ASTRAL Software Development Environment User's Manual

Paul Z. Kolano
Reliable Software Group
Computer Science Department
University of California, Santa Barbara

revised July 1996

1. Introduction

The success of any language, be it for implementations or specifications, is very often directly related to the availability and quality of tools that support it. Implementers have been the beneficiaries of tools such as integrated development environments for quite some time. Specification writers, however, have not been as lucky. It is the rare specification language that is supported by a full development environment. This is quite unfortunate since such tools are usually accompanied by increases in both efficiency and correctness. They do so by not only offering all the features of traditional standalone components (i.e. editors and spec processors), but by additionally offering features which only become possible when the components become integrated. Some examples of these are:

- hierarchical navigation of specifications
- syntax directed editing
- on-line syntax documentation
- automatic logic expression formatting
- ability to jump directly to error locations

ASTRAL [CKM 94] is a formal specification language for real-time systems. It is intended to support formal software development and, therefore, has been formally defined. The structuring mechanisms in ASTRAL allow one to build modularized specifications of complex systems with layering. A real-time system is modeled by a collection of state machine specifications and a single global specification.

The ASTRAL Software Development Environment (SDE) is an integrated set of design and analysis tools based on the ASTRAL formal framework that incorporates all of the above items and more into a single package. The tools that make up the support environment are a syntax-directed editor, a specification processor, a verification condition generator, and a browser kit. This document describes the features and use of the ASTRAL SDE.

2. Navigation

The SDE operates in one of two modes: specification mode for working on single specifications and composition mode for working with multiple specifications and composition specific clauses. The functionality of the two modes is identical except for the items discussed in section 6. The initial and customary mode of operation is specification mode. Figure 1 shows the SDE on startup. The most important feature of the figure is the navigation window. The navigation window allows the user to hierarchically traverse the current specification or composition and perform various operations on the location displayed. To simplify the discussion throughout the remainder of the manual, the term "navigation object" will be used to refer to the current object being hierarchically displayed in the navigation window (i.e. the current specification if the SDE is in specification mode or the current composition in composition mode). At startup, the window shows the highest level of abstraction in specification mode known as the specification level. Note that level in this instance refers to the level of the specification and not the level of refinement in an individual process. The specification level shows the name of the global section and the names of all processes defined. The navigation window is arranged hierarchically meaning that most of the items listed can be selected and the display moved "down" to a finer level of detail or "up" for a coarser view.

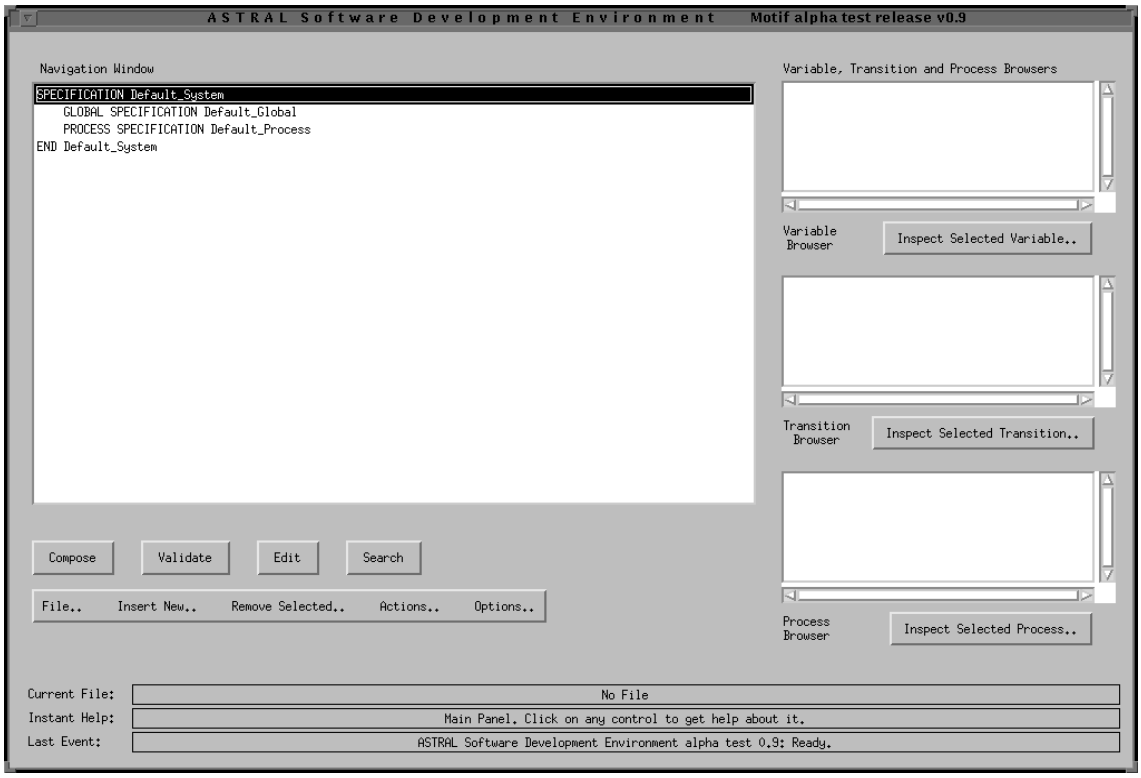

Figure 1: the ASTRAL Software Development Environment

Figure 2 shows the specification mode hierarchy of the navigation window which is not coincidentally identical to the hierarchy of ASTRAL specifications. Wherever a parent-child relationship exists in the tree, there exists some item in the navigation window when in the parent's level that can be double clicked on to move to the child's level. For example, to see a particular process in finer detail, double click on the name of that process in the specification level. The navigation window then lists the names of all refinement levels in that process. Similarly, to move up the tree, the topmost item in the navigation window (which will be the name of the current navigation window level) can be double clicked on and the parent level will be displayed. Double clicking is not the only way to change levels. Directly beneath the navigation window are an up and a down arrow . These arrows will only appear when these operations are available for the currently selected item. For example, when a process is selected at the specification level, the down arrow will appear because a process can be seen in finer detail but the up arrow will not because the specification level is the highest level possible in the specification mode hierarchy. To move down from a particular item, select it and click on the down arrow. The up arrow has no relation to the current selection of the navigation window and will always move up to the parent's level. There is no situation when the arrows can be used while the double clicking method cannot although in certain instances, such as in lengthy transitions, it may be more convenient to just click the up arrow than to scroll up and double click the transition name.

From a navigation perspective, a "down" at the leaves is not a valid operation. Since all leaves are text, however, this has been made a convenient location to invoke the editor. Similarly, an "up" at the root invokes the editor on the root node's name. Unlike the up and down arrows, however, there are places (i.e. interior nodes with user defined names) where double clicking cannot be used to invoke the editor since it already functions to move down a level. In these cases, the item must be selected with a single click and the '**Edit**' button

at the bottom of the SDE pressed. Note that this technique also works for the leaf nodes but is usually not as convenient. The '**Edit**' button will be "grayed" (i.e. unavailable) if the selected item cannot be edited. Note that in multi-line text sections, the editor cursor will be automatically moved to the line which was double clicked on to invoke the editor.

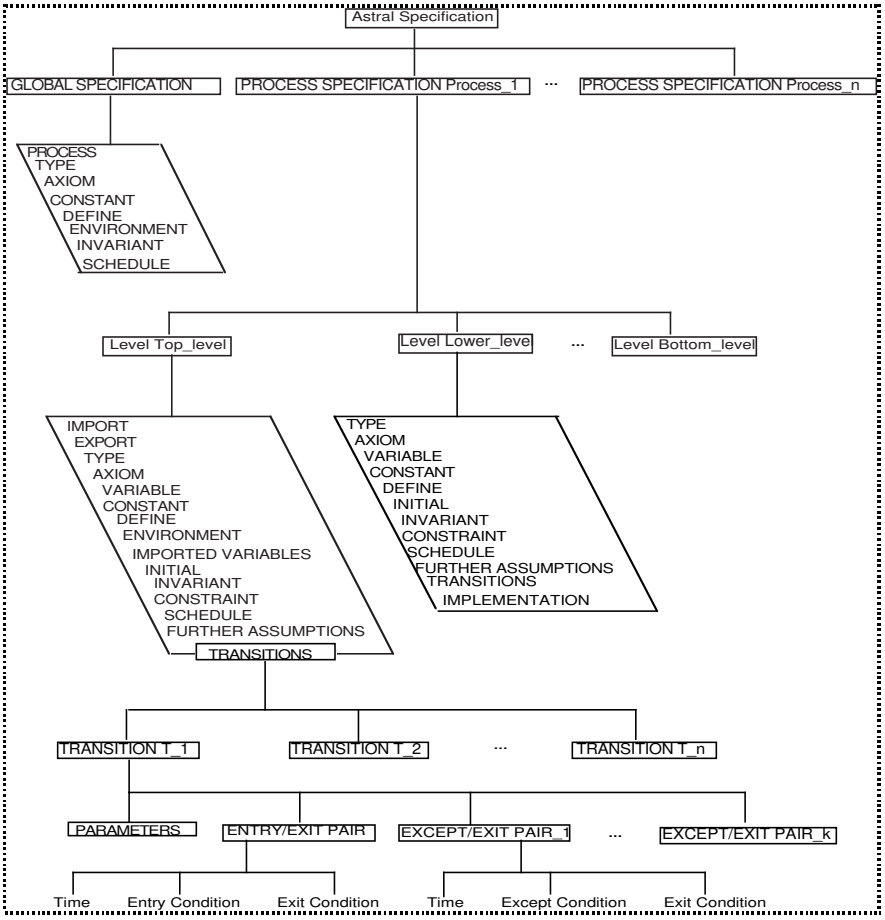

Figure 2: the navigation hierarchy

3. Editor

Although the editor (figure 3-1) provides only the most basic functionality of common general purpose editors such as vi or emacs, it is rarely the case that an ASTRAL section is so large as to require the same level of functionality that such general purpose editors provide. More importantly, the syntax checking, automatic formatting, and on-line syntax documentation of the SDE editor more than compensate for this shortfall.

3.1. Editing

The editor is simple to use. The cursor is moved with the arrow keys or a single mouse click and then text is entered or deleted. Cutting and pasting is accomplished using standard X-Window techniques. First, a portion of text is selected by either pressing the left mouse button at the beginning of the block and releasing it at the end of the block or by using selection shortcuts. These shortcuts are a double click to select a word, a triple click to select a line and a quadruple click to select all text. After

a selection is made, the selected block is deleted by pressing either the delete or backspace key or can be replaced by entering the replacement text directly or pasting it in from another location. To paste text from another window or another part of the editor, select it, move the mouse pointer to the appropriate insertion location within the editor and click the middle mouse button. The editor also supports drag and drop, so text may be moved by selecting it, "dragging" it with the middle mouse button, and releasing the button when the pointer is in the desired location.

3.2. True selection

All of the well formed formula sections (i.e. those that are logic expressions and not declarations) are created with default text of "TRUE". Unless the user explicitly changes them, they will remain "TRUE" throughout their lifetimes. When the editor is invoked on a section that has "TRUE" as its text, the "TRUE" is automatically selected so that any text entered will replace it. Since this will be the desired action in almost all cases, true selection relieves the user of having to do this manually every time.

3.3. Syntax-directed editing

All editable items in the SDE are associated with a specific grammar, whether the simple alphanumeric constraint on names or the complex grammars of well formed formulas. Through the use of these grammars, the editor is able to parse its current text and indicate the presence or absence of syntactic errors. If the user is unaware of the exact syntax of a section, the '**Help**' button displays not only the corresponding grammar but also other pertinent information about the item being edited. The '**Cancel**' button will exit the editor and leave the text unchanged. When the text is correct with respect to its grammar, the '**OK**' button is available at the bottom of the editor. When a syntax error is found, however, the '**Parse Error**' button is displayed instead. In addition, the line that is believed to contain the error is underlined. Note that this line is only an approximation to where the error is located. If, for example, the parenthesis are unbalanced, the parser has no way of knowing this until the last line has been processed so it reports the error at the last line (and hence that is the line underlined by the editor). In most other cases, however, the line underlined will be either the line with the actual error or the line immediately following the error. Figure 3-1 shows the editor window.

3.4. Formatting

When the '**OK**' button is pressed, the text in the editor replaces the text the editor was originally invoked on. Before the text is replaced, however, the new text is automatically reformatted into a fixed

format. Such formatting is a very useful feature for specification writers. When first writing specifications, it is much more important to be concerned with the content rather than the readability of the specification. This doesn't mean, however, that readability is unimportant. In fact, an unreadable specification is likely to contribute unnecessary confusion, errors, and additions to development time. Manual formatting, however, is also likely to impact development time, especially during the initial design stages when changes are likely to occur frequently. In the same way as word wrap (which is turned on in the editor for this very reason as well) in word processors allow writers to concentrate on their words instead of where to place carriage returns, so automatic formatting in the SDE allows the specifier to focus not on how the specification is entered but on what it says. Figures 3-1 and 3-2 show before and after formatting snapshots of a random text segment. The main disadvantage of fixed formatting is that the format cannot be changed. The programmed format is readable enough and the other benefits so great, however, that this lack of flexibility seems insignificant.

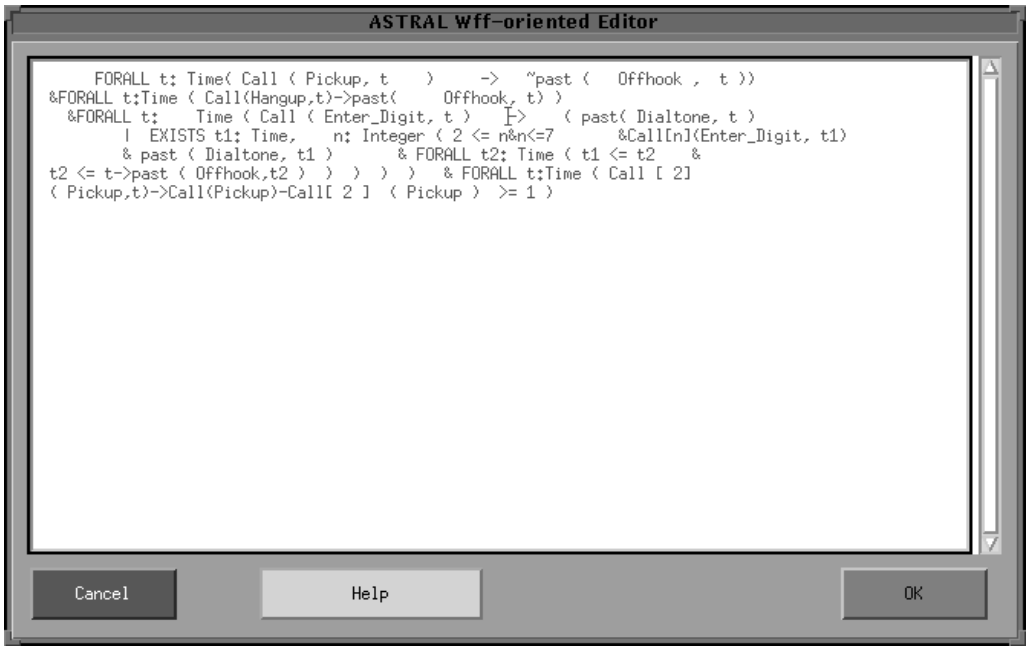

Figure 3-1: before formatting

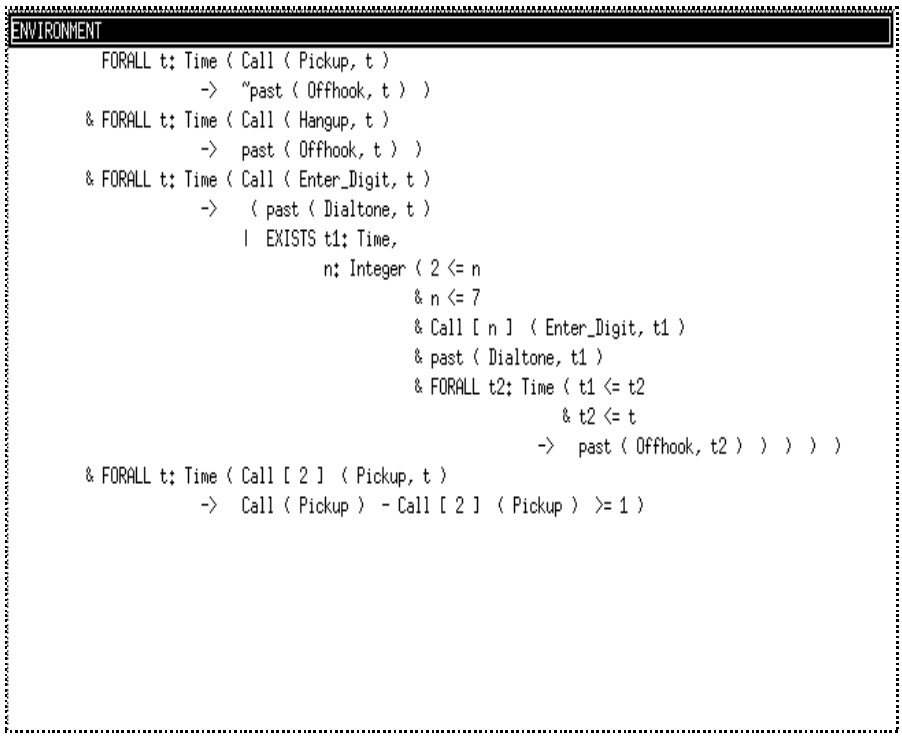

Figure 3-2: after formatting

One advantage of fixed automatic formatting which is not immediately obvious is that it allows the user to catch different types of semantic errors that might otherwise go undetected in specification analyzers. Figures 3-3 and 3-4 show a portion of a specification with two possible trouble spots. Figure 3-3 shows what was entered while figure 3-4 shows what was most likely desired. The first error (lines 3-10) occurred because precedence rules were neglected. After formatting, the user can instantly see the logical precedence of the expression and verify its validity. The second error (lines 22-23) is easier to spot. After a chain of three "FORALL t", the last FORALL is not within the same scope which points to a misplaced parenthesis. These errors would most likely go undetected with manual formatting because the user would format them as it was assumed they were written, which was in this case incorrect, even though the text was in fact, both type correct and syntactically correct.

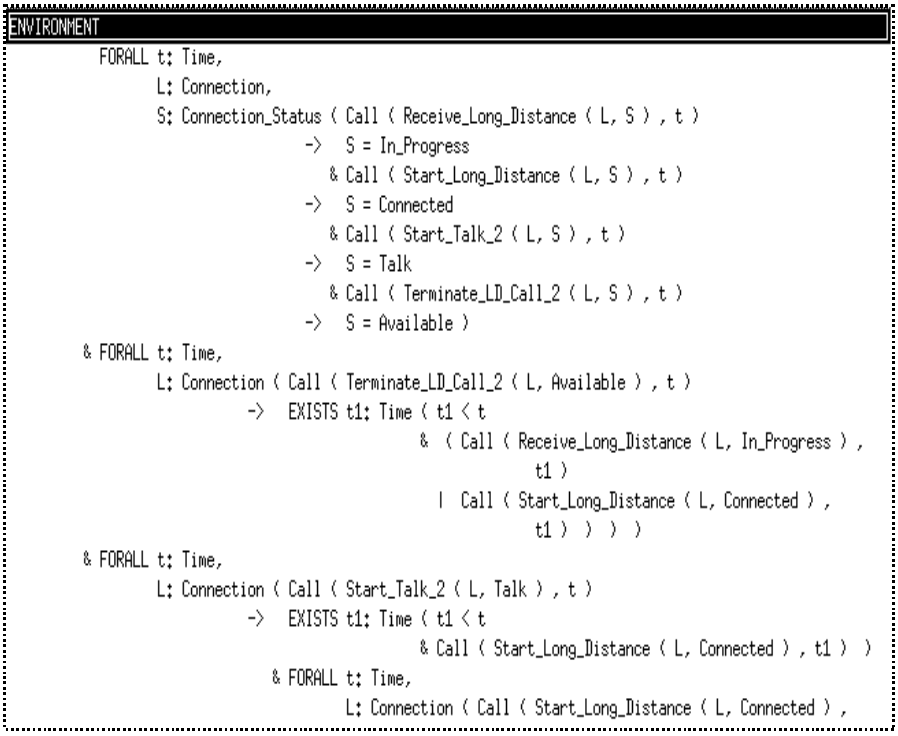

Figure 3-3: formatted form of what was entered

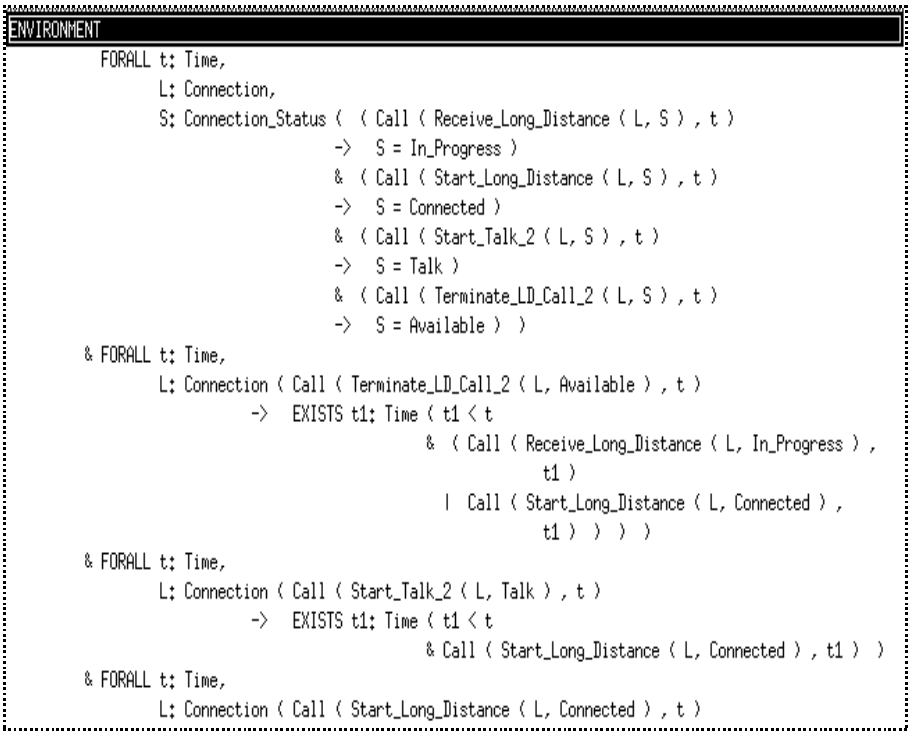

Figure 3-4: formatted form of what was desired

3.5. OK and Parse Error buttons

After formatting, the '**OK**' button may update various internal items, such as symbol tables, etc. In most cases, this information is updated in the parser so if the text is syntactically incorrect, the actions of the update are undefined. For this reason, the user is not allowed to press '**OK**' when the text is in such a state. It may still be desirable, however, to be able to incorporate the changes so the specification can be saved. For this purpose, the '**Parse Error**' button is available. The '**Parse Error**' button basically says "update this section with the understanding that it is grammatically incorrect". The SDE does not correct any errors but either transforms the text into "TRUE" followed by the editor text embedded within a single comment in well formed formula sections or for names will replace what was entered with 'Default_Name'. ASTRAL does not support embedded comments so '/* */' pairs are changed to 'BEGINCOMMENT ENDCOMMENT' to avoid this situation. When the text is edited in the future, these delimiters must be manually changed back to '/* */' pairs.

4. Insert New.. menu

The editor is used to change the text of items already in existence. It cannot, however, manipulate the internal structure of the current specification or composition. For this purpose, the '**Insert New..**' and '**Remove Selected..**' menus are available. '**Insert New..**' allows the user to add various types of items to the current navigation object. The choices include specifications, call generation clauses, processes, levels, transitions, further assumptions sections and except-exit pairs. Objects can only be added when the navigation window is at the appropriate level. To add a specification or call generation clause, the window must be at the composition level. Note that the insert specification option is actually a misnomer. This button brings up the load dialog box (section 7), since when performing a composition, it is assumed that the specifications being composed already exist. To add a process, the navigation window must be at the specification level. To add a level, it must be at the process level. For transitions or further assumptions sections, it must be at the level level and for except-exit pairs, it must be at the transition level. In any other location, the choices will be grayed in the '**Insert New..**' menu. In figure 4 below, '**PROCESS in Current Specification**' is the only option available so the navigation window at this time was at the specification level. When adding a new process, level, or transition, the user can choose to copy from an existing item. For example, when adding a process level that refines another level, it might be easier for the specification writer to copy from the level being refined and then just update the portions that need changing rather than rewriting the entire level which may introduce unnecessary inconsistencies. To copy an item, select that item in the navigation window and then insert an object of the same type. An insertion box is displayed with areas to enter various names. After entering this information, press

the '**copy from ..**' button to complete the operation. Note that any information fields left blank or that are syntactically incorrect will become 'Default_Name'.

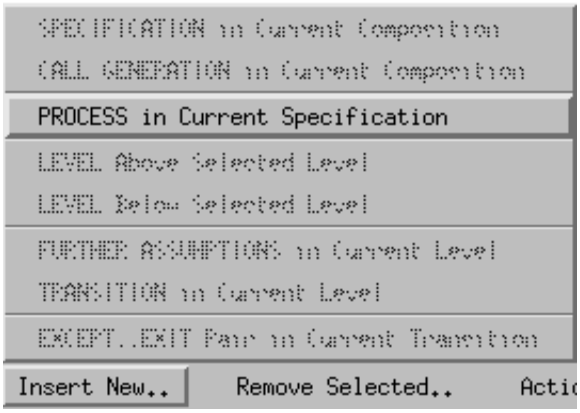

Figure 4: the Insert New.. menu

5 Remove Selected.. menu

The '**Remove Selected..**' button works in a similar fashion to the insertion button. To remove an item, select it in the navigation window and click '**Remove Selected..**'. Then choose the ungrayed option which applies to what is selected. If nothing is selected or if the selection cannot be removed, no option will be available in this menu. There must always be at least one object of each type at the appropriate level. An item cannot be removed if it causes this restriction to be broken. In figure 5, all options are grayed so either the selected item is not one of the choices or it is the "last" of its kind. Before actually removing an object, the SDE displays a dialog box requesting confirmation. '**OK**' proceeds with the removal and '**Cancel**' aborts the operation.

Figure 5: the Remove Selected.. menu

6. Compose/Build

The '**Compose**' button sets the SDE into composition mode. There are three effects this has on the SDE. First, the composition level is made accessible in the navigation hierarchy as the new root. This level contains specifications, call generation clauses, types, defines, constants, and axioms. When the '**Compose**' button is first pressed, the current specification becomes the first specification of the composition and an empty call generation clause is added for each exported transition. Similar clauses are added when other specifications are loaded in using the '**Insert New..SPECIFICATION in Current Composition**' button. Secondly, the '**Generate composition proof obligations to file..**' action is made available in the '**Actions..**' menu (section 13). Finally, the '**Compose**' button becomes the '**Build**' button, described below. When a saved composition is loaded from a file, composition mode is set automatically. Note that when validating a composition, it may not be possible to correct every error that appears without eliminating all name conflicts. Identifiers which have the same meaning across specifications do not need to be explicitly changed to avoid conflicts since duplicates will be removed by the '**Build**' procedure. In this case, however, the user may receive error messages such as: 'WARNING: id "..." declared in specification "..." conflicts with declaration in specification "..." -- assuming first declaration...types incompatible (a_type/a_type)'. Messages in this form where a type incompatibility is reported but the two types given in parentheses have the same name usually signifies that the message can be safely ignored.

The '**Build**' button takes the information provided in the call generation and related clauses of the composition level, along with the specifications being composed, and builds the composite specification as described in [CK 93]. This procedure can only be invoked if all call generation clauses reference an existing transition exported from an existing process and the formula is in the correct form (as indicated by the validation procedure). Although a large portion of the changes required could be incorporated automatically, some of them could not. Specifically, the following *are* done:

- Newly "unexported" transitions are removed from the corresponding process's export clause.
- All references to unexported transition calls in local and global schedules and invariants are replaced by the corresponding "equivalent" formulas constructed from the call generation clauses.
- All parameters are removed from names of unexported transitions and are replaced in the entry-except/exit text by the parameters given in the corresponding call generation clause.

- The "existential version" of each call generation formula is added to the related transition's entry-except clauses and also to the exit clauses if the transition formerly had parameters. The relevant id of the formula is "frozen" to "self". For example, the call generation formula: *FORALL t:time, c:central_control_id, p:phone_id (c.LDOut... <-> c.call(..., t))* is changed to: *EXISTS t:time, c:central_control_id, p:phone_id (c = self & c.LDOut...)* and added to the appropriate central_control transition.
- Any global constants, types, defines, or process instances used in the call generation clauses of a process are imported by that process if not already imported.
- Any variables of other processes used in the call generation clauses of a process are imported by that process.
- The processes clause of the new global specification is set to the processes clause of the composition.
- Global types, defines, and constants of all specifications and the composition are combined to form the corresponding clauses in the new global specification with duplicates removed. Note that it is important for the user to verify that any declaration in these sections which produce name conflicts in validation are identical across specifications because the build procedure only keeps the first instance of any declaration it finds in each section.
- Global axioms, invariants, and schedules of all specifications are combined (via conjunction) to form the corresponding clauses in the new global specification.
- All processes of all specifications are added to the new specification.

The following are *not* done:

- Removal of calls to unexported transitions and related formulas in global and local environments.
- Addition of old environment to new imported variables clause in any process with a relevant call generation clause.
- Change of names in new specification which involve names differing between the old global processes sections and the composition processes section (e.g. a specification contains Phones: array[0..N] of Phone, but the composition has Another_Name: array[0..N] of Phone, so "Phones" needs to be changed to "Another_Name" throughout the new specification)

7. File.. menu

Throughout the editing process, the user may wish to save the current navigation object, load a different file, or begin on a new specification entirely. The file name of the current navigation object is always displayed in the '**Current File**' box (figure 7-1) at the bottom of the SDE. If the navigation object is not yet associated with a file (i.e. has not been loaded or saved as), the box displays 'No File'.

The '**File..**' menu (figure 7-2) has five options. '**New**' sets the SDE into specification mode, loads an empty specification and clears the file name box. '**Save**' saves the current navigation object to the file name shown. This option will be grayed if the file name box contains 'No File'. It is also unavailable if read-only mode (section 8) has been set. '**Save as..**' sets the current file to the pathname selected in the file selection box discussed below and then performs a '**Save**'. '**Load**' brings up the file selection box and the file chosen is loaded into the navigation window and becomes the current navigation object. Finally, '**Quit**' exits the SDE.

Figure 7-1: the current file box

Whenever the current navigation object has been modified since the last new, load, or save, the SDE displays 'SPECIFICATION NOT SAVED' or 'COMPOSITION NOT SAVED' next to the navigation window label as well as next to the options in the file menu that might cause the changes to be lost (i.e. '**New**', '**Load..**', '**Quit**'). If one of these options is chosen, the SDE pops up a dialog box requesting confirmation of the operation. If '**OK**' is pressed, any changes are lost. '**Cancel**' aborts the operation and allows the user to save the changes.

Figure 7-2: the File.. menu

Whenever '**Load..**' or '**Save as..**' is selected, the file selection box (figure 7-3) pops up. The purpose of this box is to select the full pathname of either an existing file or one to be created. The box is divided into five sections: filter, directory, files, selection, and various buttons. '**Selection**' shows the full pathname of the box's current working directory. '**Directory**' shows all directories in that directory. '**Filter**' is the pathname of the working directory plus a mask (using the standard shell '?' and '*' wildcards) that determines which files in the working directory are to be displayed in the files area. '**Files**' shows all files in the working directory that match the filter. As can be seen, all of these items are interconnected around the current working directory. Whenever the working directory is changed, all of them are updated to reflect the change. The working directory can be changed in several ways. The most common method is to double click on one of the directories listed in the directories area. The other way is to edit the pathname in the filter area and press the '**Filter**' button. The

directory entered in this box becomes the working directory. The filter and files area are also interconnected around the file mask. When the filter button is pressed, the files area will reflect the newest mask as well as the new directory, if any.

Figure 7-3: the file selection box

When the pathname in the selection area is the name of a file instead of a directory, the '**Load**' and '**Save**' buttons (depending on which option in the file menu was chosen) are used to load or save to that file. A file is chosen by either manually entering the full pathname into the selection area or by clicking on one of the file names in the files area which will automatically update the selection area pathname to that file. A useful shortcut is to double click on the file in the files area causing the default action (load or save) to be enacted.

8. Options.. menu

The '**Options..**' menu, at present, contains four items as shown in figure 8. The '**Read-only mode**' option toggles write protection on the current navigation object. It is akin to the read-only option in vi whereby the SDE performs as usual, but work can only be saved with the '**Save as..**' button (i.e. it doesn't allow quick save). The '**Save**' button is grayed and unavailable. This mode is toggled automatically if a loaded file is not writable. If read-only is turned off, the SDE will still have a record of whether the current navigation object has been changed either before read-only was set or while it was set. Thus, the normal '... NOT SAVED' warnings will be shown with respect to when the file

was first loaded or last saved to. Read-only mode is reset whenever '**New**', '**Load..**', or '**Save as..**' is executed.

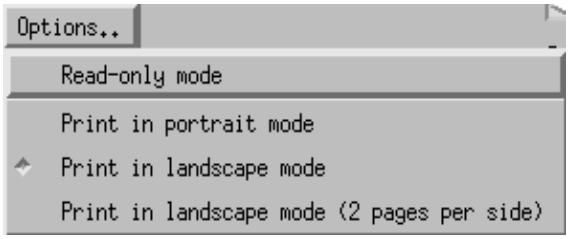

Figure 8: the Options.. menu

The other three items are mutually exclusive print options which determine the command used to print when the '**Dump current work to default printer..**' button (section 13) is pressed. '**Print in landscape mode**' is the default option which selects "wide" printing using 'enscript -rq'. Two landscape pages can be printed on each side using '**Print in landscape mode (2 pages per side)**' which invokes the command 'mpage -2lHfP-' piped to 'lpr'. '**Print in portrait mode**' selects standard printing using 'lpr'. This option is recommended only if neither 'mpage' nor 'enscript' is available, since well formed formulas often exceed 80 characters in width but are rarely more than the 132 characters of landscape mode.

9. Last Event and Instant Help

Throughout the use of the SDE, the '**Last Event**' box at the bottom of the SDE (figure 9), will be constantly updated to display significant events that were just enacted. For example, when read-only mode is entered, the message 'Read-Only Mode Entered' is displayed. The '**Instant Help**' box displays one line help messages when available. For example, if one of the '**Inspect Selected..**' buttons is held down and the pointer is moved over one of the options, the '**Instant Help**' box is updated to display the action that will be performed if the option pointed to is chosen.

Figure 9: the Instant Help and Last Event boxes

10. Browsers

The process, transition, and variable browsers (figure 10-1) enable the user to view various relationships between the three types of items. It is still not clear exactly what impact the browsers

will have on the specification writing process. It is clear, however, that there will certainly be instances when the user may wish to know which processes import a particular variable, which transitions set variables set by a selected transition or any other number of relationships that may be helpful at a particular moment. Uncovering such relationships by hand, however, is a daunting task at best. The browsers are able to make use of the symbol tables maintained during editing to easily ascertain and display the appropriate information.

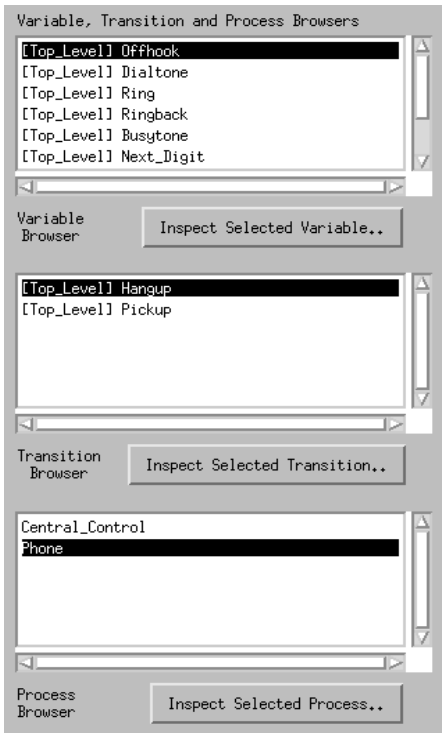

Figure 10-1: the browsers

To begin the query process from scratch, the user selects 'All Processes defined' under '**Inspect Selected Process..**' This option is the only one which does not require a particular item to be highlighted before use and can be used at any time to restart from scratch. In specification mode, it displays a list of all processes defined in the current specification, and in composition mode, it displays all processes in all specifications of the current composition inside the process browser window. After this, all items are queried in the same way. First, an item is selected in the appropriate browser window. Then, the '**Inspect Selected..**' button beneath that window is pressed. At this point, the list of predefined queries is displayed. Figure 10-2, 10-3, and 10-4 display the queries available in the browsers and the exact nature of the information shown for each is given in the appendix.

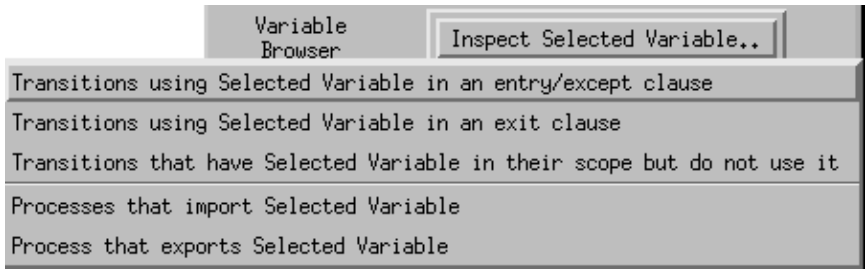

Figure 10-2: variable browser queries

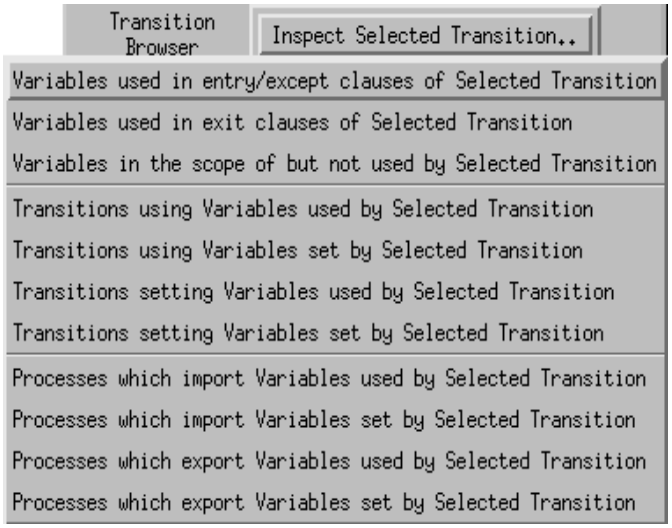

Figure 10-3: transition browser queries

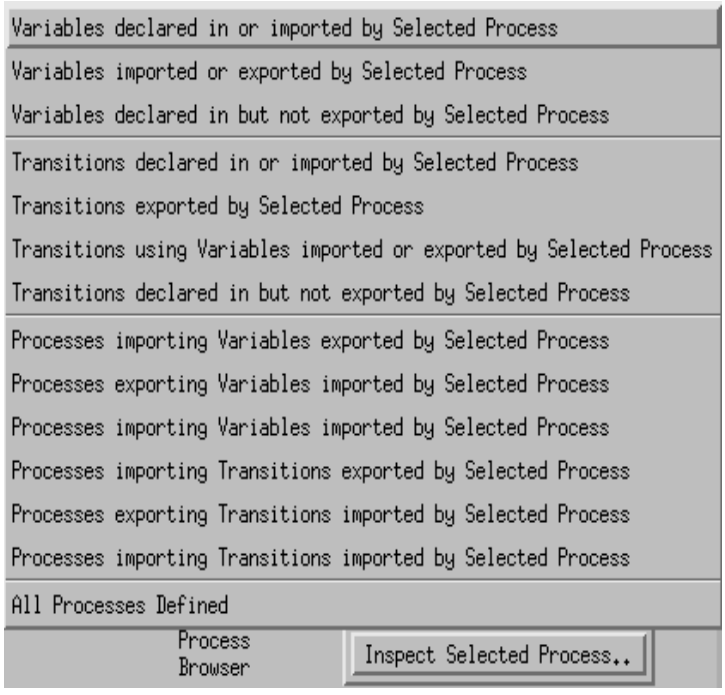

Figure 10-4: process browser queries

Another feature of the browsers is that double clicking on a particular item will "warp" the navigation window to a location associated with that item. For processes, the window will go to the process level of that process. For transitions, the window will go to the actual transition itself. Finally, for variables, the window will jump to the 'VARIABLE' section of the level of the variable selected. Whenever a process, level, or transition is removed, browser items which are related to those objects are deleted from the appropriate browser lists. Figure 10-1 shows a sample browser session. Note that variable and transitions are displayed in the form '[level_name] item_name' where level_name is the level the item was declared at. Different queries use the level information in various ways as indicated in the appendix.

The browsers are not fully accurate. They merely allow the specification writer to obtain a quick and for the most part accurate approximation of the queries desired. The most notable inaccuracy is that defines are currently not taken into account when any use/set queries are issued. In other words, if a define uses a variable and the transition proceeds to use the define, that variable will not show up in the browser. Although this simple case could be handled, one could easily imagine defines using defines using defines, etc. such that the chain would be difficult if not impossible to follow. The browsers also cannot, for obvious reasons, take into account user errors such as misspellings or missing declarations, etc. but these types of errors are within the domain of the validation procedure.

11. Validation

One of the most valuable tools that the SDE has to offer is the validation mechanism. The validation procedure performs type and scope checking on each section in the current navigation object. It also does a few other simple checks such as warning of primed variables which are not in a constraint clause or the exit clause of a transition. When a specification validates without error, it indicates that the specification may be ready for the next stage in its development, namely consistency proofs (section 13). Similarly, when a composition validates without error, it may be ready for the construction of the new composite specification (section 6).

The main strength of validation is when there *are* errors in the specification. The procedure provides a user friendly way to quickly locate and "warp" to the errors in the navigation window. The 'Validation Results' window (figure 11) is the feedback the user receives after pressing the '**Validate**' button

. There are 3 types of results that may appear in this window. The most common type of message is of the form 'line XX in <section_name>: <error>' where XX is a number, <section_name> is either a transition name or section name, and <error> is a string indicating the exact

problem found. Any message of this form can be double clicked on and the navigation window will go to the given location and highlight the offending line. Once a section is changed, however, the line number may be invalid. As was the case with the editor, the line number may also be off by one due to the way the parser computes its line numbers. Another message is of the form 'line ?? in <section_name>: <error>'. In this case, it is not clear exactly what line number the error occurred at. For example, name conflicts when validating compositions may be associated with a number of different sections. Validation will report 'line ?? in <composition_name>: WARNING: name conflict -- ...'. Double clicking on this type of message moves the navigation window to line 1 of the appropriate section. Finally, the least common messages are those surrounded by "—" which indicate information that is not associated with any section in particular, usually a report on the status of the validation procedure. Double clicking on these items has no effect.

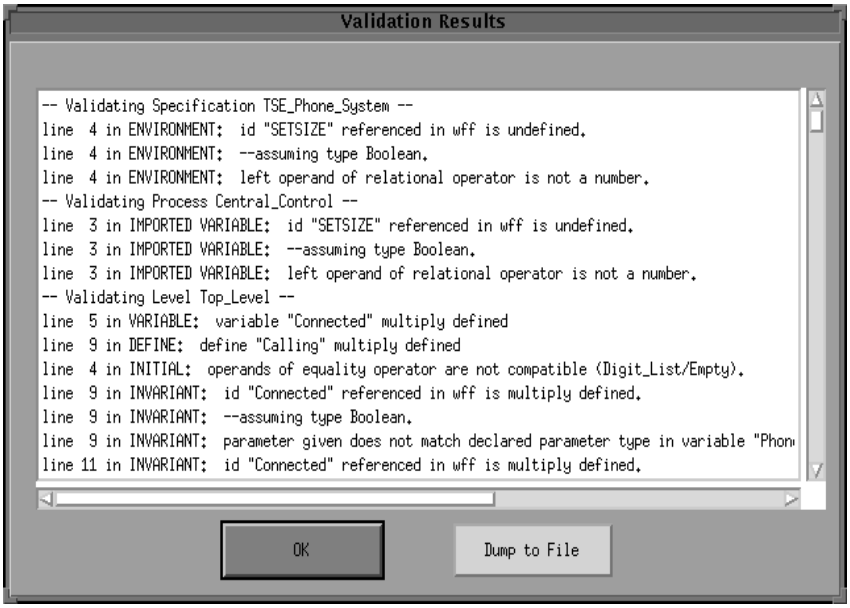

Figure 11: the Validation Results window

The '**Dump to File**' button copies all the information from the validation window to the file 'error.output' so the user may peruse the results off-line. If the file already exists, a warning will appear and request confirmation before the file is overwritten. The '**OK**' button brings down the validation window.

12. Search

Since it is oftentimes the case that a mistake made in one location is similarly repeated elsewhere, the search procedure provides a convenient way to replace multiple occurrences of the same

validation error. The errors that are most suited for this type of repair are those that involve simple identifier problems such as name conflicts in composition validation. The '**Search**' button

brings up the search and replace box shown in figure 12. This box allows the user to search for the regular expression currently appearing in the '**Find regular expression:**' box in all names and text of the current navigation object. The regular expression syntax supported is equivalent to that used by the emacs search routines described in the emacs user's manual.

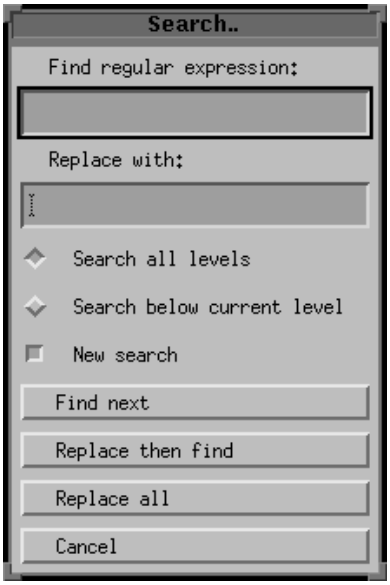

Figure 12: the search and replace box

12.1. Search options

The current navigation object can be searched either in its entirety or from only those levels at or below the level shown in the navigation window. The '**Search all levels**' option chooses the former while the '**Search below current level**' chooses the latter. Note that these two options are only applicable when '**New search**' is set and one of the search actions is performed. Otherwise, the search performs by how the options were set when the search was first initiated with a '**New search**'. The '**New search**' option can be toggled and will only initiate a new search with the options given when it is set and a search action is invoked, at which time it will be automatically disabled. Note that this option is assumed and need not be set explicitly for the '**Replace all**' action.

12.2. Search actions

All search actions are performed with respect to the options selected when the current search was first initiated as described in the previous section. '**Find next**' searches for the next occurrence of the current regular expression. '**Replace then find**' replaces the last occurrence of the current regular expression with the current replacement text located in the '**Replace with:**' box and then finds and displays the next match. Note that if the text that the last occurrence was found in has been modified using the editor since the last search and the current regular expression no longer occurs at the same location as it did, this action becomes a '**Find next**'. '**Replace all**' replaces all occurrences of the current regular expression with the current replacement text according to the search options selected. The user should be cautious about the regular expression used with the '**Replace all**' action. To illustrate, suppose there are two similar identifiers: "phone_id" and "set_of_phone_id". The user may think to use the regular expression "[^a-zA-Z0-9_]phone_id" so that only "phone_id" is replaced. The problem is that not only will "phone_id" be replaced, but also the non-alphanumeric character ([^a-zA-Z0-9_]) preceding it which is most likely an undesired consequence. If there is a conflict such as this example, it is safer to use '**Replace then find**' to replace "phone_id" and '**Find next**' to ignore "set_of_phone_id". Both replacement actions will abort if the current replacement text causes a parse error within the text but any replacements up to the appearance of the error will stay replaced. In other words, there is no option to "unreplace" the replacements that occurred up to the point of the error. If the new text is syntactically acceptable, it will be automatically reformatted after any replacements have occurred. '**Cancel**' brings down the search box while keeping the current settings.

13. Actions.. menu

To complete the specification writing process, the user may wish to obtain a printout of the current navigation object or to generate the proof obligations necessary to prove consistency. The format of an SDE saved file is not equivalent to just the text displayed in the various navigation windows, since it also contains other characters to help with section separation, etc. The '**Dump work to file..**' and '**Dump work to default printer..**' buttons in the '**Actions..**' menu (figure 13) produce the equivalent text version of the current navigation object. '**Dump work to file..**' produces a file called 'spec.output' if the SDE is in specification mode or 'comp.output' in composition mode. These files will be placed in the directory the SDE was initialized in. The SDE will warn the user if the files already exist so they can be moved out of harms way if desired. Otherwise, the files will be overwritten. The '**Dump work to default printer..**' button uses the command associated with the current print option (section 8) to print the current navigation object on the default printer. '**Generate intralevel proof obligations to file..**' constructs the intralevel

proofs described in [CKM 94] and outputs them to the file 'intra.output'. Similarly, '**Generate interlevel proof obligations to file..**' constructs the interlevel proofs from [CKM 95] and outputs them to 'inter.output'. In order to generate the interlevel proofs, the current specification must be valid or in the case of compositions, each specification must be valid (although the full composition itself is not required to be valid). The '**Generate composition proof obligations to file..**' button is only available when the SDE is in composition mode. This action produces the proofs described in [CK 93] and saves them in the file 'comp_proofs.output'. Finally, '**About ASTRAL..**' displays miscellaneous information about the SDE.

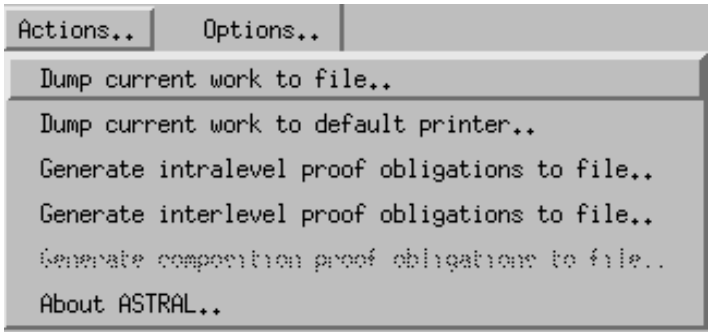

Figure 13: the Actions.. menu

14. References

[CK 93] Coen-Porisini, A. and R.A. Kemmerer, "The Composability of ASTRAL Realtime Specifications", *Proceedings of the International Symposium on Software Testing and Analysis*, Cambridge, Massachusetts, July 1993.

[CKM 94] Coen-Porisini, A., R.A. Kemmerer and D. Mandrioli, "A Formal Framework for ASTRAL Intra-level Proof Obligations", *IEEE Transactions on Software Engineering*, Vol. SE-20, No. 8, August 1994.

[CKM 95] Coen-Porisini, A., R.A. Kemmerer and D. Mandrioli, "A Formal Framework for ASTRAL Inter-level Proof Obligations", *Proceedings of the Fifth European Software Engineering Conference*, Barcelona, Spain, September 1995.

Appendix

1. Process browser queries

'*Variables declared in or imported by Selected Process*'

Display all variables declared in all levels of the selected process along with imported variables that are exported by another process.

'*Variables imported or exported by Selected Process*'

Display all variables which are declared in the top level of the selected process and exported along with all imported variables that are exported by another process.

'*Variables declared in but not exported by Selected Process*'

Display all variables declared in the top level of the selected process which are not exported along with all variables declared in levels below the top level.

'*Transitions declared in or imported by Selected Process*'

Display all transitions declared in all levels of the selected process along with imported transitions that are exported by another process.

'*Transitions exported by Selected Process*'

Display all transitions declared in the top level of the selected process and which are exported.

'*Transitions using Variables imported or exported by Selected Process*'

For each variable declared in the top level of the selected process and exported along with all imported variables that are exported by another process, for each process importing the variable (or declaring and exporting it), display all transitions in all levels (top level) which have the variable in their entry/except or exit text.

'*Transitions declared in but not exported by Selected Process*'

Display all transitions declared in the top level of the selected process which are not exported along with all transitions declared in levels below the top level.

'*Processes importing Variables exported by Selected Process*'

Display all processes that import at least one variable exported by the selected process.

'*Processes exporting Variables imported by Selected Process*'

Display all processes that export at least one variable imported by the selected process.

'*Processes importing Variables imported by Selected Process*'

Display all processes that import at least one variable also imported by the selected process.

'*Processes importing Transitions exported by Selected Process*'

Display all processes that import at least one transition exported by the selected process.

'*Processes exporting Transitions imported by Selected Process*'

Display all processes that export at least one transition imported by the selected process.

'*Processes importing Transitions imported by Selected Process*'

Display all processes that import at least one transition also imported by the selected process.

'*All Processes Defined*'

Display all processes defined in the current specification regardless of what is selected.

2. Transition browser queries

'*Variables used in entry/except clauses of Selected Transition*'

Display all variables declared in the same level as the selected transition or imported by the process the transition resides in which appear in the entry/except text of the selected transition.

'*Variables used in exit clauses of Selected Transition*'

Display all variables declared in the same level as the selected transition or imported by the process the transition resides in which appear in the exit text of the selected transition.

'*Variables in the scope of but not used by Selected Transition*'

Display all variables declared in the same level as the selected transition or imported by the process the transition resides in which do not appear in the entry/except or exit text of the selected transition.

'*Transitions using Variables used by Selected Transition*'

For each variable in the entry/except or exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then for each process importing the variable (or declaring and exporting it), display all transitions in all levels (top level) which have the variable in their entry/except or exit text; otherwise, display all transitions declared in the same level as the selected transition which have the variable in their entry/except or exit text.

'*Transitions using Variables set by Selected Transition*'

For each unprimed variable in the exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then for each process importing the variable (or declaring and exporting it), display all transitions in all levels (top level) which have the variable in their entry/except or exit text; otherwise, display all transitions declared in the same level as the selected transition which have the variable in their entry/except or exit text.

'*Transitions setting Variables used by Selected Transition*'

For each variable in the entry/except or exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then for each process importing the variable (or declaring and exporting it), display all transitions in all levels (top level) which have the unprimed variable in their exit text; otherwise, display all transitions declared in the same level as the selected transition which have the unprimed variable in their exit text.

'*Transitions setting Variables set by Selected Transition*'

For each unprimed variable in the exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then for each process importing the variable (or declaring and exporting it), display all transitions in all levels (top level) which have the unprimed variable in their exit text; otherwise, display all transitions declared in the same level as the selected transition which have the unprimed variable in their exit text.

'*Processes importing Variables used by Selected Transition*'

For each variable in the entry/except or exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then display all processes that import that variable.

'*Processes importing Variables set by Selected Transition*'

For each unprimed variable in the exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then display all processes that import that variable.

'*Processes exporting Variables used by Selected Transition*'

For each variable in the entry/except or exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then display the process that exports that variable.

'*Processes exporting Variables set by Selected Transition*'

For each unprimed variable in the exit text of the selected transition, if the variable is imported by the process the transition resides in or if it has been declared in the top level of that process and exported, then display the process that exports that variable.

3. Variable browser queries

'*Transitions using Selected Variable in an entry/except clause*'

Display all transitions declared in the same level as the selected variable that have it in their entry/except text and if the variable is in the top level and exported by the process it resides in, then also display transitions in all levels of processes that import the variable and which have it in their entry/except text.

'*Transitions using Selected Variable in an exit clause*'

Display all transitions declared in the same level as the selected variable that have it in their exit text and if the variable is in the top level and exported by the process it resides in, then also display transitions in all levels of processes that import the variable and which have it in their exit text.

'*Transitions that have Selected Variable in their scope but do not use it*'

Display all transitions declared in the same level as the selected variable that do not have it in their entry/except or exit text and if the variable is in the top level and exported by the process it resides in, then also display transitions in all levels of processes that import the variable and which do not have it in their entry/except or exit text.

'*Processes importing Selected Variable*'

If the selected variable is in the top level of the process it resides in and is exported by that process, then display all processes that import that variable.

'*Process exporting Selected Variable*'

If the selected variable is in the top level of the process it resides in and is exported by that process, then display that process.