concept/link) can be in a selected state in the map pane at any time.

MapTutor provides three other operations which can affect the learner's map. These operations, which are irreversible, are:

- **Tidy It Up**. This operation is provided due to a technical implementation problem: when a link which had been intersecting other links is deleted it will sometimes eat part of the intersecting ones. To correct this, operation *Tidy It Up* simply redraws the eaten pieces (see **Implementation Note 1** in **Section 5.10**). This command is not reversible, but it does no harm anyway.

[5] Selecting a concept in the map pane is conceptually different from selecting a concept in the text pane. Incidentally, by virtue of design, selecting a concept in the map pane is a result of selecting the corresponding one in the text.

[6] Using the interface semantics described thus far, the program cannot allow selection of concepts by clicks on the concept itself in the map pane. By doing so, it would not be able to decide whether the user's intention was selecting the concept or starting drawing a link. The meaning of a click inside a concept would therefore be ambiguous.

- **Clear All Links**. This operation clears all links but leaves the concepts which have already been selected in the text. This operation is not intended to be used by novices, and it is not even mentioned in the tutorial program.
- **Clear All**. This command clears all (concepts and links in) the map. It is useful when the learner wants start again from scratch a new map instead of, for example, fixing the current one. As the previous one, this operation is intended to be used by advanced users only.

Before carrying out the last two operations, MapTutor always reminds the learner about their meanings and that they are not reversible. These operations could easily be made reversible too. But since they are neither expected to be used frequently nor are carried out without previous warnings about their effects, they were made irreversible.

5.3.5 The Tools Pane

The **Tools Pane** is a pane containing a grid with icons representing mapping tools — i.e., tools with which the learner can carry out most of those drawing tasks in the map pane described in **Section 5.3.4**, plus the *Assistant Tool* by means of which she can call the **Help** program (see **Section 5.5**). Each tool has an associated icon to remind the learner of the meaning of the given tool, but just in case, the tools pane also has an associated definitions pane which presents the (written) definition of each tool when the cursor is over the respective icon (see below). The tools provided in the tools pane are:

- **Tool Delete** — this tool is represented by an eraser icon in the tools grid and its action is to delete a selected link.
- **Tool Invert** — this is a command in the tools grid which can be used to invert a selected link. This tool is represented by two inverted arrows in the tools grid.
- **Tool Rename** — this is a command in the tools grid which can be used to rename a selected link. This tool is represented by two L's in the tools grid.
- **Tool Revert** — this is a command in the tools grid which can be used to revert (undo/redo) the learner's last action (of course, provided it is reversible). This tool is represented by a U-turn icon in the tools grid.
- **Tool Tidy It Up** — is a command in the tools grid which can be used to attempt to clean the learner's map. This tool is represented by the magic wand icon in the tools grid.

- **Tool Assistant** — is a command in the tools grid which invokes MapTutor's help program (see **Section 5.5**). This tools is represented by the question mark icon in the tools grid.

These tools invoke the appropriate action whenever they are clicked on, but the action itself is only executed when it is possible. For example, if the learner clicks on the *Rename Tool*, without having chosen a new name to replace the desired link's name, there will be no effect at all.

5.3.6 The Tools Definitions Pane

The **Tools Definitions Pane** is a small pane situated below the tools grid. It simply presents short definitions of the tools laid above it. A definition is presented whenever the cursor is over an icon representing a tool. This pane is intended to prevent those icons from being misinterpreted.

5.3.7 The Tutor Pane

The tutor window is a small window with an icon inside the tutor pane[7]. This pane is initially situated near the top left of the main window. It can be moved around the screen at the learner's best convenience but it can never be closed. The intended metaphorical meaning of this pane is to provide both a place where the learner can read (automatic or upon request) feedback messages and visual feedback as to whether the program has something to tell the learner (see **Section 5.7**). Thus, the only function of this pane is to deliver feedback messages and visual clues for the learner.

5.3.8 Pull-Down Menus

MapTutor has pull-down menus which complement the application's environment, but they are not expected to be used very frequently. The menu options offered by the program are:

- **File Menu** — this menu contains the standard file manipulation items (e.g., New, Open, Save, etc.).
- **Edit Menu** — this menu contains (redundantly) all those mapping tools describe above plus commands **Clear All** and **Clear All Links**.
- **Experimental Menu** — this menu (see **Figure 5–3**) should not be used by a learner. It is intended to be used by the experimenter in the experimental stage of the program. This menu contains the following items:

[7] The distinction between window and pane is not important here. Thus, this whole interface element will henceforth be referred to as tutor pane.

◊ **Create In-Session Report** — used to create a report about the objects drawn in the map pane. It is only useful for debugging the program and adjusting drawing parameters.

◊ **Show Knowledge Base** — when chosen, creates a report about the current knowledge base of the program. It was used to create **Appendices A** to **C**.

◊ **Shut Up Tutor** — used to make the tutor silent when the programmer is testing other parts of the program than those responsible for delivering feedback. If used by a learner, the program will behave like a structured interface only (in the sense of **Section 5.2**).

◊ **Ask Eval Questions** — an attempt to ask evaluation questions about the correctness of diagnosis and usefulness of feedback while the subjects of the experiments are engaged in drawing their maps, but this option turned out to be very annoying for the learner.

◊ **Enable Tests** — used by the programmer to test any procedure in the debugging stage.

◊ **Diagnostic Strategies** — this item contains a sub-menu which allows for changing the current diagnostic strategy in real-time (i.e., during a tutorial session). It has also served for testing the behaviour of the program when using the various strategies. The sub-menu items correspond to those ordering relations defined in **Chapter 4**. As seen in that chapter, there are altogether six such relations and they are identified in this sub-menu as a sequence of characters which resembles the respective relation (see **Figure 5–3**). For example, the option ML < MC < MR stands for the ordering relation,

$$\textit{MisunderstandsLinkMeaning} \prec \textit{MisunderstandsConcepts} \prec \textit{MisunderstandsRelationship}$$

In addition to the ordering relations defined in **Chapter 4**, there is another option — called *Ask Learner* — by using which the program will always ask the learner in order to decide about an ambiguous preliminary diagnosis (i.e., when more than one basic diagnostic procedure succeeds — see **Section 4.5**, in **Chapter 4** for more details). The diagnostic strategy currently in use corresponds to the option which is check-marked in this sub-menu.

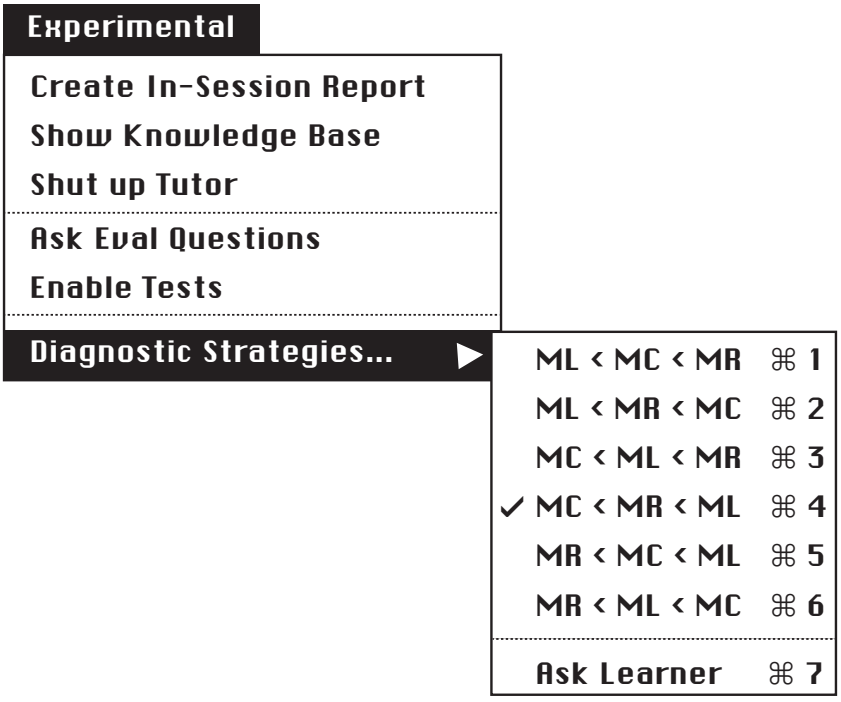

FIGURE 5–3: MENU USED IN EXPERIMENTAL STUDIES

5.4 Interpreting and Keeping Track of the Learner's Actions

Interpreting and keeping track of objects drawn in the map pane is one of the hardest parts of the implementation of MAPTUTOR. These functions refer to identifying what the learner has actually done and how her last action affected her map. Moreover precisely, this suite of procedures is responsible for identifying object locations (e.g., where in the map pane a given concept lies) and how they fit together (e.g., by determining which concepts a given link connects). Although bookkeeping is closely related to diagnostic evaluation, these are rather distinct functions. In fact, the bookkeeping functions described here precede diagnosis, and the diagnostic procedures described in **Chapter 4** are called only when the learner's action results in a valid new link[8]. Although interpreting learners' inputs is much easier than (say) interpreting natural language inputs, it is not as easy as it may appear at first sight.

Defining what should constitute legal moves is the first step of designing the interpretation module. A legal link for MAPTUTOR is one which connects two concepts in an unambiguous way without duplication. By contrast, an anomalous link is dangling — i.e., it does not connect two concepts; duplicated — i.e.,

[8] In this sense, a link which has been renamed or inverted is considered as a new one, as if it had been deleted and then drawn again.

there is another link connecting the same concepts; or ambiguous — i.e., the program cannot tell which concept the link is pointing to/from[9].

As MapTutor does not have a general perception of the objects presented on the screen (i.e., it does not interpret the map as a whole image), it has to keep track of each object separately. Anomalous links could lead to wrong diagnosis, and thus MapTutor only analyses legal ones. Therefore, spurious constructs must be avoided as early as possible. For example, ambiguity is introduced in the map being constructed when, as a result of an operation in the map, the program cannot tell which concept a given link nearby is pointing to/from. **Figure 5–4** shows such an example of link ambiguity. In this figure, the program cannot decide whether link is a points to concept habitat or to concept place. Thus, when the learner draws an ambiguous link (from the program's perspective), MapTutor will ask her to clarify this link before proceeding. In the example presented in **Figure 5–4**, the program would present a dialogue box asking the learner to choose among the options: habitat, place or abort the link (see **Figure 5–5**). If the learner does not clarify the ambiguous link the program will abort it anyway.

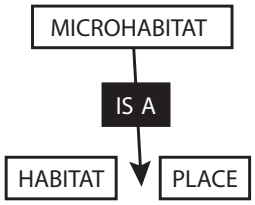

Figure 5–4: Example of an Ambiguous Construct

The drawback of the dialogue window in **Figure 5–5** seems to be that it hides the construction under scrutiny. MapTutor alleviates this problem by warning the learner earlier that there is an ambiguous link and that it will soon afterwards ask for clarification.

According to a generally accepted engineering methodology, when designing, we should always bear in mind the worst possible situation. In the current context, this means that we should not design for a well-behaved learner. How to avoid anomalous links then? Otherwise, if this is not possible or desirable, how to reduce or manage the anomaly space? The easiest answer seems to be, do not allow spurious links to be drawn altogether. Taking this too rigorously, however, may frustrate the learner (e.g., 'Why have so many of my link been disallowed by the program?'). Thus, a compromise should be reached that the program would categorically reject anomalous links only when they jeopardise the diagnostic

[9] Note that these features are not mutually exclusive. For instance, a link may be duplicated and ambiguous.

and teaching processes. In such a case, the program must either tell the learner why her link has been rejected, or give her a chance to repair the problem before doing so. But what are the anomalous cases which really jeopardise the whole process? The first obvious candidate is ambiguity.

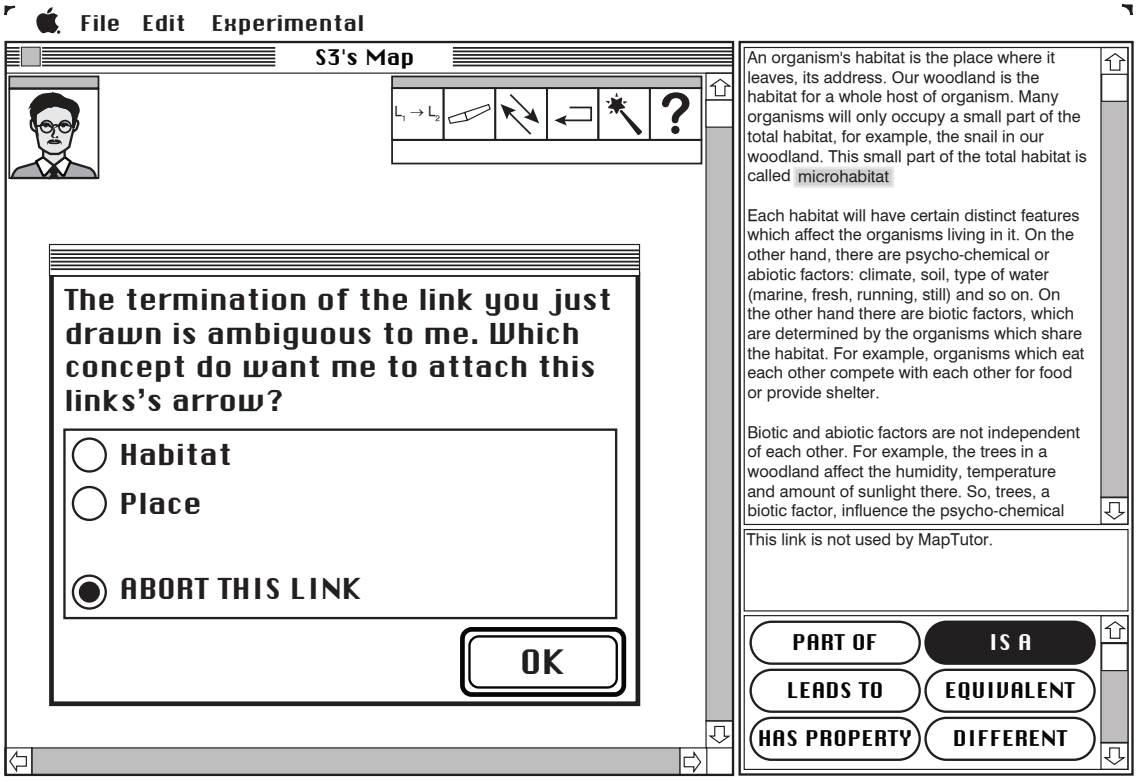

FIGURE 5–5: ASKING FOR CLARIFICATION OF AN AMBIGUOUS LINK

Coming back to **Figure 5–4**, let us suppose for the sake of argument that the program allowed that construction. Let us further imagine that the learner wanted to remove the ambiguity by moving one of the involved concepts farther apart. The program would have to decide whether the ambiguous link would go along with the to-be-dragged concept or stay with the retained one. This decision would obviously be rather arbitrary because the program has no clue whatsoever about the learner's intention. But this is not so bad: if the program had attached the ambiguous link to the undesired concept, the learner would simply undo her dragging, and then drag the other concept instead. In any case, we would end up with a non-ambiguous link which then would have to be evaluated by the diagnostic processes, feedback would be provided and so on. Now, to make things much worse, imagine that in **Figure 5–4** we had two ambiguous links (say, another link starting in a point between concepts HABITAT and PLACE) instead of only one[10]. At the end of the process just described, the program would have to

[10] This may sound a bit weird and unlikely, but remember the engineering design principle (or the equivalent motto: better safe than sorry).

provide feedback referring to two ex-ambiguous links in a row, and this could cause even more confusion for the learner[11]. To sum up, the best solution to this hypothetical problem is to prevent it happening in the first place. In the example just described, the best thing to do is to avoid the link being ambiguous by asking the learner to which concept she intends to attach her link as soon as she introduces the ambiguity, or to disallow the link in the case where she cannot decide.

Another construction considered spurious by MapTutor is that involving duplication of links. A link is duplication when there's another one that links the very same concepts as the first link, and these links are not allowed to coexist together (see **Section 3.7.2**). This kind of anomaly is not as serious as ambiguity, but even so they should also be avoided. MapTutor does not disallow duplicated links, but it does not diagnose or provide feedback on them.

5.4.1 Map Tasks Records

In order to keep track of the learner's actions, MapTutor maintains a number of data structures referred to as map tasks records. These records support a variety of external (e.g., reversible tasks) and internal (e.g., self-consistency testing) facilities. **Table 5–3** sums up the main data structures of MapTutor's map tasks records. The present section has been included with the sole purpose of give some idea of the internal workings of MapTutor, since full details would be too lengthy and beyond the scope of this book. Thus, only the most important structures are included in **Table 5–3**. Some of the structures presented in this table will be briefly described next.

Here, a task (or more precisely, a drawing task) is any of those actions which can be performed in the map pane (see **Section 5.3.4**). A drawn concept is represented by the drawn concept record presented in **Table 5–4**, whereas a drawn link is represented by the drawn link record presented in **Table 5–5**. A task, as defined here, is represented into MapTutor according to the task reference record shown in **Table 5–6**. Many of the data presented in **Table 5–3** to **Table 5–6** are in fact redundant and could be calculated by the program as needed, but this redundancy of data accelerate the algorithms which make use of them.

Data Member	Definition
Current drawn concept	the concept (if any) used in the current drawing task.

[11] Even worse from the programmer's point-of-view: suppose that afterwards, the learner decided to revert (undo) her action. Then, the program would also have to revert its whole updating process. The troubles caused by ambiguity seem to be endless.

Data Member	Definition
Saved drawn concept	the concept (if any) used in the last drawing task. (This is necessary to support reversible tasks.)
Current drawn link	the link (if any) used in the current drawing task.
Saved drawn link	the link (if any) used in the last drawing task. (This is necessary to support reversible tasks.)
List of drawn concepts	list of concepts drawn in the map pane.
List of drawn links	list of links drawn in the map pane.
List of tasks	list of tasks carried out in the map pane.

TABLE 5–3: MAPTUTOR'S RECORDS

Field	Definition
Name	the concept's name.
Concept rectangle	the position of the rectangle around the concept.
Name position	the position of the concept's name.
Linked	flag indicating whether there is any link originating or arriving at the concept.

TABLE 5–4: DRAWN CONCEPT RECORDS

5.4.2 Pre-Evaluating the Learner's Move

The first stage of evaluation carried out by MAPTUTOR is to preliminarily interpret the learner's last move. Moves include not only the draw tasks described above, but also actions performed by the learner over other interface elements.

When the last move was a click in the text pane, MAPTUTOR proceeds as follows:

- if a concept has been hit for its first time, it draws the concept in the map pane.
- if a concept has been hit but it has already been drawn in the map pane, it just makes it the currently selected concept in the map pane.
- if there's no concept of interest where the learner clicked on, it tells her there was no important concept where she clicked.

Field	Definition
Name	the link's abbreviate name.
Initial point	point where the line originate at.
End point	line's termination point.
Name rectangle	the rectangle enclosing the link's name.
Arrow rectangle	the rectangle enclosing the link's end arrow.
Links from	index into list of concept of the concept at which the link originates.
Links to	index into list of concept of the concept at which the link terminates.
Deleted	flag indicating whether this link has been deleted.
Duplication	index of a duplication, if this link is duplicated.
Assessment	the link's assessment value (*ass* — see **Section 3.5**) determined by the diagnostic procedures.

TABLE 5–5: DRAWN LINK RECORDS

Field	Definition
Task ID	task reference number.
Object index	the index of the main object (concept/link) used in the task.
Link Duplication removed	flag indicating whether the learner has removed an earlier duplication of links.
Link Duplication introduced	flag indicating whether the learner has drawn a duplicated link.
Evaluation	identification number of the task evaluation carried out by the diagnosing methods. A task is evaluated only when it results in a new connection between two concepts (see **Section 4.3**).
Links to	index into list of concept of the concept at which the link terminates.

TABLE 5–6: TASK REFERENCE RECORDS

If the last move was a click in the links pane, MAPTUTOR informs the map pane that all subsequent links drawn in that pane will have this new chosen name. If the last move was a drawing task in the map pane, MAPTUTOR interprets the task and fills in some fields of those records[12] described in the last section. Then, it

[12] Some records are only fully filled after diagnosis.

calls a procedure responsible for evaluating the mapping task. Evaluation of a mapping task is the last stage before the task is actually diagnosed and feedback is provided as described in **Chapter 4**. This stage of evaluation is depicted next.

5.4.3 Evaluating a Mapping Task

The evaluation of mapping task stage is responsible for deciding whether or not the learner's last task should be diagnosed and the tutoring parameters described in **Chapter 4** updated. A task is diagnosed and the appropriate feedback is delivered only when it results in a new legal link[13], according to the criteria established in the previous section. **Table 5–7** summarises the actions taken by MapTutor at this stage, according to the last mapping task.

If the last task was...	The action taken is...
Concept dragging	nothing at all needs to be done, because moving a concept does not result in new links. Objects which changed their position as a result of the dragging have already been updated in the preceding stage.
Link drawing	if the link is a legal one, it will be diagnosed and feedback will be provided.
Link clearing	no diagnostic or feedback is carried out on the link itself, but a if a duplication has been removed as a result and the remaining ex-duplicated link is a legal, not previously evaluated one, the latter will be diagnosed as if it has just been drawn.
Link renaming/inversion	if the renamed/inverted link is a legal one, it will be diagnosed and feedback will be provided, as if it had been deleted and then redrawn, with a new name/direction.

Table 5–7: Evaluation a Mapping Task

No diagnostic or feedback procedure is executed when the learner has changed her mind and has reversed (i.e., undone/redone) her last task. However, if the tutoring parameters, discussed in **Chapter 4**, were updated when the original task was carried out (i.e., done), MapTutor must also revert this updating.

[13] As a matter of fact, the only anomalous link which could arrive at this stage is link duplication. A would-be dangling link will have already been disallowed, and a would-be ambiguous link will have been either clarified by the learner or disallowed too.

5.5 The Help and Tutorial Programs

MapTutor has two separate but non-independent auxiliary programs which contribute to the main program's overall environment: (1) the tutorial program, and (2) the help program. These programs have been developed by using a commercial authoring tool Guide Maker, by Apple Computer, Inc. Both the tutorial and the help programs associated with MapTutor serve basically the same purposes: (1) to teach the learner how to use the program; and (2) to provide her with some more useful related information. The tutorial program assumes that the learner is a newcomer to the main program and does not possess the skills necessary to use it properly. Thus, the tutorial presumes that the learner who uses it needs assistance as well as a guided path (i.e., a well-defined instructional sequence) throughout the tutorial itself. Moreover, although the tutorial does not impose any instructional sequence, it does suggest that the learner should follow the one prescribed to her. **Figure 5–6** shows the panel of the tutorial program which contains the main sequence of study suggested to the learner. MapTutor launches its tutorial program whenever a new session[14] starts (see **Figure 5–1**), but if the learner does not wish to use this program, she can dismiss it by simply closing its window. To sum up, the tutorial program associated with MapTutor implements both the Pre-Teaching stage of the training approach described in **Chapter 3**, and teaches a novice learner how to use the main program.

On the other hand, the help program is much more flexible and has a broader content than the tutorial. The help program assumes that the learner already knows how the program works and it is called for only when she faces some difficulty or cannot remember something she had already learnt during the tutorial. Furthermore, in contrast to the tutorial program, the help program allows for free browsing, and various styles of searching are supported as well. For example, the learner may look for a topic (e.g., how to rename a link) by browsing the index or the list of topics, or by asking the program to do the search on her behalf (e.g., by providing *rename* as a search key).

[14] More precisely, the tutorial is launched when the learner starts a new map in a new session, because if the learner invokes a new session to continue the construction of an old map, MapTutor will assume that she already has the necessary basic skills in working with the program, and thus it will not launch the tutorial program. She may, of course, always launch this program by herself by pulling down the help menu and choosing the tutorial option.

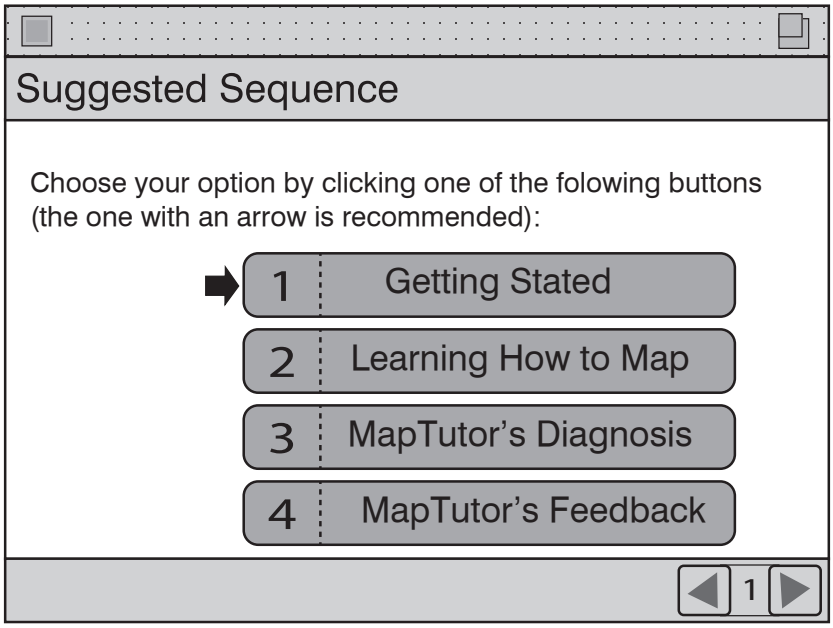

FIGURE 5–6: TUTORIAL PROGRAM'S SUGGESTED SEQUENCE

Figure 5–7 shows the help program's main browsing window. As can be seen in that figure, the instructional material is chunked into topic areas (see left column in **Figure 5–7**); the topics themselves (right column in **Figure 5–7**) are divided into headings which answer procedural (e.g., *How do I* and *Why can't I* headings) or descriptive (e.g., *Definitions* heading) queries.

As mentioned above, the help and tutorial programs are not independent from the main program, and that dependency means that (1) MAPTUTOR will behave in a slightly different way when one of these programs is being used than otherwise, and (2) it will provide context-dependent information to these auxiliary programs to act in a more intelligent manner (see **Implementation Note 2** in **Section 5.10**). For example, if the learner asks the help program to teach her how to rename a link, its behaviour will depend upon whether she already has a selected link in her map or not, she has already selected a new name to replace the old one or not, and so on. **Figure 5–8** illustrates this example. In that situation, the learner had already a selected link in her map when she asked the help program to teach her how to rename the link, but she has not selected a new name yet. The help program is informed by MAPTUTOR about the current situation and starts its instructional sequence by telling the learner that she should select a new link name in the first place.

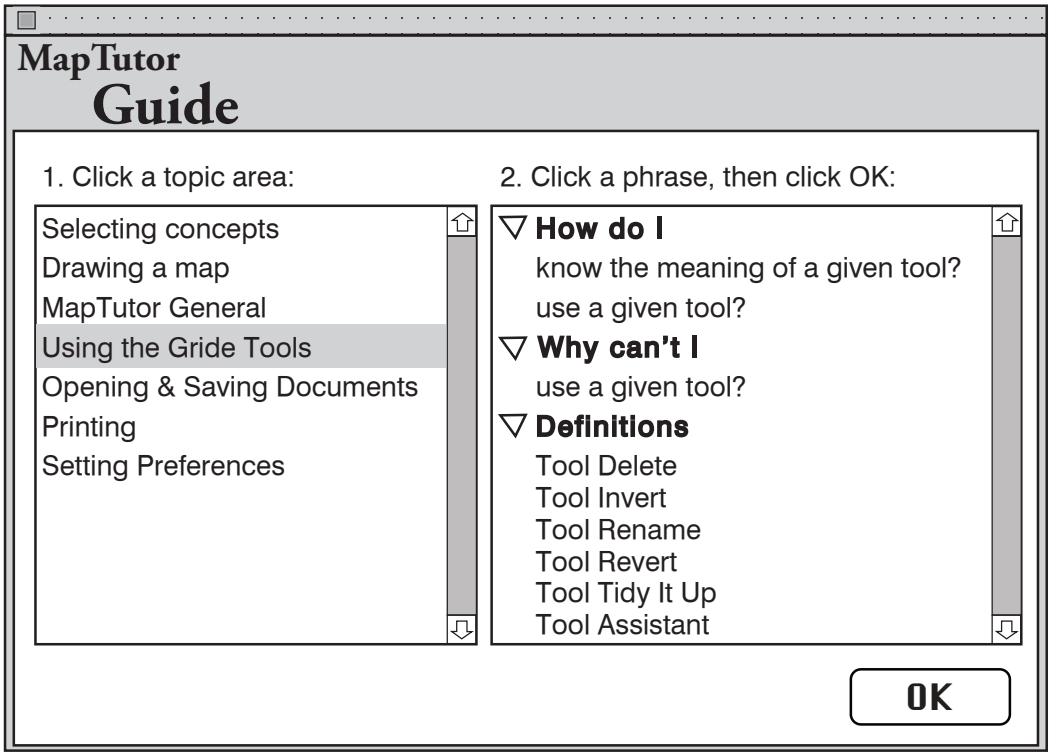

FIGURE 5–7: HELP PROGRAM'S MAIN BROWSING WINDOW

The help program will also know whether the learner has carried out correctly its instructions (e.g., the program instructed her to click on the rename tool, but she clicked on the invert tool instead) and adapt its feedback sequence accordingly. Continuing with the last example above, in **Figure 5–9**, the learner has not followed the instruction (**Figure 5–8**) given by the help program. It has then stopped the normal sequence to warn her about this. The help and tutorial programs also provide the learner with visual information by means of **coach marks**[15], which are intended to complement the instructions or definitions provided, and facilitate learning (at the very least, coach marks embellish the feedback and may even motivate the learner to proceed just to see where the next coach mark is going to be drawn). The coach mark positions are also provided by the main program in the cases where they refer to mobile (e.g., windows) or changing objects (e.g., concepts and links). **Figure 5–10** shows the help program presenting a coach mark around a link the learner has just renamed. The scenario depicted in **Figure 5–10** would be the last in the rename example developed above, and the final panel along with the coach mark are intended to call her attention to the effect of her last operation.

[15] A coach mark is an on-screen graphic that circles or points to an item on the screen' (Apple Computer, 1995, p. 2–79) so as to call the user's attention to the item.

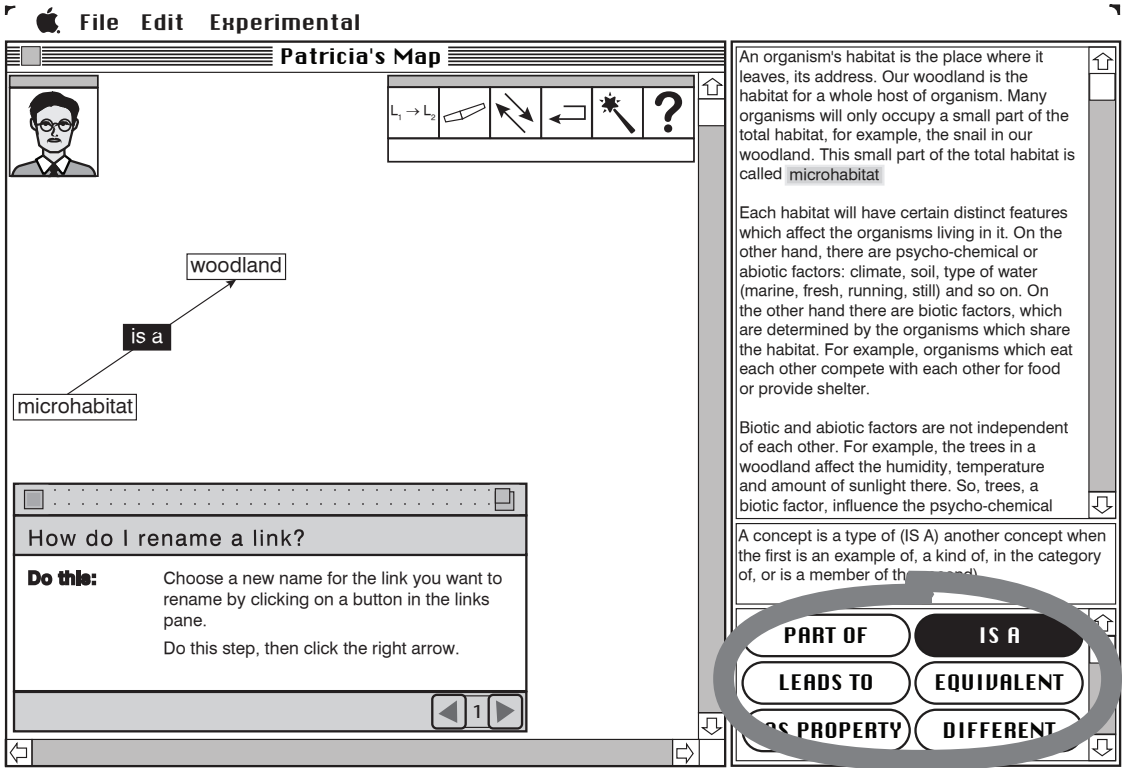

FIGURE 5–8: HELPING THE LEARNER TO RENAME A LINK I : THE LEARNER HAS JUST ASKED THE HELP PROGRAM TO TEACH HER HOW TO RENAME A LINK.

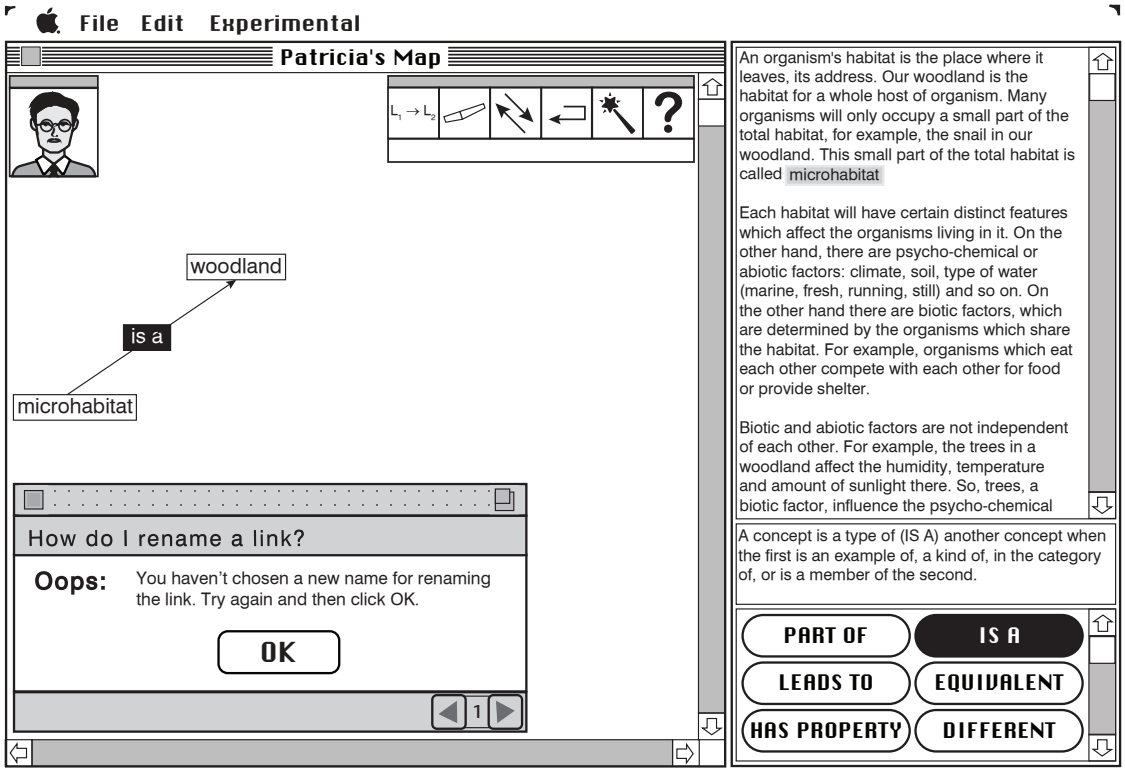

FIGURE 5–9: HELPING THE LEARNER TO RENAME A LINK II: THE LEARNER HAS NOT DONE WHAT THE HELP PROGRAM HAS TOLD HER TO DO

Another form of help supported by MapTutor is by means of Apple's **balloon help**. To use this feature, the learner should choose *Show Balloons* in the help menu. When the balloons are on, the learner can see the meaning of each interface element of MapTutor by simply moving the cursor around these elements.

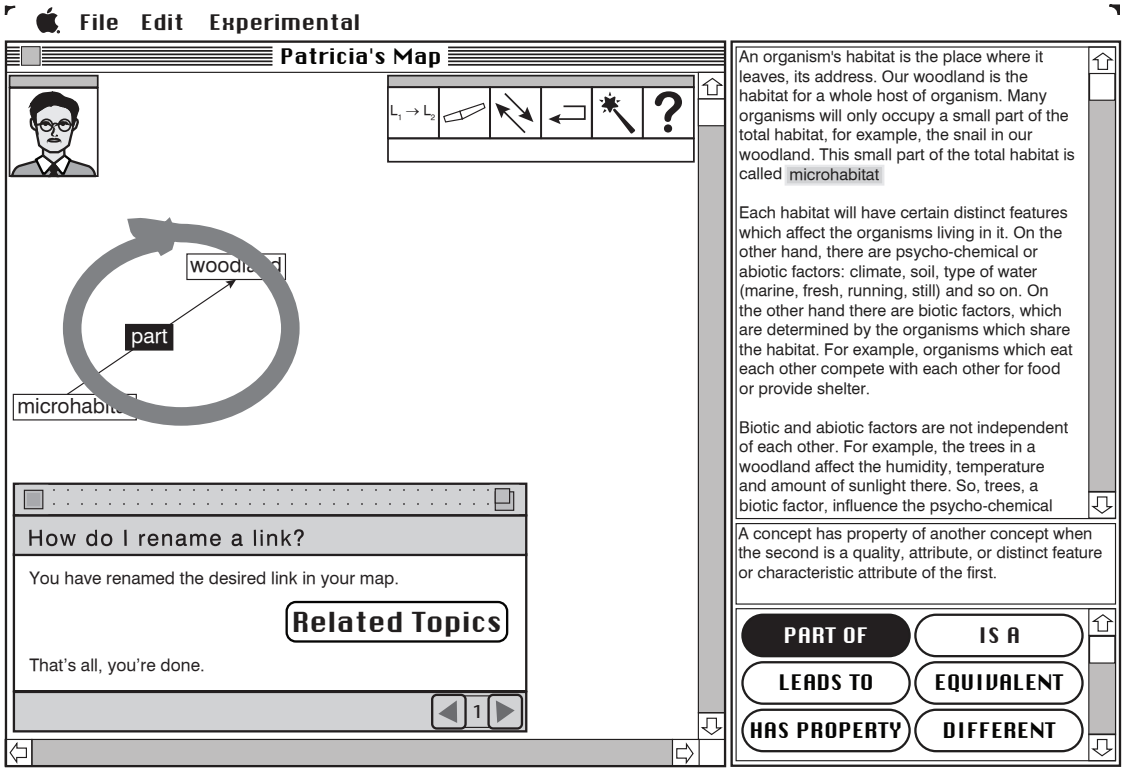

Figure 5–10: Helping the Learner to Rename a Link III: The learner has just completed the renaming operation

5.6 Using MapTutor

The user of MapTutor is instructed by the tutorial program described above to follow the sequence below when using the program:

1. Read all the text thoroughly.
2. Select concepts which are relevant to the learning objectives[16].
3. Connect the concepts selected in **Step 2** in the map pane until you prove to know all the major concepts (according to the criterion defined in **Section 3.5**).

[16] Somehow, the learner must know what the learning objectives are. One can read a text with any objective in mind. Therefore, obviously, the learner's objective must not conflict with the program's.

There might be many variations of the sequence above, and MapTutor is not rigorous about this. For example, **Step 2** can be carried out in tandem with the first-time reading, or alternatively, can be executed in a second pass throughout the text. The latter probably has some additional learning gain because the learner reads the text twice (see, e.g., Just & Carpenter, 1987).

After this brief, general introduction, the tutorial program teaches specific details about how to use MapTutor's facilities in order to construct a graphical map. These topics can be summed up as follows[17]:

- **To draw a concept**: choose a concept in the text pane by clicking on its name until it is highlighted, and let MapTutor do the rest on your behalf. Whenever a new concept is drawn, it appears with an animated rectangle (called marquee) around it indicating that it is selected. A selected concept (i.e., a concept with a marquee) can be moved around the map pane.
- **To draw a link**: choose a link name, represented by a button with the link's name inside, in the links pane by clicking on its button until it becomes highlighted. Then click on close to a concept in the map pane and drag a line to a point close to another concept. You will know whether the pointer is sufficiently close to a concept when it takes the shape of a pencil.
- **To drag a concept** (i.e., to move it to another position in the map pane): select the concept by clicking on the respective concept in the text until it gets selected (with a marquee). Then click the mouse again and keep it pressed while you move the concept around. When the concept is at the desired position, release the mouse.
- **To delete (clear) a link**: select the link by clicking on or around it in your map until you see a marquee around the link's name. Then press the DELETE/BACKSPACE key. Alternatively, you can also click on the *Delete Tool* in the tools pane. The Delete Tool takes the shape of an eraser.
- **To rename a link**: select the link by clicking on or around it in your map until you see a marquee around the link's name. Then choose a new name in the links pane by clicking on its button name until it becomes highlighted. Then click on the *Rename Tool* in the tools pane. The Rename Tool takes the shape of two L's.
- **To invert a link**: select the link by clicking on or around it in your map until you see a marquee around the link's name. Then click on the *Invert*

[17] These topics are distributed over a number of tutorial panels.

Tool in the tools pane. The Invert Tool takes the shape of two opposite arrows.

- **To revert (undo/redo) a drawing action**: click on the *Revert Tool* in the tools pane. The Revert Tool takes the shape of a U-turn.
- **To clean up your map**: click on the *Tidy It Up Tool* in the tools pane. The Tidy It Up Tool takes the shape of a magic wand. However, tidying your map up is not always possible, so try to keep it neat by yourself.

The mapping procedures depicted above constitute part of the **Learning How to Map** sequence which is initiated by clicking the button which has this name inside (see **Figure 5–6**).

5.7 Tutorial Interventions

The tutor's interventions are mostly in form of balloons (see **Implementation Note 3** in **Section 5.10**) centred around the tutor pane. After reading a comment from the tutor, the learner can dismiss the balloon by simply clicking the mouse anywhere.

MapTutor supports two types of feedback messages, namely automatic feedback messages — i.e., those which the program believes to be essential and thus must be delivered automatically, and user-demanded feedback messages — i.e., those which the program simply signals (see **Section 5.3.7**) the learner they are available, but are only delivered upon the learner's own request. The latter type of message avoids the program being to obtrusive when dealing with minor problems. For example, the learner is warned before starting a new session by the tutorial program that concepts will only be drawn in the map pane if she hits an important concept in the text. Thus, MapTutor assumes that the learner will remember this whenever no concept is drawn as a result of her click. Then, it will highlight the tutor pane to signal the learner that something went wrong with her last action; if she does not know what the problem was, she will ask the tutor by clicking on the tutor pane; otherwise, she will simply keep going.

Note that automatic feedback messages, when the program has determined their necessity, are only presented to the learner at idle time[18]. This period of time usually does not represent much time in computational terms, but it prevents

[18] In fact, idle time refers to period of time the program is not doing any serious business (e.g., it may be animating a marquee or blinking a caret simply because it has nothing else to do), but in practical terms, as the program's responsiveness is very rapid, this also means the period of time the user is inactive.

the learner being suddenly offered a feedback message while she is doing something else.

Another form of tutorial intervention occurs when the program asks the learner some thing as part of its diagnostic process (see **Section 4.7**). In these situations, MapTutor presents the learner with a dialogue window containing options for her to choose from. **Figure 5–11** shows an example of such a dialogue window presented by MapTutor. In this figure, the program is asking whether the learner is sure to have understood the relationship between concepts organisms and microhabitat. Note also that the piece of text the question refers to is highlighted in the text pane.

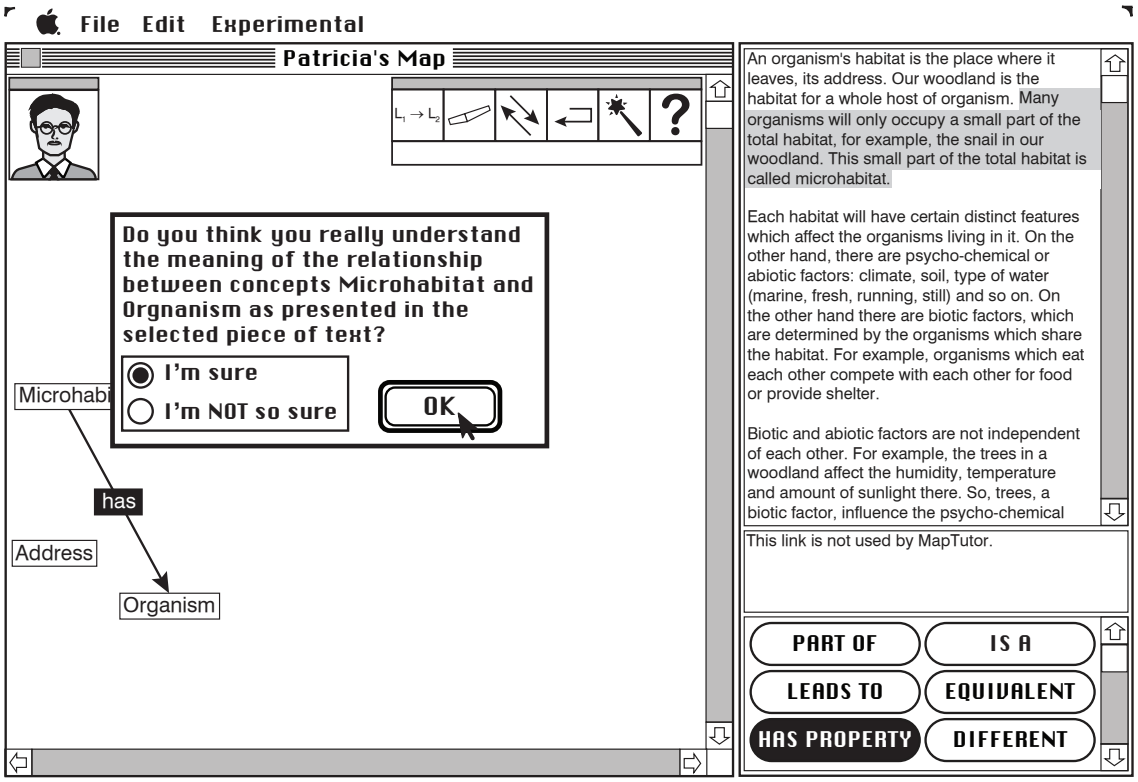

Figure 5–11: Asking for Confirmation

5.8 End-of-Session Report

At the end of a mapping session, MapTutor always creates a final report consisting of the session's log. This report is then saved as a text file separate from the file containing the final map itself, which is saved in MapTutor's own file format. **Table 5–8** shows an example of such report created by the program. In this table, added text is enclosed by square brackets. Some text has been intentionally deleted at the end to keep it short.

```
MapTutor's Final Report

[Preamble]
Student's name: Patricia
This session started on 27/8/95 at 5:41:28 am.
This session ended on 27/8/95 at 5:46:12 am.
Duration of session: 4min44secs.
Overall evaluation: 0.00.
This session was aborted;

[Beliefs Section]
The student has proven to know the following concepts:
. Organisms competing for food
. Organisms eating each other
. Organisms providing shelter
. Running
. Fresh
. Climate
. Marine
. Place
. Address
The program believes the student somewhat knows how to
map the following concepts:
. Organisms competing for food (1.000)
. Biotic Factor (0.400)
. Organisms eating each other (1.000)
. Organisms providing shelter (1.000)
. Microhabitat (0.455)
. Habitat (0.273)
. Running (1.000)
. Type of water (0.375)
. Fresh (1.000)
. Climate (1.000)
. Physico Chemical Factor (0.200)
. Abiotic Factor (0.144)
. Organism (0.182)
```

TABLE 5–8: END-OF-SESSION REPORT CREATED BY MAPTUTOR (CONTINUES...)

```
. Marine (1.000)
. Place (1.000)
. Address (1.000)

[Mapping Task Section]
There follows the list of actions the student
performed in the map pane:
[1] Draw a link named ISA from Organisms competing
for food to Biotic Factor. MapTutor found that this
link was wrong and the cause was misunderstanding
of meaning of canonical links.
[2] Renamed a link named LEADS from Organisms
competing for food to Biotic Factor. MapTutor found
that this link was correct.

[deleted text]
```

TABLE 5–8 (CONT): END-OF-SESSION REPORT CREATED BY MAPTUTOR

As can be apprehended from **Table 5–8**, the end-of-session report consists of information gathered during the mapping session. This report can be divided into three main sections:

- **Preamble** — containing general information about the session, such as the student's name, date, time, etc. This section also contains an item — called **Overall Evaluation** — which indicates the proportion of major concepts (learning objectives) the program believes the learner knows (according to the criterion established in **Chapter 3**) in relation to the whole number of major concepts.
- **Beliefs Section** — this section contains precisely the program's beliefs about the learner's performance (see **Chapter 3**). It can be subdivided into two sub-sections:
  - ◊ **Known Concepts** — consisting of those concepts MAPTUTOR believes the learner has mastered[19] (see **Section 3.5**);
  - ◊ **Linked Concepts** — this sub-section contains the list of concepts the learner both selected in the text and linked in the map pane. The belief-degree (BD — see **Section 3.5**) associated with each linked concept is the number following the respective concept and enclosed by parentheses.

[19] The Knowing-Threshold (KT) was set at 0.6 at the time this report was created.

- **Mapping Tasks Section** — this section contains all mapping tasks the learner performed in the map pane.

The end-of-session reports created by MapTutor serve two main purposes according to whom makes use of them:

1. **For the Researcher**. From the researcher's standpoint, these final reports are useful not only to see whether the program is working properly as specified (e.g., by comparing hand-calculated BD's with the program's), but also and most importantly, to verify whether the theory is appropriate (e.g., by analysing the various reports and trying to uncover error patterns — see **Chapter 6** and **Appendix D**). Thus, these reports can also be used as a research tool.

2. **For the Learner**. The student may also learn more from her own mistakes by studying these reports at the end of each session.

5.9 Entering the Knowledge

The knowledge base of MapTutor has been currently input using a programmer's interface, instead of the authoring interface proposed in **Figure 3–1**. The reason for this is that the latter would require a hefty programming effort, which is unjustifiable for the primary goals of this research. The current interface does not present any trouble for a programmer familiar with Macintosh development, but it is unsuitable to a user who is not familiar with such an environment. This interface contains a series of templates associated with each of those object prototypes presented in **Section 3.7**. These templates were built using a resources editor (see **Implementation Note 4** in **Section 5.10**). Despite requiring some specific programming knowledge in order to be directly entered into the program, the knowledge base can easily and quickly be specified (and modified) without requiring such specific knowledge. It is estimated that the knowledge associated to a text and a set of canonical links like the ones describe in this book would take about eight hours to be fully specified. Then, a programmer would take no more than two more hours to enter it. A friendly authoring interface would save half of this time. The design of such an authoring interface is suggested in **Chapter 7**.

5.10 Implementation of MapTutor

The object-oriented paradigm is a very natural choice for implementing a program described with an architectural design like the one in **Figure 3–1**. Thus, C++, the language chosen for implementing MapTutor, has proven to be a suitable

option. Moreover, the object-oriented approach is very suitable for implementing graphical interfaces and has helped to reduce the programming time and effort.

MapTutor has been implemented and runs on the Macintosh[20] using MacOS[21] 7.5. The implementation language has been Symantec C++ 7.0 using Think Class Library 2.0[22]. Currently, MapTutor's source code alone has about 20,000 lines distributed over about 100 files[23]. Put together with the library utilised, the source code goes beyond 200,000 lines but not all of this figure is actually part of the final object code because the linker strips off unused procedures. Thus, it is difficult to estimate how many lines of code MapTutor actually has. The tutorial and help programs were built using Guide Maker[24]. These programs work only with MacOS 7.5 or later.

Implementation Notes:

1. MapTutor's implementation of graphical objects is a mix of vector-based graphics and bitmapped graphics. In other words, it does have mathematical (i.e., vector based) definitions of the objects, which it uses for bookkeeping and file-related operations, but it still draws the objects using bitmapped images. The latter causes the *bites* when intersecting links are deleted. You might be wondering, Why is it then those concepts are never bitten by either intersecting links or other concepts? The trick here is simple: MapTutor drawings are made of a superposition of two bitmaps — one for drawing concepts and the other for drawing links.

2. The dependence between MapTutor and its auxiliary programs is established by means of an interapplication communication protocol. For more information, see Inside Macintosh: Interapplication Communication, Apple Computer (1994).

3. MapTutor uses *Balloon Help* by Apple Computer, Inc., and this technology has brought some problems since its intended use is not the one the program puts it to. The major problem is that if the user moves the mouse suddenly just before a balloon is presented, the operating system will abort the balloon. This problem has been overcome by using two feedback queues which swap feedback messages with each other when a given

[20] Trademark of Apple Computer, Inc.

[21] Trademark of Apple Computer, Inc.

[22] Thus, part of the source code of MapTutor is copyrighted by Symantec Corporation.

[23] These figures include the knowledge base, and the help and tutorial program's source code.

[24] ©Apple Computer, Inc.

message cannot be delivered. Also, it is partially due to this problem that feedback messages are only presented in idle time. Another solution attempted to deliver feedback was the use of modal dialogue windows, but although this is the recommended way of presenting messages to users of other types of application programs, this is not a good solution for a highly interactive program like MapTutor, which needs to deliver messages quite frequently. Another solution could be to create another pane large enough to contain the feedback messages, but this was also discarded because the screen was already critically occupied by other interface elements.

4. The resources editor utilised was *ResEdit*, ©Apple Computer, Inc. Sadly, the version of ResEdit used most of the time during the development of MapTutor was buggy and several hours of work were lost. The new version is still buggy, but at least they moved the bug elsewhere!

5.11 Conclusion

This chapter presented an overview of the practical aspects of implementation of MapTutor. As Flagg (1990) points out, a good, friendly interface does not guarantee that learning will take place, but a poor interface can interfere negatively with the learning process. From a research view point, an unfriendly interface could also hamper the re searcher's goals. For example, as seen in **Chapter 2** by using an interface which did not tie symbols representing concept to words-in-the text the symbols are supposed to represent, Feifer (1989) might have introduced a problem unique to his research, namely the icon interpretation problem. He also spent most of his thesis attempting to resolve this problem, which could have been minimised simply by presenting the text on-line and tying together the symbols representing the concepts with the context in which those concepts appear in the text. Therefore, the interface of a program like MapTutor should at least permit that the research hypothesis be evaluated without the bias introduced by a poor interface.

The implementation of MapTutor's interface represented an enormous programming effort. This interface was the most difficult part of implementation, the most time consuming, and the largest part of the whole system. Including interpretation and book keeping of objects, the interface of the program represented about 70% of all programming effort. On the other hand, MapTutor's interface has taken seriously the issue of how a real-world interaction would look like, which many programs found in ITS research overlook.

This chapter also illustrates the use of a commercial authoring tool — *Guide Maker* — with a tutoring program. At first sight, it may appear that the tutorial and help program are mere frame-based program (in computer-assisted instruction terms). This does not seem to be the case, however. When well integrated with a main program which behaves intelligently, help and tutorial programs constructed with this tool can be made cleverer too. The former would be responsible for providing the necessary intelligence to the latter.