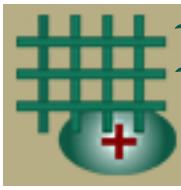


RepGrid • WebGrid • RepGrids
RepNet • RepDoc • RepScript

# Rep Plus

## Conceptual Representation Software

# RepGrid Manual

### Eliciting, Entering, Editing and Analyzing a Conceptual Grid

May 2021

Brian R Gaines and Mildred L G Shaw

http://cpsc.ucalgary.ca/~gaines/repplus/

# Contents

# 1 Conceptual Grids

To support the application of his ***personal construct psychology***, George Kelly (1955) developed a method for eliciting a person, or group's, conceptual framework that he termed a ***conceptual grid*** (p.301). He described it as providing a ***cybernetic model*** (p.302) of a person's ***psychological space*** (p.301)—emphasizing his rejection of the dichotomy between constructivism and behaviorism, one of the many that he explicitly transcended.

Kelly's grid elicitation and analysis techniques enable one to model personal, or social, conceptual frameworks for diverse domains by eliciting the bipolar ***constructs*** through which people differentiate significant ***elements*** in a particular domain. His primary interest was clinical psychology and he developed a particular form of the grid to model a clients' models of their immediate social world in which people were significant to them in a variety of roles. He termed this a ***role repertory grid***, often abbreviated to ***repertory grid*** or ***repgrid***. In much of the personal construct psychology literature these terms became used more generally to denote Kelly's generic ***"conceptual grid,"*** but his general term is more descriptive.

Kelly's grid method is applicable in any domain and has been used for conceptual modelling across a very wide range of disciplines such as clinical psychology (Leach et al., 2001), psychiatry (Kirkcaldy et al., 1993), education (Pope and Denicolo, 2001), management studies (Tan et al., 2009; Bellman, 2012; Rad et al., 2013), consumer studies (Mireaux et al., 2007), architectural studies (Tofan et al., 2014), and software engineering (Young et al., 2005) including requirements elicitation (Shaw and Gaines, 1996a; Dey and Lee, 2017).

## 1.1 RepGrid, WebGrid and RepGrids

The ***RepGrid***, ***WebGrid***, and ***RepGrids*** tools provide the capability to elicit, enter, edit and analyze conceptual grid data, and to reflect back the underlying conceptual representations in graphic form. They can be scripted to offer interactive dialogs and analyses, and includes scripts for Shaw's (1980) conversational elicitation, and for the entry of grids that have been elicited through interviews and other methods (Jankowicz, 2004; Fransella et al., 2004; Fromm, 2004; Caputi, 2011). The analyses present grids in a way that reflects their ***meaning*** in order to promote discussion, understanding, decision-making, conflict mediation, and further elicitation.

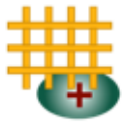
In the 1980s Kelly's conceptual grids became recognized as powerful tool for eliciting knowledge from domain experts in order to develop computer-based ***expert systems*** and were enhanced in various ways to support such applications. Most of the enhancements are consistent with Kelly's (1955) exposition of personal construct psychology and, in particular, they may be used to exemplify his ***fundamental postulate,*** that psychological processes are driven by the need to ***anticipate*** events.

RepGrid includes such enhancements, notably a wider range of data types for grids, the capability to collect and store additional data as part of a grid file, and the ability to use the conceptual structures in grids to ***classify*** elements in a variety of ways through ***classes*** defined in terms of Kelly's (1955, p.121) ***"intersect of properties"*** and having subordination and preference relations to one another.

These enhancements allow RepGrid to be used for teaching and research in ***cognitive science, artificial science*** and ***knowledge management***. However, the user interface has been designed such that the new capabilities are not intrusive and the program can be used as a conventional grid elicitation and analysis tool without involving them. For researchers in ***personal construct psychology,*** the enhancements also enable Kelly's (1955, p.121) notions of how people anticipate events to be explored.

RepGrid is designed to be ***open architecture*** through its scripting capabilities which enable its functionality to be used by other programs, and its own capabilities to be modified and extended by researchers. Like the other Rep Plus tools it incorporates the ***RepScript*** tool and provides programmable access to its own functionality, statistical techniques, the graphic capabilities of ***RepNet,*** and so on.

# 2 Opening, saving and creating grid files




An orange *grid* icon shown on the right is used for a RepGrid file. It contains lists of the elements and constructs, ratings of elements on constructs, class specifications, analysis parameters and numerous ancillary items of information such as the purpose of the elicitation, name of elicitation, date and time of elicitation, and so on. It may contain user-defined data specific to a particular study, and system-defined data relating to parameters chosen, window size and position, and so on, supporting RepGrid and WebGrid operation. The data is held in a text format that is both human and machine readable.

## 2.1 Opening a grid file

Clicking on the *Open* button in the *RepGrid* panel of the *Rep Plus Manager* window (Figure 1) brings up a dialog for opening a file, with grid files indicated by their distinctive icons. If a file is selected RepGrid will attempt to open it as a grid, reporting an error if the data cannot be interpreted as such. RepGrid can decode grids files in most of the many formats that have been used by past and current generations of grid programs (including a WebGrid page saved as the HTML source), but those it saves are not necessarily backwards compatible.

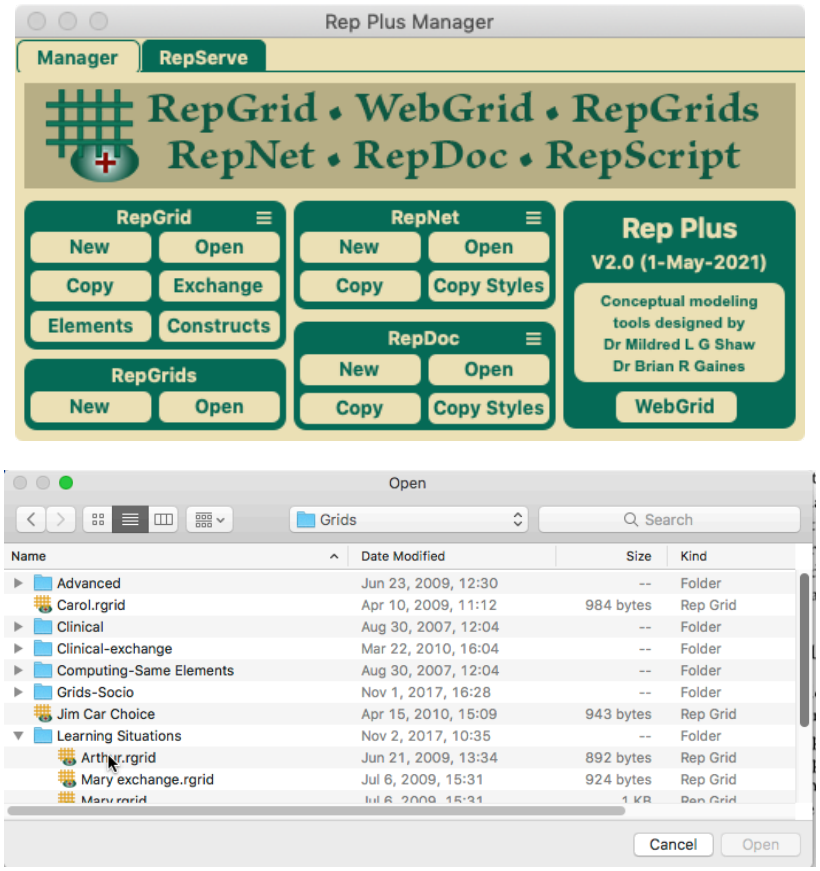



Figure 1: Opening a grid file from the Manager Window

All Rep Plus files contain a file type identifier and grid files may also be opened by using the ***Open*** (command-O) or ***Open Recent*** commands in the ***File*** menu, double-clicking on a grid file, or dragging it to the Rep Plus application icon or Manager window.

## 2.2 Save, Save As

Selecting the ***Save*** option in the ***File*** menu or keying command-S, may be used to save the grid data to its existing file. The ***Save As*** option may be used to save the grid to a new file.

When a grid is saved the analysis parameters last used and the states of the analysis tools and scripts menus are saved with it. They are restored when the grid is re-opened so that if particular analyses or scripts are being used for that grid they do not have to be set up each time it is opened. The RepGrid window size and position is also saved and restored, as are the column widths of the various tables of grid data.

## 2.3 Undo

When grid data is changed the state of the grid before the change is recoded, the ***Undo*** item in the ***Edit*** menu becomes active, and selecting it or keying command-Z restores the state of the grid before change. The last fifty state changes since the grid was opened are recorded supporting multiple undoing. This information is discarded when the grid is closed.

## 2.4 New

Selecting the ***Grid*** option in the ***New*** submenu of the ***File*** menu, or keying command-N, creates a new grid with standard default values. Clicking in the ***New*** button in the ***RepGrid*** area of the ***Rep Plus Manager*** window has the same effect

The grid that is created is a copy of the current default grid stored in the ***GridScripts*** directory in the ***RepPlus*** directory in the ***Documents*** or ***MyDocuments*** directory, or in the ***GridScripts*** directory in the same directory as the ***Rep Plus*** application where a default grid is preinstalled.

Users may store as many potential default grids as they wish in the ***GridScripts*** directory in the ***RepPlus*** directory in the ***Documents*** or ***MyDocuments*** directory. This enables a user to set up a default grid with different parameter defaults, such as terms for elements and constructs, analysis parameters and graphic styles, or even default initial elements and constructs. They can select which one is to be copied when a new grid is created by clicking in the ***Menu*** symbol (☰) in the ***RepGrid*** panel of the ***Manager*** window (Figure 2).

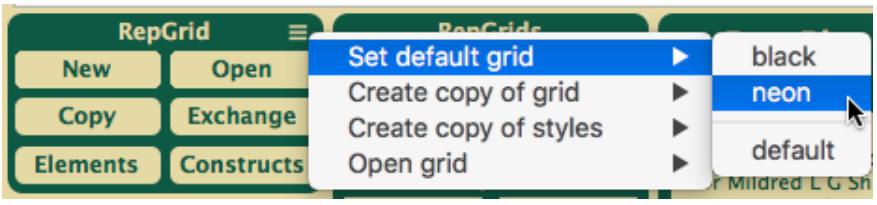



Figure 2: Setting the default grid to be copied when creating a new grid

The hierarchical popup menu shows the available grids in the submenu on the right. The grids ***black*** and ***neon*** are user-developed for purposes of presentation. The grid ***default*** is system installed and designed for use in publications that will be published in colour but often printed in monochrome.

The main menu on the left provides three options:

**Set default grid** Select which of the grids on the right is the default to be copied when a new grid is created;

**Create copy of grid** create a copy of the selected grid on the right—a useful alternative to changing the default grid when only one new grid of a different type is required;

**Create copy of styles** create a copy of the selected grid on the right containing only the styles—useful if some default grids contain elements and/or constructs;

**Open grid** Open the selected grid for editing—making it simple to access the possible default grids in order to make changes to them.

Users do not need to access these capabilities and can simply use the ***New Grid*** command in the ***File*** menu or ***Manager*** window to create an empty grid that is a copy of the system-installed default grid. However, in complex projects the capability to switch between default grids may be very useful to help organize the workflow and help to ensure uniformity in the styles of presentations.

## 2.5 Copy, Exchange, Elements, Constructs—Open dialogue or drag and drop

It is often required to open a copy of a grid with only the elements or constructs or with both but unrated. These options are available through four buttons in the ***RepGrid*** panel of the ***Manager*** window:

**Copy** Open a copy of a grid;

**Exchange** Open a copy with the ratings all set to be open;

**Elements** Open a copy with only the elements;

**Constructs** Open a copy with only the constructs.

If the button is clicked then a file selection dialogue appears that may be used to specify the grid. Alternatively, a grid file may be dragged to the button (Figure 3)..

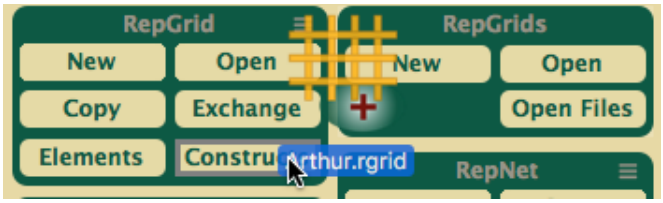



Figure 3: Dragging a grid file to one of the buttons in the ***RepGrid*** panel

Drag and drop is also effective with the ***New*** and ***Open*** buttons, in the first case opening a copy and in the second the original grid.

# 3 Editing a grid

Clicking on the file named *Arthur* in the file open dialog opens that grid file in the *Options* tab of a RepGrid window (Figure 4).

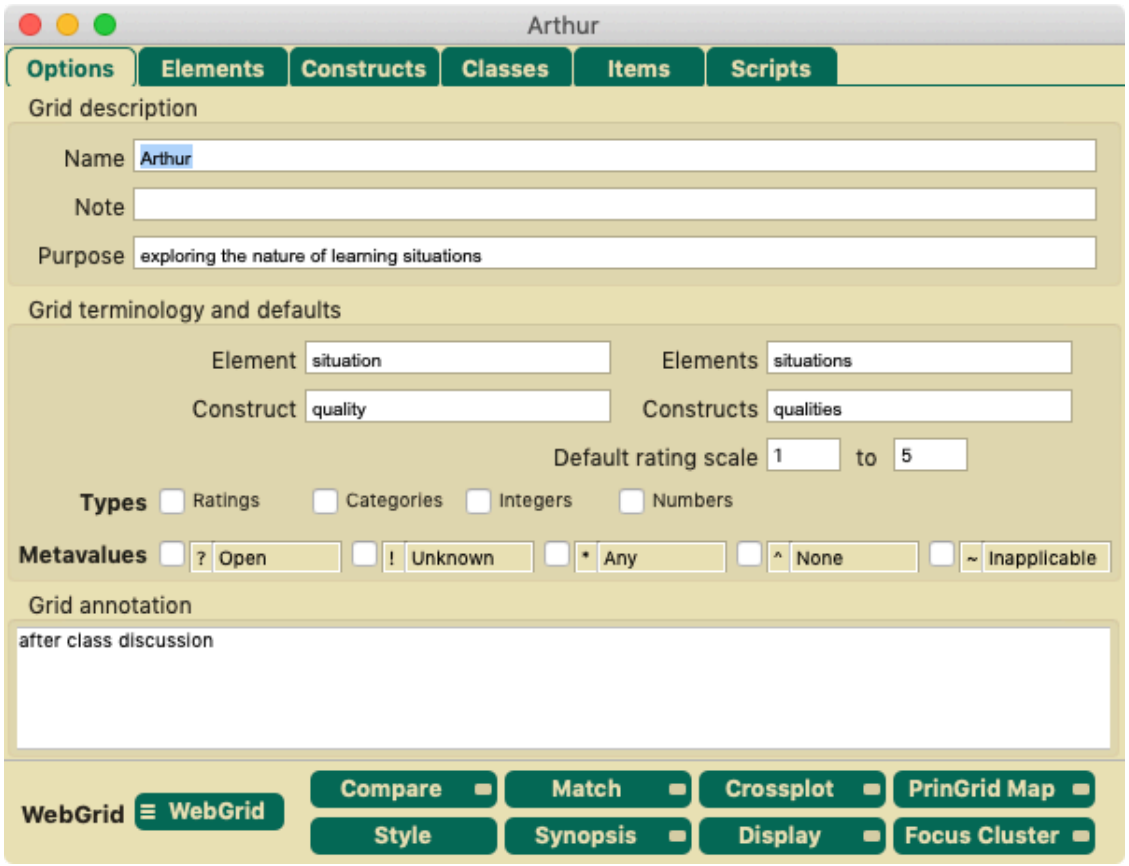



Figure 4: Opening a grid file—the RepGrid *Options* pane

The tab ribbon along the top lets allows one to select any one of the panes: *Options, Elements, Constructs, Classes, Items,* and *Scripts.* The two rows of buttons at the bottom provide access to a range of grid analyses (\$5).

When the data in a RepGrid window is changed one can save the changed data in its original file through the *Save* command (or command-S) in the *File* menu. The *Save As...* command in the File" menu allows one save the grid in a different file. RepGrid supports multi-level undo through the *Undo* command at the top of the *Edit* menu (or command-Z) which becomes active if grid data is changed. Only changes in grid data are recorded as undoable and, since the grid files also keep track of the window position, the *Save* command is always available. If the *Undo* command is active and one attempts to close a RepGrid window without saving the grid data then a warning is given providing the opportunity to save the data before the window is closed.

The RepGrid window opens with the *Options* pane showing, and clicking on one of the other tabs brings its pane into view. The following sections describe the functions of each pane.

## 3.1 Options pane

The ***Options*** pane shows some overall features of the grid and allows them to be entered or edited. It has three panels: ***Grid Description, Grid Terminology and Defaults,*** and ***Grid Annotation*** (Figure 4).

### *3.1.1 Grid description*

The ***Grid Description*** panel comprises three items. The ***Name*** field is intended to identify the person from whom the grid was elicited. This is used to identify the grid in the titles of the analyses.

Since several grids might be elicited from the same person, a ***Note*** field is provided to allow further identification. The note, and other identifiers such as the date and time when the grid was created, may be specified to be included in parentheses after the name ***Arthur*** in analyses (§6).

The ***Purpose*** field is used to express the purpose or context for eliciting the grid. Computer-based elicitation tools use this field to remind the user of the reason why they are developing a grid, and the analysis tools add this to the title to show the purpose or context of the elicitation.

### *3.1.2 Grid terminology and defaults*

The ***Grid Terminology and Defaults*** panel comprises seven items. In grids the entities being construed are termed ***elements*** and the dimensions of construing, their perceived characteristics, are termed ***constructs.*** These are somewhat technical terms, and it is often better for users to substitute more colloquial terms appropriate to the domain or topic under consideration. In this grid Arthur, or the person facilitating the elicitation, has chosen to use the term ***situation*** for ***element*** and ***quality*** for ***construct.***

### *3.1.3 Default rating scale*

The default scale for the ***Ratings*** data type in RepGrid may be set to range over a set of integers from a minimum of 100 to a maximum of +100. Scales of 1 to 5, 7 or 9 are commonly used. A two-valued rating scale of 1 to 2 forces the user to make binary distinctions without 'shades of gray.' Using 1 to 3 provides a middle option that allows a user to express neutrality between the two poles of a construct. However, users often prefer to have the further gradations of a 1 to 5, 1 to 7 or 1 to 9 scale available to them. In the literature one will also find a variety of other scales used, such as a −7 to +7 scale or a 1 to 11 scale.

One may change the rating scale as one adds constructs, and thus have constructs with different rating scales in the same grid. RepGrid will analyze such grids correctly since it rescales the constructs to a common range as part of its analyses. However, having different rating scales for different constructs is not a common usage of grids, and we do not recommend that this feature be used except with experienced users when it is appropriate to their project.

### 3.1.4 Grid annotation

The ***Grid Annotation*** panel at the bottom provides a multi-line text field to store any annotation that may be useful. It is not shown in any of the analyses and may be used to help keep track of any data relevant to the grid. If there is extensive annotation the window may be enlarged to display it.

### 3.1.5 Rating scale data types: ratings, categories, integers, numbers

The ***Data Types*** check boxes control what data types are offered. If none are checked then conventional rating scales only are offered, and advanced features such as construct names and rating scale categories, weights, and so on, are not made available so as to simplify usage. If any type or combination of types is checked then those types are offered together with the more advanced features.

Four data types are currently supported:-

**Ratings:** integer ratings in the range -100 to +100 with the default scale as specified, where one may specify category labels (including the pole names) for ranges of ratings;

**Categories:** labeled categories, implemented as a subtype of ***Ratings*** where the rating scale is derived from the number of categories specified;

**Numbers:** floating point numbers representing the value along a dimension such as height, weight or time;

**Integers:** integer numbers, implemented as a subtype of ***Numbers*** restricted to have no decimal places, a significant distinction conceptually but not technically.

### 3.1.6 Metavalues in ratings: open, unknown, any, none, inapplicable

RepGrid also supports the analysis of grids having meta-values for some of the ratings. The ***Metavalues*** check boxes control what meta-values will appear on the rating scale popup menus that are made available to users for grid data entry. The possible meta-values are:-

**? Open:** indicating that a rating has not yet been requested or entered;

* **Any:** indicating that any of the metavalues or values might apply, usually used in the specification of an ***ideal element*** to indicate that a construct is irrelevant;

^ **Inapplicable:** indicating that the construct is not applicable to the element, perhaps because the element does not fall under the the pole of a another construct superordinate to that under consideration;

**! Unknown:** indicating that the construct is applicable but the specific rating is unknown or irrelevant;

~ **None:** indicating that the construct is applicable but none of the available ratings is appropriate, usually only used with ***Categories*** when a suitable one has not yet been made available.

The single characters such as as *?* and *!* before the names provide a succinct representation of a metavalue when a grid is displayed. The attached description is intended a reminder of the meaning of the metavalue, understandable to those facilitating the use of grids.

For purposes of display to users, both the single character and the description may be edited by the facilitator to be more meaningful to users. The description that appears to end users may be modified by the facilitator to be more meaningful to them, e.g. ***Open*** might be changed to ***Enter a rating***. When the grid is stored the standard character for the metavalue is used, but when it is displayed the alternative specified is used.

The meta-values and rating types are related through a logical structure that may itself be represented as a tiered structure of constructs (Figure 5).

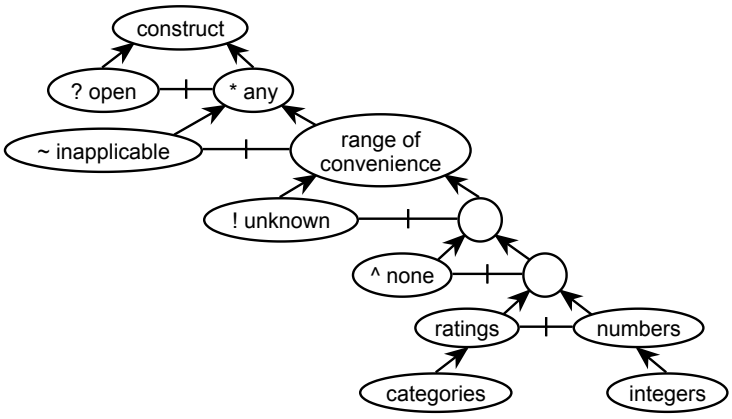



Figure 5: Type hierarchy of meta-values and rating types

The ***construct*** node at the top represents a partial definition of a construct that is completed through the specification of its ***range of convenience*** below (supporting Kelly's (1955) notion that constructs with the same name may have differing ranges and foci of convenience). The first three metavalues reflect this structure allowing a construct to be ***mentioned*** but possibly not ***used***: ***open*** when no data has been entered; ***any*** when the construct is not relevant to a distinction because it is inapplicable or its value does not matter, typically used in specifying ***ideal elements***; and ***inapplicable*** when the construct is not applicable to an element, a distinction important to representing the ordinal, or ***laddered*** (Reynolds and Gutman, 1988; Corbridge et al., 1994; Korenini, 2014), structure of constructs in grid form. e.g. in a grid on ***holiday resorts***, the construct ***poor beaches—good beaches*** may be inapplicable to inland resorts but still very significant in comparing waterside resorts.

The final two metavalues indicate that an element is within the range of convenience of a construct but: its rating is ***unknown*** or, possibly, irrelevant to the purpose of the elicitation; or corresponds to ***none*** of the ratings, usually categories, made available.

In computing matching scores, ***any*** matches any value or metavalue, ***inapplicable*** matches ***inapplicable***, and ***none*** is treated as an alternative middle value in relation to the poles but at the greatest possible distance from the actual middle value of the scale. Otherwise, metavalues match no other value, returning the greatest possible distance as if they were opposite poles.

In most conventional grid elicitation, the only metavalue used is ***?*** to indicate that a rating has not yet been entered. However, sometimes clients are given options such as ***any*** and ***none,*** or ***both*** and ***neither,*** and RepGrid supports such metavalues.

If metavalues are not used then clients typically use the middle value of a rating scale as a neutral value representing all the possible metavalues above, and the distinction between them is not captured in the grid (Yorke, 1978, 1983). This seems to to be adequate in many situations, but the capability to refine grids with more explicit metavalues is significant when using them to develop computational models of anticipatory processes (Gaines and Shaw, 1993a,b) and to represent hierarchical knowledge structures (Shaw and Gaines, 1998).

## 3.2 Elements pane

Clicking on the ***Elements*** tab brings up the elements pane (Figure 6). This lists the numerical order of the elements in the grid in the column on the left, and their names in the next column.

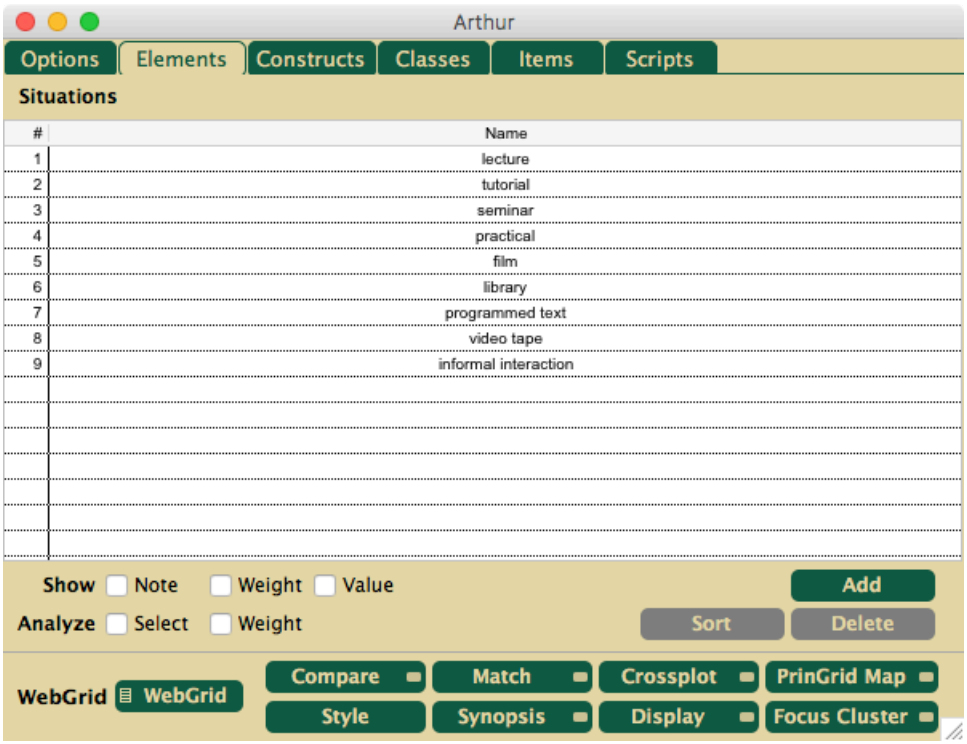



Figure 6: RepGrid ***Elements*** pane showing element numbers and names

Rows may be selected by clicking in the number column on the left which selects and highlights the row clicked. Multiple rows may be selected by holding down the ***shift*** key for contiguous selections and the command key for non-contiguous selections, consistent with the normal conventions of the Mac and Windows operating systems. One purpose of selecting rows is to allow them to be deleted by pressing the ***delete*** or ***backspace*** key. Others are the selection of only part of a grid for display and analysis (§5.8), and to enable the data in them to be dragged to another application such as a net in RepNet or a document in a word processor.

### 3.2.1 *Element annotation and weights*

The element names are always shown, and the associated note, weight and value fields may also be shown dependent on which boxes are checked in the row on the left under the data. Figure 7 shows the ***Elements*** pane with all the fields activated and some elements selected.

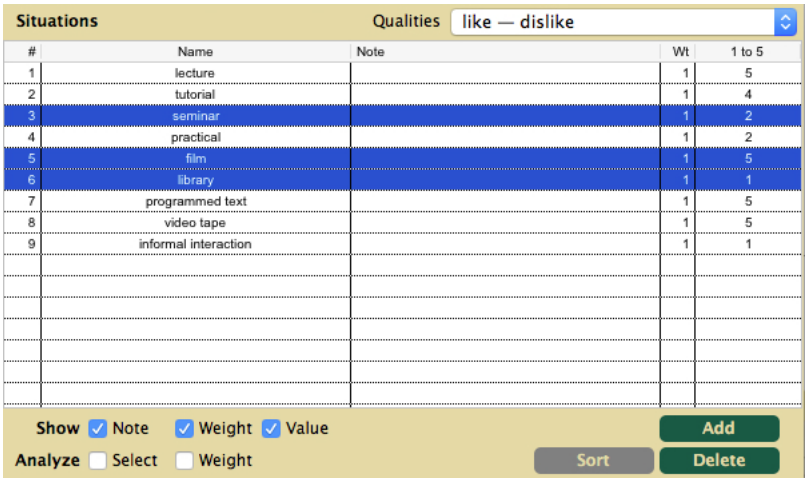



Figure 7: RepGrid ***Elements*** pane with all fields showing

The note field may used to annotate an element, and the first line may be displayed in the output from analysis. If the annotation in the note field is long it may be edited separately by double-clicking on the element number to open its note field in an edit pane (Figure 8).

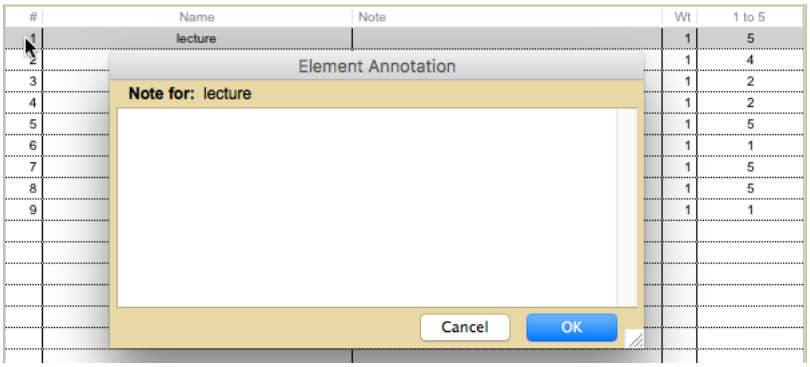



Figure 8: RepGrid ***Elements*** pane note field opened by double-clicking on number field

Element annotation is particularly useful if the relevant name is not sufficiently descriptive of the element or the reason for including it, for example, a person referenced by name might have a note ***best friend*** or ***manager***, and one referenced by role might have a note giving their actual name.

The weight value may be used to give some elements more influence than others in various analyses (§5.9). **If integer values are used the effect of weighting is the same as if the element data had been entered the number of times specified in the weight.**

The ***Weight*** checkbox determines whether the element weights are used in analyses, and the ***Select*** checkbox determines whether only selected elements are used in analyses.

### 3.2.2 *Editing element data*

Any field except the number in the first column and the values in the last column may be edited by clicking on it, which selects it for editing (Figure 9). When the name has been edited, pressing the ***return*** or ***enter*** key or clicking elsewhere deselects it for editing and centres the name again.

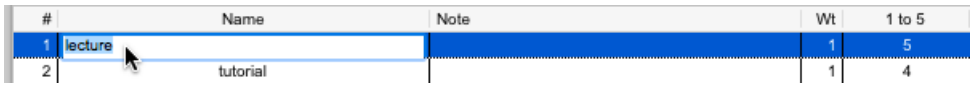



Figure 9: Clicking in a field selects it for editing

Pressing the ***tab*** key makes next visible field editable, stepping from name to note and weight if these are visible, and then to the name in the next row, creating a new row if at the end of the table. This supports rapid entry of whatever fields are being used for a particular grid. The ***Add*** button at the bottom right may also be used to add an additional row.

The popup menu of constructs at the top right appears when the ***Value*** box is checked allows a construct to be selected for which the rating values are provided in the rightmost column. The column header shows the range of possible values.

The ratings of the elements on the construct may be entered through a popup menu whose availability is indicated by the ***menu cursor*** that appears as one mouses over a rating (Figure 10).


Figure 10: Mouse cursor changes to indicate a popup menu is available to edit a rating

Clicking in the rating field activates the popup menu allowing one to select a rating (Figure 11).

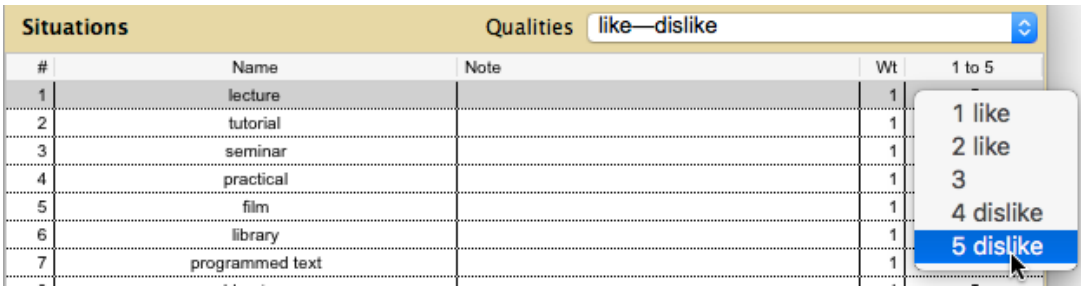



Figure 11: Popup menu editing a rating in the ***Elements*** pane

Ratings may also be edited as text in the same way as other fields by pressing the ***tab*** key to make the first rating editable, or by clicking in a rating and releasing the popup menu without selecting a new rating.

When editing the ratings as text, the ***tab*** key sequences down through the ratings, and has no effect if pressed in the last one.

### 3.2.3 *Sorting elements*

The list of elements may be sorted by clicking on the element number and dragging it to a different position, or by clicking on a column heading which sorts textual columns alphabetically and numeric ones numerically. Clicking on the heading of an already sorted column reverses the sort order. Figure 12 shows the elements sorted by ratings on the construct ***like—dislike*** after the ratings column header has been clicked. The elements will remain sorted by ratings if the construct selected is changed.

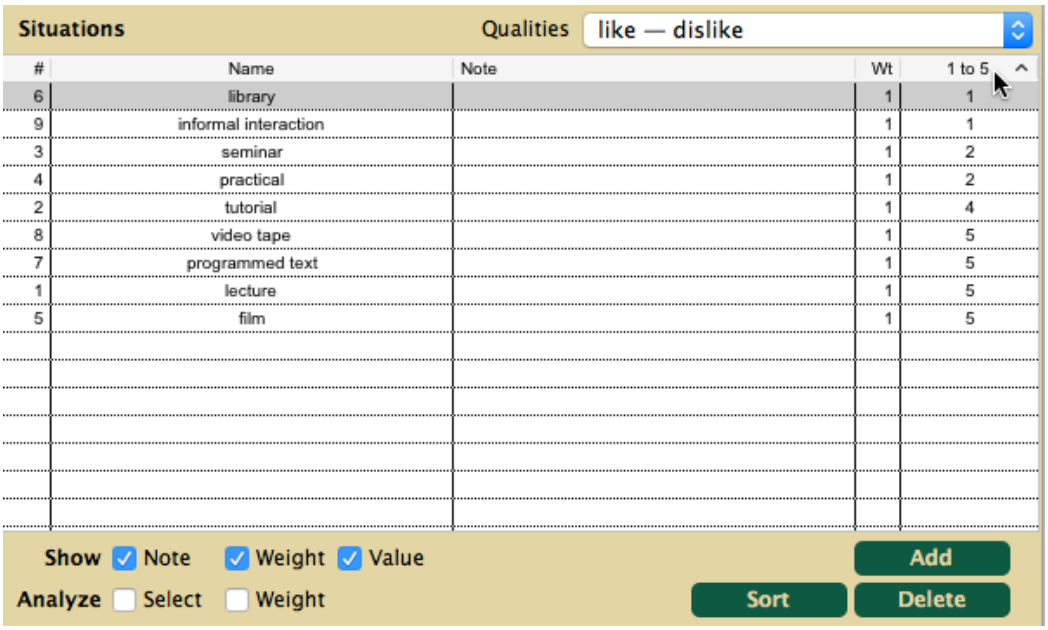



Figure 12: Sorting elements

Sorting in this way, or by dragging, does not renumber the elements and hence is only temporary. Clicking on the ***Sort*** button at the lower right makes it permanent and renumbers the elements accordingly.

## 3.3 Constructs pane

Clicking on the ***Constructs*** tab in the RepGrid window brings up the constructs pane (Figure 13). This lists the numerical order of the constructs in the grid in the column on the left, their left hand pole (LHP) names in the next column, and their right hand pole (RHP) names in the next column.

Editing constructs in this pane is similar to editing elements as described above. Rows may be selected for deletion with the ***delete*** or ***backspace*** key by clicking in the number field. They may be added by clicking the ***Add*** button or keying ***tab*** when the last (non-value) field of a construct is editable. The construct pole name fields are always shown, and the associated note, weight, value, level and output fields may also be shown dependent on which boxes are checked in the row under the data. Figure 14 shows the ***Constructs*** pane with all the fields activated and some constructs selected.

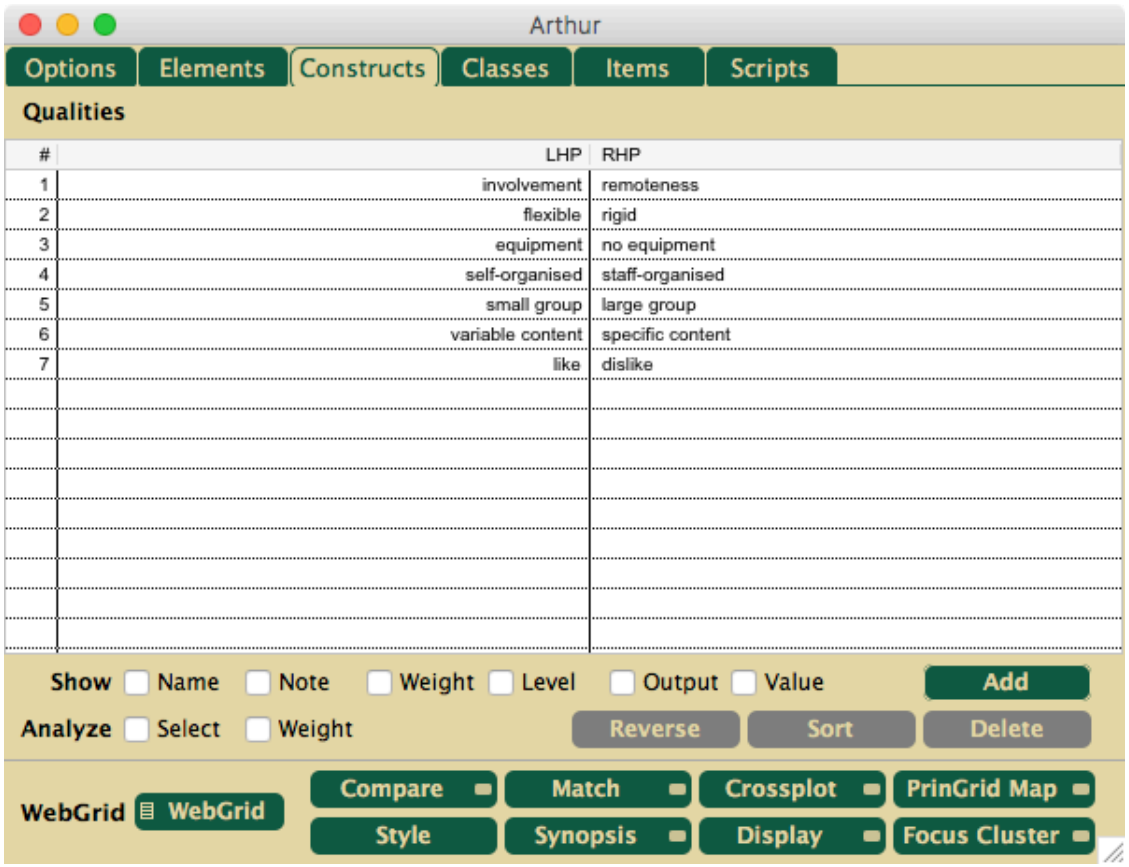



Figure 13: RepGrid ***Constructs*** pane showing construct numbers and poles

### *3.3.1 Construct names*

Construct ***names*** may be used a way of naming the range of convenience of the construct separately from that of its poles. When it is displayed in graphic or textual output, the construct name is treated as a noun and the pole name as an adjective qualifying that noun, e.g. ***golfer*** as construct name and ***good—poor*** as its a pole names will display as ***good golfer—poor golfer***. The name column is placed to the right of the pole columns in the Constructs pane to encourage this usage. In the literature constructs are often named through their pole names, e.g. ***flexible—rigid***, and RepGrid defaults to this convention if a name is not supplied.

Construct names are uncommon in normal grid data but are useful in applications where the pole names are insufficient to identify the construct, e.g. with "Q-sort" (Stephenson, 1953) data where the elements are statements, all the constructs are ***agree—disagree,*** and the name field is used to identify the person responsible for rating that construct.

Construct names can also be used to represent ***ordinal relations*** between constructs (Kelly, 1955, p.56) by entering a construct name that is the same as a pole name of a superordinate construct (§3.3.6).

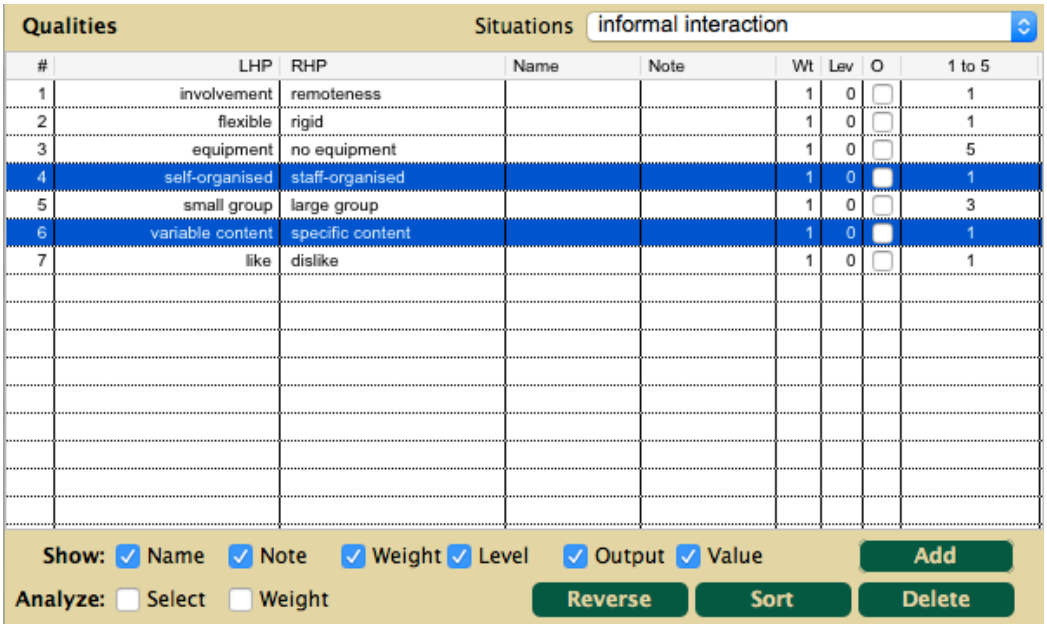



Figure 14: RepGrid ***Constructs*** pane with all fields showing

### *3.3.2 Construct annotations, weights and reversal*

Each construct can be annotated, and the first line of the annotation may be shown in analyses. Lengthier annotation can be edited as described for elements, by double-clicking on the construct number to open its note field in an edit pane.

The weight value enables some constructs to be given greater weight than others in various analyses (§5.9). If integer values are used the effect of weighting is the same as if the construct data had been entered the number of times specified in the weight.

The ***Weight*** checkbox determines whether the construct weights are used in analyses, and the ***Select*** checkbox determines whether only selected constructs are used in analyses.

The capabilities of specifying a construct as an ***output*** or giving it a ***level*** relative to other constructs are provided to support algorithms for modelling the conceptual structure represented in a grid, such as various forms of machine learning.

Constructs may be sorted by clicking the column headers or by clicking on the construct number and dragging it to a different position. Clicking on the ***Sort*** button renumbers the constructs to make the sort order permanent.

Clicking on the ***Reverse*** button reverses any selected constructs. That is the poles are interchanged and the ratings complemented such that there is no difference in meaning. Such reversal occurs automatically in analyses but it is sometimes appropriate to adjust it in the original grid, for example, to show all the preferred poles on the left or right.

### 3.3.3 *Editing the ratings of elements on constructs*

Ratings may be edited through the keyboard or popup menu as for elements, and the ***tab*** key behaviour is the same as for elements.

Double-clicking on a construct number opens up the note field of a construct for lengthier annotation as it does for an element (Figure 15). It also makes the construct's rating scale range available for editing in case it does not correspond to the default range given in the ***Status*** pane. RepGrid supports the editing and analysis of grids containing constructs with differing rating scales by rescaling all ratings to the same range (a floating point double in the range -1.0 to +1.0) before analysis. If a rating scale is changed after element ratings have been entered then the ratings are rescaled proportionately along the new scale (note that this rescaling may lose information).

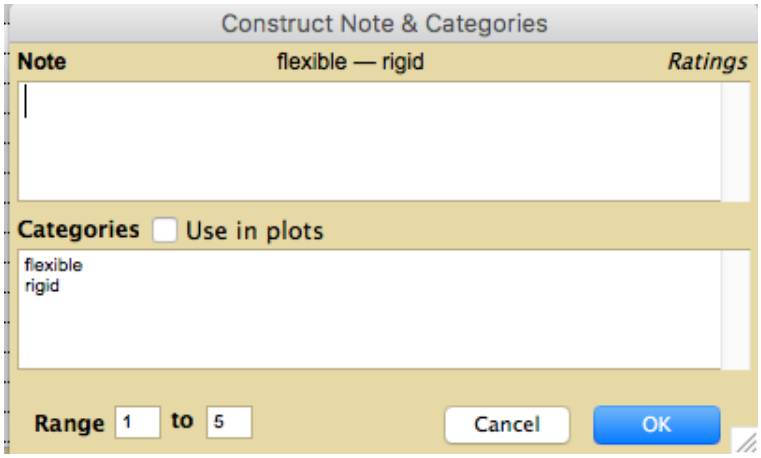



Figure 15: RepGrid ***Constructs*** pane note and categories field for a rating scale

### 3.3.4 *Assigning categories to rating scale ranges*

The points, or ranges, on a rating scale may be assigned textual labels termed ***categories***. The left and right hand pole names are treated as initial categories (Figure 15) and, if no further categories are allocated, will be assigned non-overlapping ranges from the lowest rating value to the mid-point of the scale, and from the mid-point to the highest rating value, respectively. The mid-point itself will be omitted to avoid overlap if there is an odd number of rating values. This is apparent in the popup menu shown on the right of Figure 11.

The ***Categories*** text field in the bottom half of the note field editor allows the pole names to be edited and additional categories to be allocated to rating scale values or ranges as shown in the in Figure 16 where a label has been specified for each value of the 5 point scale. These labels are used in the popup menus (Figure 17), and are also available to conceptual modelling algorithms.

More complex specifications of the relations between ratings and categories are possible by specifying an explicit numeric range for a category as one or two numbers separated by a space before the category label as shown in Figure 18. A single number specifies the rating associated with the category, and a pair the range of ratings.

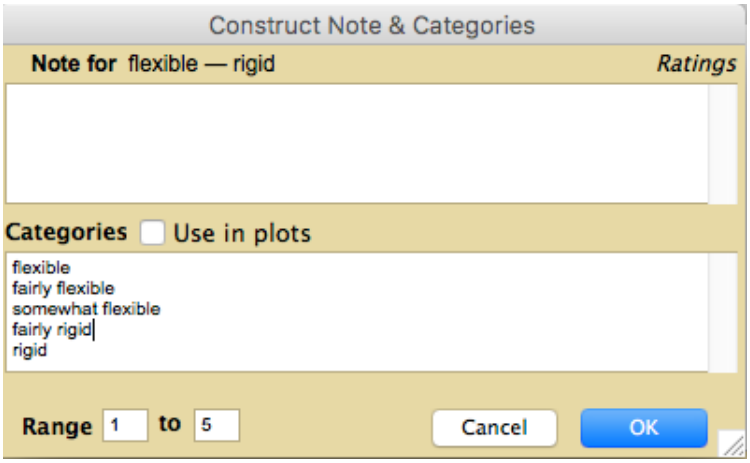



Figure 16: Specification of intermediate categories

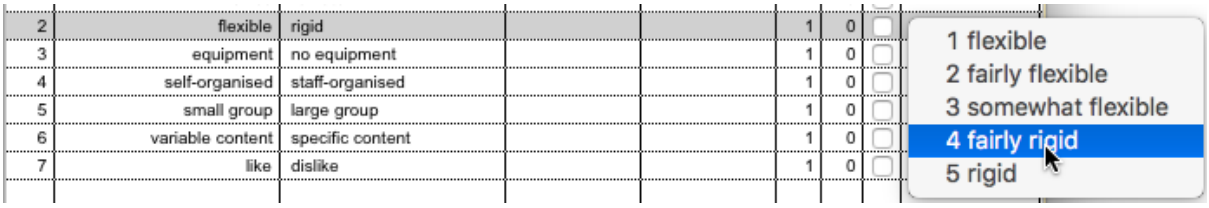



Figure 17: Popup menu showing specified values and categories

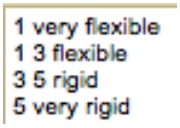



Figure 18: Specification of overlapping category ranges

The ranges for categories may overlap and, if two or more categories apply to the same rating, all of those that apply are shown separated by commas (Figure 19).

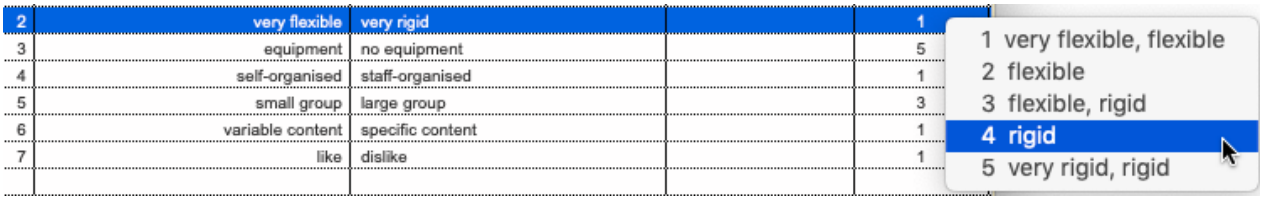



Figure 19: Popup menu showing overlapping categories

Bare labels and numerically specified labels may be freely mixed, and the RepGrid labelling algorithm will attempt to achieve what seems to be intended. Any problems with label specification can be made apparent by checking the popup menu it generates. The first and last label are used as the pole names in the plots from the various analyses.

The capability to specify labels indicating the ***meanings*** intended to be associated with intermediate rating values is particularly useful in communal studies where a grid is developed by a core group

reflecting expected community constructs and provided without ratings to members of the target community to provide their individual ratings. In such studies it may be useful to attempt to promote a common understanding of the use of intermediate values on the rating scales.

For example, the overlapping labelling of a scale illustrated in Figure 19 may be more appropriate for some purposes than one which allocates a different label for each scale point. It indicates that the stronger terms, ***very flexible*** and ***very rigid,*** apply only to the extreme of the scale but the more general term, ***flexible*** and ***rigid,*** encompasses a range of ratings, and this representation may be closer to common linguistic usage.

The ***Use in plots*** check box indicates that the category labels rather than the numeric rating values should be used in the output of analyses that show rating values, such as Synopsis, Display and Focus.

### *3.3.5 Categorical, numeric and integer construct rating scale types types*

As well as conventional rating scales, RepGrid supports the use of categorical, integer and numeric (floating point) data types in grids (§3.1). If all four construct types are checked in the ***Options*** pane then the ***Constructs*** construct ***Add*** button at the lower right changes to one with menu icon on the left that allows the type of construct to be added to be selected. Figures 20 and 21 illustrate this for a grid about house choices that uses multiple construct types.

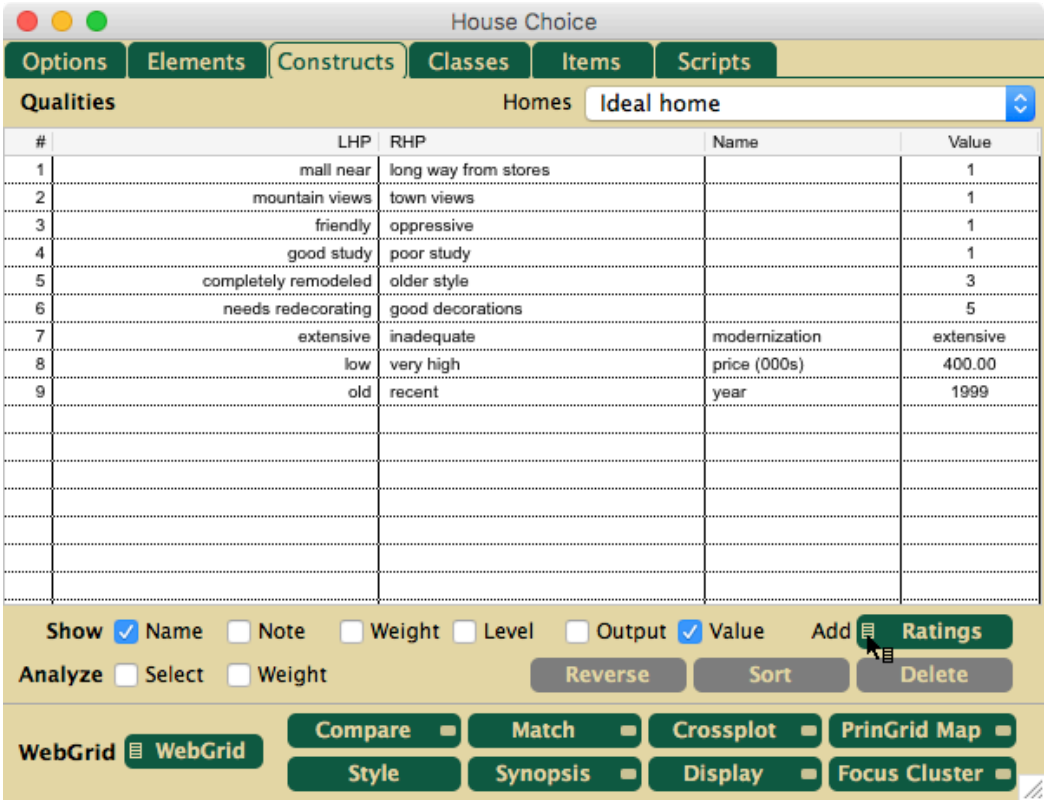



Figure 20: RepGrid ***Constructs*** pane offering multiple construct types

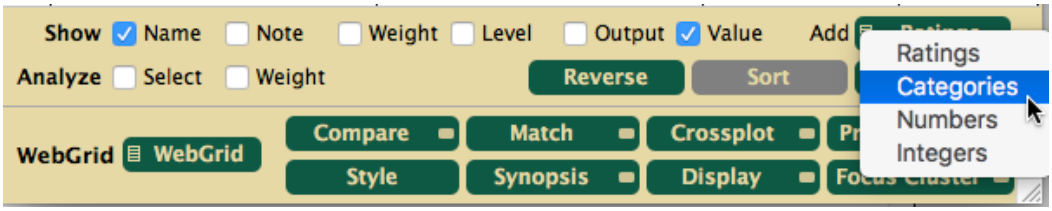



Figure 21: Popup menu for selecting a construct type

In this grid the first six constructs are rating scales and the last three are categories, integers and numbers, respectively. Double-clicking on the seventh construct opens up the note and categories fields for editing (Figure 22).

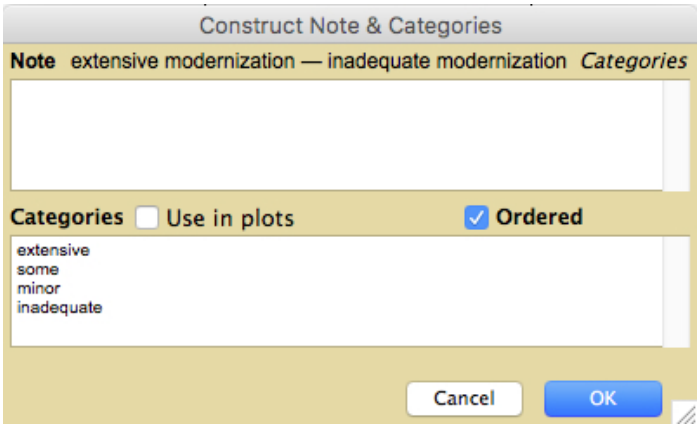



Figure 22: RepGrid ***Constructs*** pane note and categories fields for a categorical construct

The category names are arbitrary strings entered when the ***Category*** button is clicked to add a categorical construct. The number of points on the associated rating scale is from 1 to the number of categories entered and the ***Use in plots*** option is set by default. If it is unset then the categories will be represented by their associated rating values in the output from analyses. In addition to the bare categories that determine the rating scale, numerically specified categories may be entered as for the ***Ratings*** type.

The ratings of elements on categorical constructs can be entered and changed through the ***Value*** field, textually or using a popup menu, in the same way as for ratings (Figure 23).

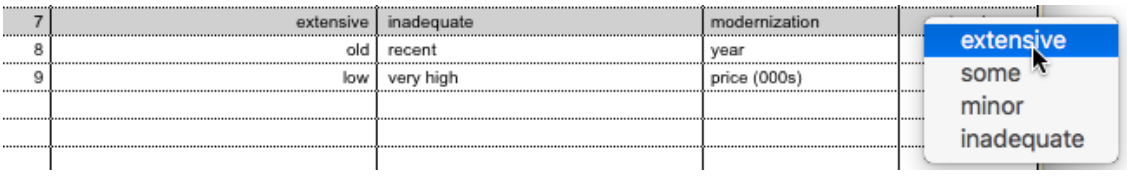



Figure 23: Popup menu showing specified categories

Categories are generally assumed to be ordered so that they may be treated as a conceptual dimension, but unordered categories may also be represented by unchecking the ***Ordered*** check box.

Some analyses can take this account, but distance-based analyses, such as Focus and PrinGrid, will still treat the categories as ordered.

The category labels can be edited after ratings have been entered, and additional categories can be added without existing ratings being affected. The order of the labels can also be changed without the ratings being affected. **However, if both the labels and the order need to be changed this should be done as two separate edits as, otherwise, the intended outcome is ambiguous.**

If a category label is deleted then any elements rated with that category will have their ratings set to be open.

The ***Construct Note and Categories*** fields opens up automatically when a new construct, other than ratings, is added in order to allow the category labels and/or the range of the numeric rating scales to be specified.

Double-clicking on the eighth construct, named ***Price (000s),*** show that it has numeric values in the range 200.00 to 600.00, and that categories have been assigned to ranges of prices (Figure 24). The precision required is inferred from the maximum number of decimal places specified in the ***Range*** fields.

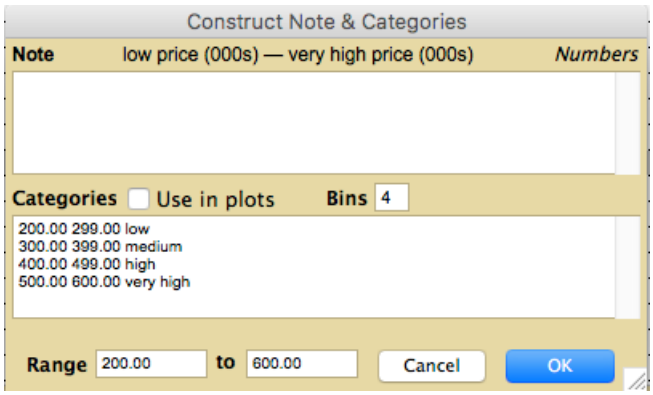



Figure 24: RepGrid ***Constructs*** pane note, range and categories fields for a numeric construct

The ***Use in plots*** option is unchecked by default but can be checked to indicate that categories rather than numeric values should be used in analyses. The ***Bins*** field allows the number of bins to be set in the Synopsis analysis histograms if categories are not being used in plots.

The ratings of elements on numeric constructs can be entered and changed through the ***Value*** field, textually or by using a popup menu, in the same way as for ratings (Figure 25).

The first and last category will be used as the pole names and it is normal for the first to have the same lower bound as the overall range and the last to have the same upper bound as the range.

Double-clicking on the ninth construct, named ***Year,*** show that it has integer values in the range 1960 to 1999, and that categories have been assigned to ranges of years (Figure 26). Subranges have been allocated category labels as shown and the other features are the same as those for the more general ***Numbers*** type, e.g. the popup rating menu of Figure 27.

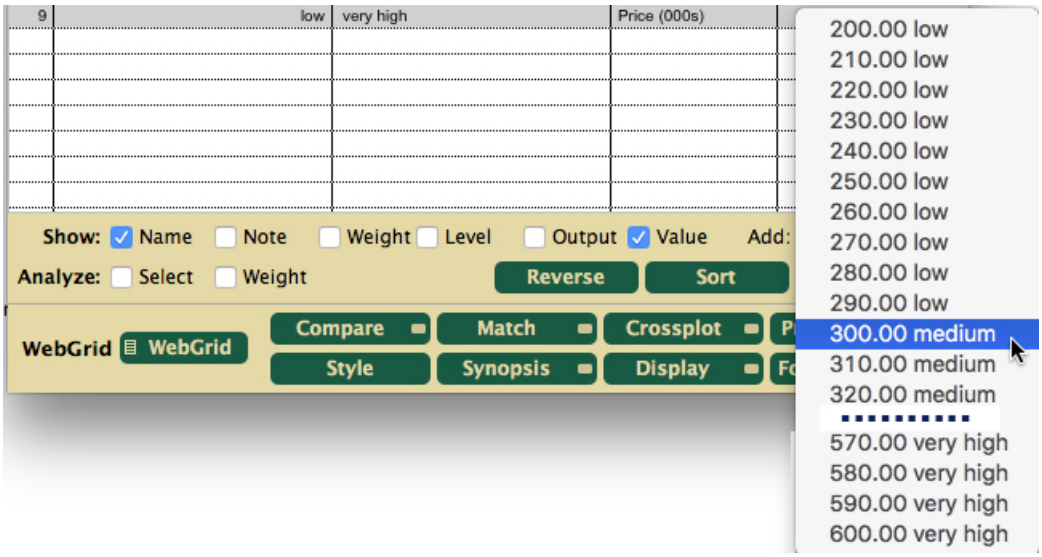



Figure 25: Popup menu (truncated) showing some possible values and categories

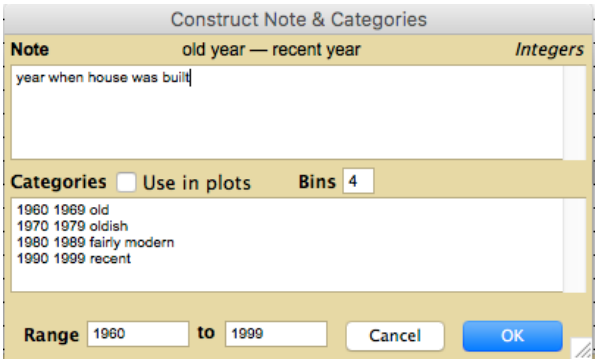



Figure 26: RepGrid ***Constructs*** pane note, range and categories fields for an integer construct

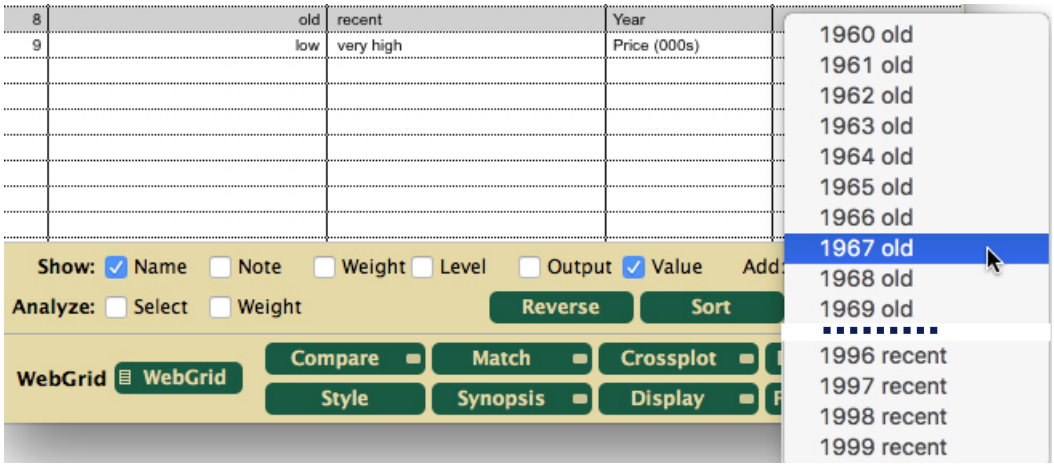



Figure 27: Popup menu (truncated) showing possible values and categories

### 3.3.6 *Representing ordinal relations between constructs*

The availability of ***metavalues*** (§3.1.6) and the capabilities to specify construct ***names*** (§3.3.1) and ***categories*** (§3.3.4) combine to enable Kelly's (1955, p.56) ***ordinal relations*** between constructs to be represented in a grid. Entering a construct name that is the same as a pole name or category of a superordinate construct is recognized as specifying that the first construct is subordinate to the second. The metavalue ***~ Inapplicable*** may be used to rate a construct as ***irrelevant*** if it is a subordinate to the opposite pole of one applicable to the element. When elements are ***split*** into two differently named elements to avoid an inconsistency arising from ordination, the previous name may be represented in a categorical construct so that the linkage between the new elements is still represented in the grid.

Figure 28 illustrates how this may be done for the example discussed in §3.1.6 of the construct ***poor beaches—good beaches*** being subordinate to the ***waterside*** pole of the construct ***inland—waterside***. At the top are the element names and construct poles and names entered in RepGrid; in the center the ratings of three exemplary elements; and bottom a conceptual net formally derived from the grid. The elements are vacation locations: Arrimoor, construed as an inland town where the ***poor beaches—good beaches*** construct is inapplicable; Stonyside, construed as a seaside town with poor beaches; and Sandyside, construed as a seaside town with good beaches.

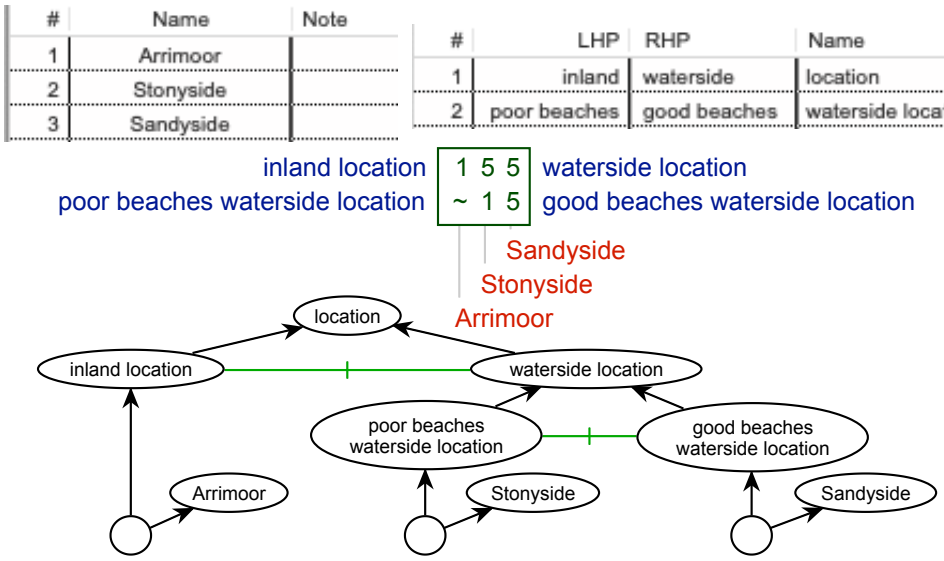



Figure 28: Representing ordinal relations between constructs in a grid

The net at the bottom may be derived from the grid (§3.4.5) and ***vice versa***—conceptual nets and conceptual grids provide alternative, but equivalent, representations of conceptual structures (Gaines and Shaw, 2012). The concepts represented by the poles ***poor beaches*** and ***good beaches*** are ***false*** for experiences of Arrimoor because they are ***irrelevant***, whereas for those of Stonyside and Sandyside they are relevantly false and true, respectively. The figure illustrates Kelly's (1955) notions that: a grid represents a conceptual "network" (p.304); there is an essential difference between a property being "false" and being "irrelevant" (p.60).

The concepts corresponding to the elements are the anonymous nodes at the bottom of the net with the element names shown underneath them. Each of these nodes represents a constellation of ***experiences***, direct and mediated, that have been attributed to an entity identified by the element name. These experiences could each be represented by further anonymous nodes subordinate to the named one, and further nodes subordinate to these. The bottom nodes could be understood as representing ***individual*** events, or states of affairs, that is, as events which are construed as the ***states*** of individual entities (Gaines, 2015). All of this is subsumed into the notion of an ***element*** in a grid.

The element names are, from a logical perspective, arbitrary labels providing a unique identifier for what is construed as an entity, but, for purposes of communication, are usually terms that are generally used to indicate the constellation of experiences associated with the hypothesized entity or descriptive terms sufficient to identify it in the context of their usage. Such considerations address Kelly's (1955) concerns deriving from Korzybski (1951) and Whitehead (1929) that one should recognize that the elements are not concrete entities given by ***reality*** but are themselves constructed by carving events from experiences and construing certain collections of events to attributable to some entity. The interpretation of the ***entity*** construct from a realist perspective might be that we evolved in a corner of the universe that naturally partitions into entities in a way relevant to our survival, and from a constructivist perspective that, perhaps as a consequence, we tend to structure our social and built environments in the same way.

The named node, its subordinates and superordinates may be seen as constituting what has come to be termed the ***mental file*** (Récanati, 2013) that a person creates to group their experiences of an ***object*** in their environment. The superordinate nodes shown are some of those in the mental file of the person from whom the grid has been elicited that she or he deems relevant to the topic of ***selecting a vacation location***.

Ordinal relations between constructs are often elicited by ***laddering techniques*** (Reynolds and Gutman, 1988; Corbridge et al., 1994; Korenini, 2014). Their explicit representation in the grid supports elicitation techniques that provide feedback to users on the consistency between the ordinal relations that have been elicited by laddering and the rating values they have specified (Korenini, 2014). For example, in figure 28, rating Arrimoor as having poor or good beaches would be inconsistent with it being rated as inland. Inconsistency may indicate error in the data entered, that a particular element is an anomaly that does not conform to normal expectations or that it is heterogeneous with different parts, or aspects, having different characteristics.

If the source of the inconsistency is attributed to the element being somewhat heterogenous and having multiple significant states then a common technique is to ***split*** it into two more elements representing these states, for example ***Mary as a doctor*** and ***Mary as a mother***. Kelly's (1955, p.50) ***construction corollary*** states that ***"a person anticipates events by construing their replications"***, but the decision to treat one event as a replication of another is a significant abstraction that may itself be problematic.

Korzybski (1951) saw such abstraction of events as a powerful technique essential to the development of human knowledge and civilization, but also cited excessive abstraction as major source of psychological problems. He promoted his ***chain-indexing*** technique as a therapy for such problems, in which client's were taught to separate events that they attributed to the ***same*** entity by indexing

the name under which they subsumed the events. Problems *"may become trivial or nonexistent if we become conscious of the identifications involved"* (Korzybski, 1951, p.173).

For example, the inconsistency of Arrimoor being rated as having ***poor beaches*** when it has also been rated as ***inland*** might be resolved if it is noted that the town is predominantly remote from water but does have a small district far from the main town that is by the sea. Entering the different locations as two separate elements with notes indicating that one is ***Arrimoor town*** and the other ***Arrimoor seaside*** allows the conflict to be resolved (Figure 29). This shows in the grid and net the commonality of the two locations as being in Arrimoor but distinguishes them in terms of the ***inland—waterside*** construct.

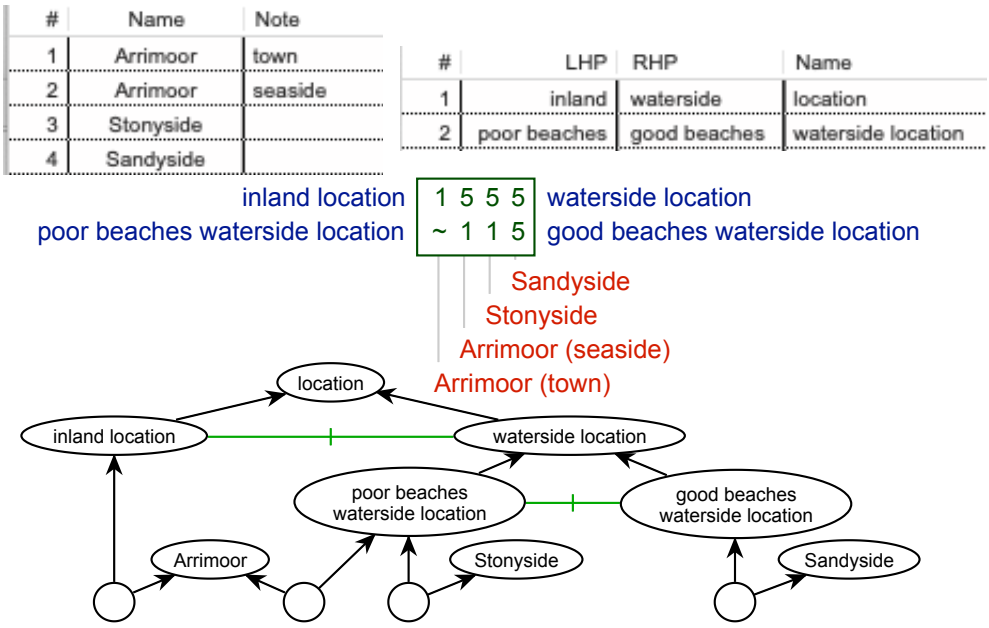



Figure 29: Splitting an element to resolve inconsistency

This example illustrates the techniques whereby structures that represent human knowledge in a way that people find understandable and adequate must be further refined to enable computational reasoning to emulate human reasoning. This process has come to be termed ***knowledge engineering***: *"The vocabulary initially used by the expert to talk about the domain with a novice is often inadequate for problem-solving; thus the knowledge engineer and expert must work together to extend and refine it."* (Hayes-Roth et al., 1983).

## 3.4 Classes—intersects and anticipation

Kelly (1955, p.121) explicates the basis of anticipation as the recognition that an event can be construed as falling at the intersect of a set of properties: *"What one predicts is not a fully fleshed-out event, but simply the common intersect of a certain set of properties. If an event comes along in which all the properties intersect in the prescribed way, one identifies it as the event he expected."* A ***class*** in RepGrid pro-

vides the means to specify an intersect of the properties of elements as encoded in a grid, notably the basic constructs, categories and already defined classes.

This specification may be seen as constituting the ***meaning*** of the class, and can be interpreted in a variety of ways. For example, classes may be treated as ***ideal elements*** or as ***compound constructs*** dependent on their roles in analysis and application (§3.4.7). In ***case-based reasoning*** (Kolodner, 1993) the ideal elements are regarded as prototypical ***cases***, and in ***rule-based reasoning*** (Buchanan and Shortliffe, 1984) the compound constructs are regarded as the premises of ***rules***.

The support of metavalues (§3.1.6) already provides the means to specify a basic intersect as an ***ideal element*** having ratings on the constructs providing the properties of the intersection, and the metavalue * ***Any*** on those that are not relevant to it—where ***ideal*** is understood in the sense of ***idealization*** rather than ***perfection***. An ideal element, such as ***ideal house*** may well represent a desired anticipation, but it may equally well be an intersect representing an undesired anticipation, such as a rugby player knowing that the intersect of handling the ball and knocking or throwing it forward is a foul play to be avoided. Classes extend the basic specification of an intersect in terms of constructs and ratings to also include specification in terms of relations with other classes, corresponding to one possible anticipation being subordinate to, preferred to, or an exception to, another.

From a personal construct psychology perspective, intersects, ideal elements, cases and rules are all subsumed by, and exemplify, Kelly's (1955, p.8-9) notion of ***templets*** as patterns that are fitted to events. The basic ***constructs*** of a conventional grid represent the dimensions of some part of the elicitee's psychological space, the compound constructs that we have termed ***classes*** represent intersects in that space in terms of the basic constructs and the other classes already defined. Classes may be regarded as more complex specifications of ***ideal elements*** and also treated as ***dichotomous constructs*** whose poles specify that the ideal element can ***fit*** (match closely), or cannot ***fit*** (match poorly), another element.

Classes were originally introduced in RepGrid to support the development of ***knowledge-based*** or ***expert*** systems where Kelly's (1955, p.46) fundamental insight, that construct systems evolved to support the ***anticipation*** of events, is operationalized by eliciting grids from domain experts and using them to emulate the anticipatory processes underlying human expertise (Boose and Gaines, 1988; Shaw and Gaines, 2005).

### *3.4.1 Classes Pane*

Clicking on the ***Classes*** tab brings up the classes pane (Figure 30). This lists the numerical order of the classes specified for the grid in the column on the left, followed by the number of elements that could fall under a class, the class names, and their meanings in terms of the constructs, categories and other classes that have already been specified.

Editing classes in this pane is similar to editing elements or constructs as described above. Rows may be selected and sorted, the class number and name are always shown and the count, meaning and note fields may be shown or hidden. The ***Add***, ***Sort*** and ***Delete*** buttons act as before. The remaining controls and the editing capabilities activated by double-clicking on a class number or description are specific to classes and are detailed below.

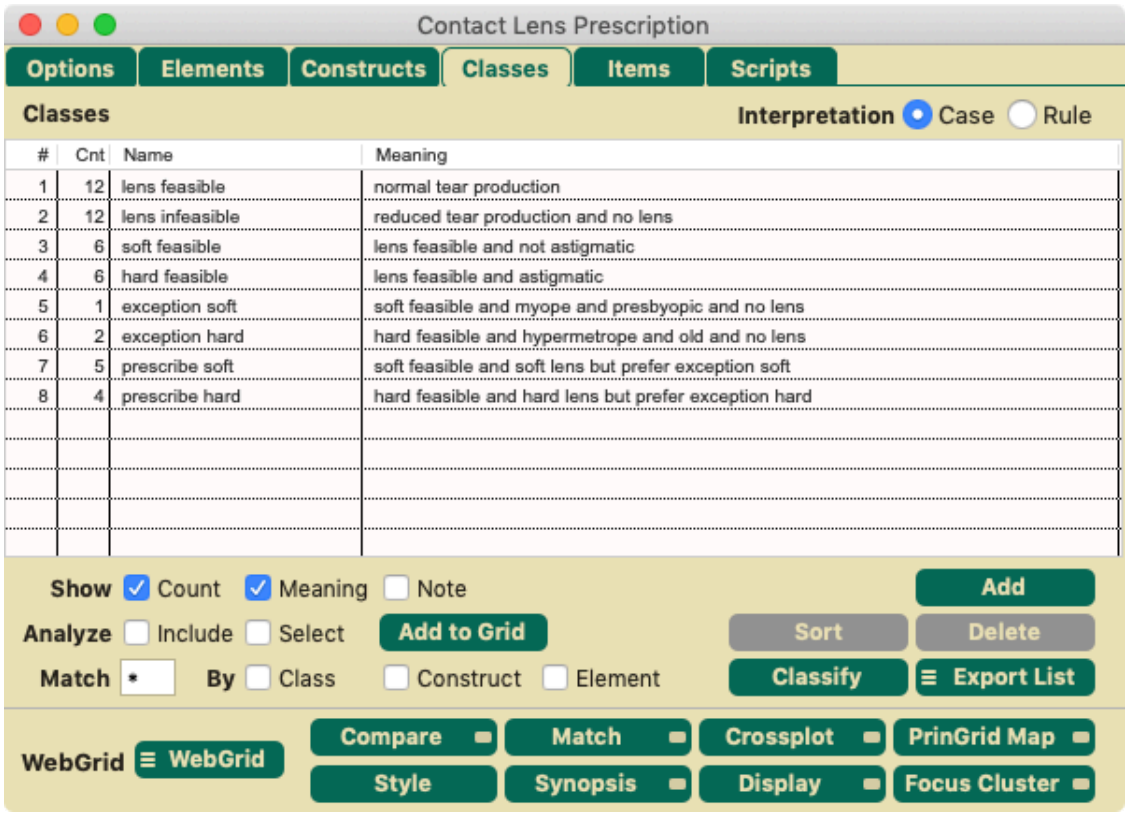



Figure 30: RepGrid *Classes* pane showing class numbers, element counts, names and descriptions

The grid shown represents a well-known expert system dataset devised by Cendrowska (1987) to demonstrate issues with machine learning algorithms in the 1980s, and has been widely studied because the problem is simple enough to be understood but complex enough to exhibit significant aspects of computational knowledge acquisition, representation and inference. The problem is presented as one of contact lens prescription where a hard lens is suitable for an astigmatic client and a soft one otherwise, but there are exceptional conditions making a contact lens unsuitable. One of these is reduced tear production but Cendrowska posits that this should only be tested if the other exceptions do not apply because the test is invasive and expensive.

The dataset is represented in the grid by the 5 constructs and 24 elements that Cendrowska specifies which encompass all possible prescription situations. The elements have been named to include the correct prescription, and the exception cases identified by an asterisk, in order to make it easier to assess the outcome of classification. The classes shown represent a complete solution in a form intended to emulate the expert's decision processes: the initial considerations based on astigmatism; the possible exceptions; and the final prescriptions.

The two prescription classes at the end are the recommendations for action: ***prescribe soft*** and ***prescribe hard***. The implicit recommendation for no prescription, ***prescribe none***, applies to the residual elements not covered by these classes. If it was desired to show it explicitly the class specification would be ***no lens but prefer prescribe soft or prescribe hard***.

### 3.4.2 *Anticipation: classes as intersects, templets, cases, rules*

The ***Case*** and ***Rule*** radio buttons at the top right specify whether the classes should be interpreted as cases or rules. The actual class specifications do not change but the way they are described is different. Figure 31 show the classes pane above when the interpretation is changed to rules. The entailments shown for the lens construct arise because that construct has been flagged as an ***Output*** and hence treated as the conclusion of a rule. The need to be explicit about the separation between premise and conclusions is peculiar to rule interpretations and not required for case-based reasoning.

**Classes** **Interpretation** ○ Case ◉ Rule

| # | Cnt | Name | Meaning |
|---|---|---|---|
| 1 | 12 | lens feasible | normal tear production |
| 2 | 12 | lens infeasible | reduced tear production and entails no lens |
| 3 | 6 | soft feasible | lens feasible and not astigmatic |
| 4 | 6 | hard feasible | lens feasible and astigmatic |
| 5 | 1 | exception soft | soft feasible and myope and presbyopic and entails no lens |
| 6 | 2 | exception hard | hard feasible and hypermetrope and old and entails no lens |
| 7 | 5 | prescribe soft | soft feasible but not exception soft and entails soft lens |
| 8 | 4 | prescribe hard | hard feasible but not exception hard and entails hard lens |

Figure 31: Classes pane with ***Rule*** interpretation

### 3.4.3 *Editing class meanings*

To enter a new class click on the ***Add*** button and type in the new class name (Figure 32).

**Classes** **Interpretation** ◉ Case ○ Rule

| # | Cnt | Name | Meaning |
|---|---|---|---|
| 1 | 12 | lens feasible | normal tear production |
| 2 | 12 | lens infeasible | reduced tear production and no lens |
| 3 | 6 | soft feasible | lens feasible and not astigmatic |
| 4 | 6 | hard feasible | lens feasible and astigmatic |
| 5 | 1 | exception soft | soft feasible and myope and presbyopic and no lens |
| 6 | 24 | exception hard | |

Figure 32: Entering the class ***exception hard***

To specify the meaning of the new class, or edit that of an existing one, double-click on the class number or meaning to bring up the ***Edit Class dialog*** (Figure 33). The first row shows the class being edited; the second a popup menu of classes and associated editing buttons for removing the class shown in the menu from the meaning of the class being edited, or adding it in positive or negative form; the third the constructs; and the fourth the categories associated with the construct above.

Under these areas is a non-editable text area showing the meaning of the class as it is defined, an editable text area where the class may be annotated, and buttons to cancel the class dialog without changing the class meaning, or to update the class meaning to be that entered or edited.

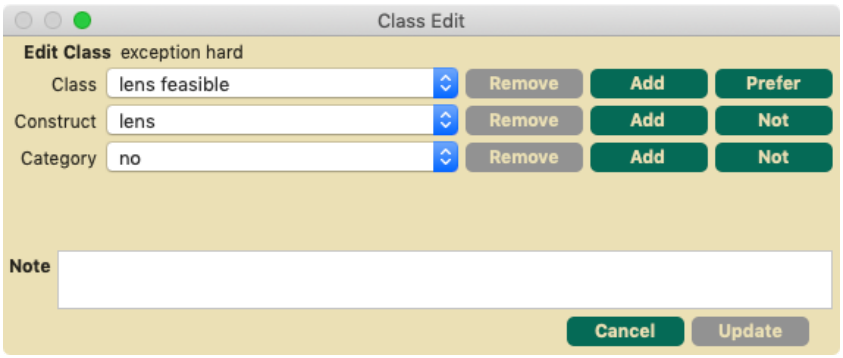



Figure 33: Editing the meaning of the class ***exception hard***

The menu of classes at the top is automatically computed to be those without any dependencies on the class being edited so that entering a dependency cannot lead to a circular specification. This computation is independent of the order of the classes so that they can be reordered as the user wishes. Selecting the class ***hard feasible*** and clicking on ***Add*** in the ***Class row*** adds that class to the specification of the meaning of ***exception hard*** (Figure 34), that is the class ***exception hard*** is specified as a ***subclass*** of ***hard feasible*** and ***inherits*** the meaning of that class (***lens feasible and astigmatic***).

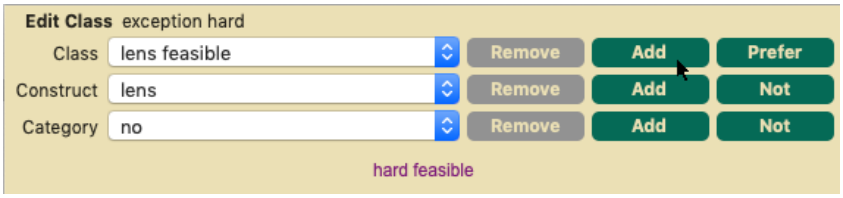



Figure 34: Entering a dependency on the class ***hard feasible***

Similarly, selecting the construct ***young—old***, its category ***old*** and clicking ***Add*** in the ***Category*** row makes ***old*** part of the meaning (Figure 35). This category is defined in the construct to be either ***presbyopic*** or ***pre-presbyopic***.

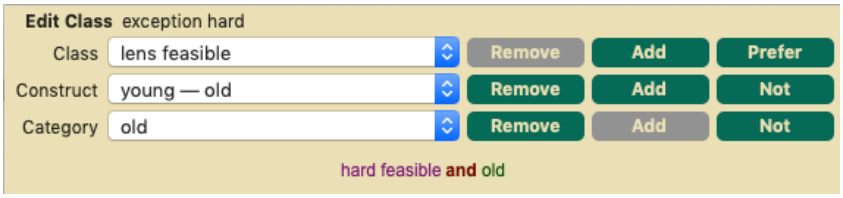



Figure 35: Entering construct category as part of a description

If ***old*** had not been specified as category for the construct, one could achieve the same effect in the class description by adding both ***presbyopic*** and ***presbyopic*** which be interpreted as the disjunction of the two categories, ***pre-presbyopic or presbyopic*** (Figure 36).

When the meaning of the class has been completely specified, clicking on the ***Update*** button sets the meaning of the class and closes the dialog.

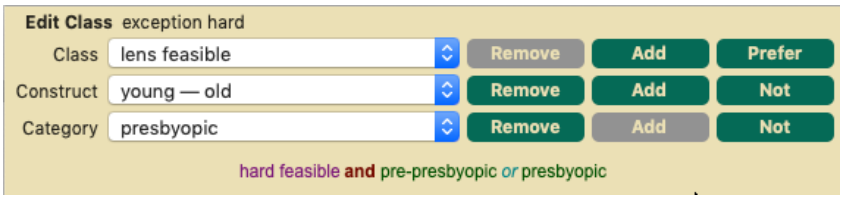



Figure 36: Entering a disjunction of construct categories as part of a description

The *Not* buttons add the negation of the class, construct or category, but their use is not recommended for constructs and categories as positive specifications are more natural and easier to understand. However negative links to classes are useful because they allow preferences (for cases) or exceptions (to the premises of rules) to be specified. The name of the button changes between *Prefer* for cases and *Not* for rules to reflect the differing interpretations of negative links between classes.

The removal of a construct is a quick way to remove the construct and associated categories. The capability to add a bare construct or its negation to a specification allows classes to be specified in terms of the relevance or irrelevance of certain constructs. For example, in the ***philosophy of art*** it has proved difficult to establish an agreed definition of what it is to be an ***art object*** and there are many competing attempts (Davies, 1991). However, consensus might be achieved about the collection of dimensions along which any art object might be construed (Gaut, 2000), and the relevance of these dimensions to the discussion of any entity might be sufficient to establish that the discussion is about an ***art object*** (Gaines and Shaw, 2012; Gaines, 2015). The positioning of an art object on certain dimensions would be significant for establishing its category as ***primitive art***, ***representational art***, ***African art***, and so on, but it is the expectation that certain ways of construing will be accepted as relevant that establishes its status as an art object.

### 3.4.4 *Using classes for classification*

The ***Classify*** button outputs classifications based on the classes and constructs. The ***Match*** text allows the metavalues to be specified which will match anything, usually just * but sometimes it is convenient to allow ? also. The three check boxes after ***By*** determine whether the classification is by class, by element or by construct. Figure 37 shows elements classified by class for the contact lens grid.

**Classification by Classes**

1: **lens feasible:** soft case 1, hard case 1, soft case 2, hard case 2, soft case 3, hard case 3, soft case 4, none case 9*, none case 11*, hard case 4, soft case 5, none case 15*
2: **lens infeasible:** none case 1, none case 2, none case 3, none case 4, none case 5, none case 6, none case 7, none case 8, none case 10, none case 12, none case 13, none case 14
3: **soft feasible:** soft case 1, soft case 2, soft case 3, soft case 4, none case 11*, soft case 5
4: **hard feasible:** hard case 1, hard case 2, hard case 3, none case 9*, hard case 4, none case 15*
5: **exception soft:** none case 11*
6: **exception hard:** none case 9*, none case 15*
7: **prescribe soft:** soft case 1, soft case 2, soft case 3, soft case 4, soft case 5
8: **prescribe hard:** hard case 1, hard case 2, hard case 3, hard case 4

Figure 37: Classification of elements by classes

The ***Select*** checkboxes for the elements, constructs and classes determine which items are included in the classification, for example, Figure 38 shows the elements classified by the last two classes.

### Classification by Classes

7: **prescribe soft:** soft case 1, soft case 2, soft case 3, soft case 4, soft case 5
8: **prescribe hard:** hard case 1, hard case 2, hard case 3, hard case 4

Figure 38: Classification of elements by selected classes

### 3.4.5 *Exporting classes as descriptions, logical expressions and conceptual nets*

The ***Export List*** button at the bottom right of the Classes pane has a menu symbol at the left providing options to export the classes as a textual list, logical expressions, or conceptual nets (Figure 39). In each case, the ***Case*** and ***Rule*** radio buttons at the top right specify whether the classes should be interpreted as cases or rules.

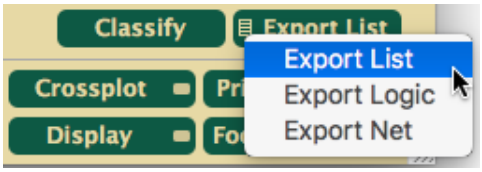



Figure 39: Class export options

Figures 40 shows the class specification exported as a list of cases or of rules.

### Cendrowska Descriptions of Classes as Cases

1: **lens feasible *means*** contact lens client **and** normal tear production
2: **lens infeasible *means*** contact lens client **and** reduced tear production **and** no lens
3: **soft feasible *means*** lens feasible **and** not astigmatic
4: **hard feasible *means*** lens feasible **and** astigmatic
5: **exception soft *means*** soft feasible **and** myope **and** presbyopic **and** no lens
6: **exception hard *means*** hard feasible **and** hypermetrope **and** old **and** no lens
7: **prescribe soft *means*** soft feasible **and** soft lens ***but prefer*** exception soft
8: **prescribe hard *means*** hard feasible **and** hard lens ***but prefer*** exception hard

### Cendrowska Descriptions of Classes as Rules

1: **lens feasible *means*** contact lens client **and** normal tear production
2: **lens infeasible *means*** contact lens client **and** reduced tear production ***and entails*** no lens
3: **soft feasible *means*** lens feasible **and** not astigmatic
4: **hard feasible *means*** lens feasible **and** astigmatic
5: **exception soft *means*** soft feasible **and** myope **and** presbyopic ***and entails*** no lens
6: **exception hard *means*** hard feasible **and** hypermetrope **and** old ***and entails*** no lens
7: **prescribe soft *means*** soft feasible ***but not*** exception soft ***and entails*** soft lens
8: **prescribe hard *means*** hard feasible ***but not*** exception hard ***and entails*** hard lens

Figure 40: Classes listed as cases or rules

Figure 41 shows the class and construct specifications exported as a conceptual net supporting case-based reasoning. A link to the data in the grid is automatically included so the elements may be used to test that the correct anticipations are made. The script *CNet* may be run to provide an inference engine for paraconsistent monadic first-order logic (FOL) and interpret the visual language representing the class and construct specifications as assertions in FOL.

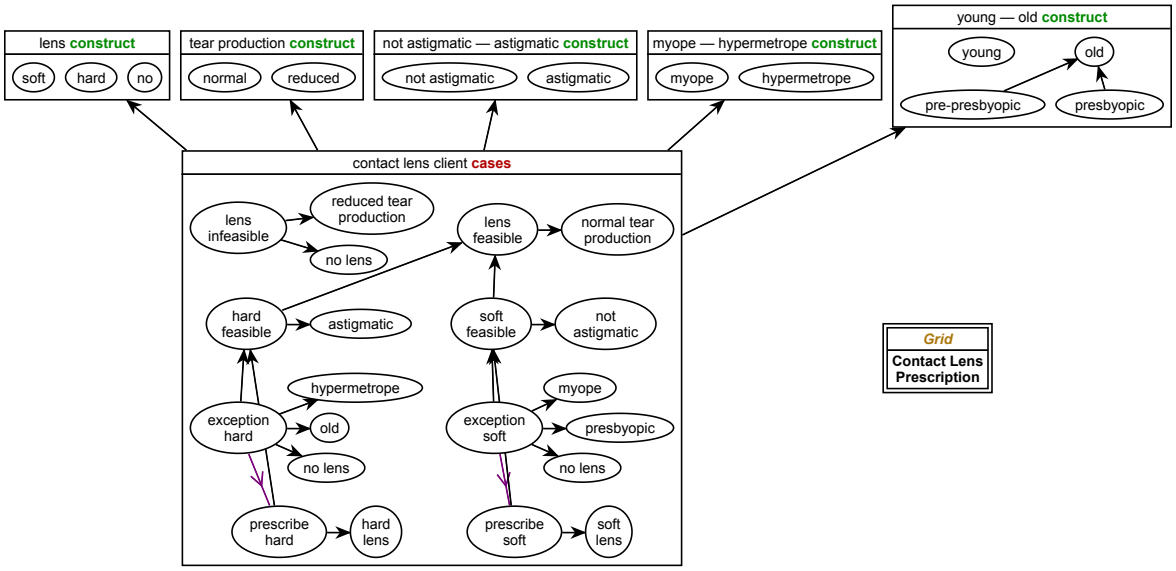



Figure 41: Classes exported as case-based reasoning net

It is difficult to design an automatic layout algorithm that can emulate as perspicuous an arrangement of the conceptual net as can a person, and the initial layout provided by RepGrid can usually be improved manually. Figure 42 shows the net of Figure 41 after manual adjustment to make it more comprehensible and where CNet has been run on a particular case.

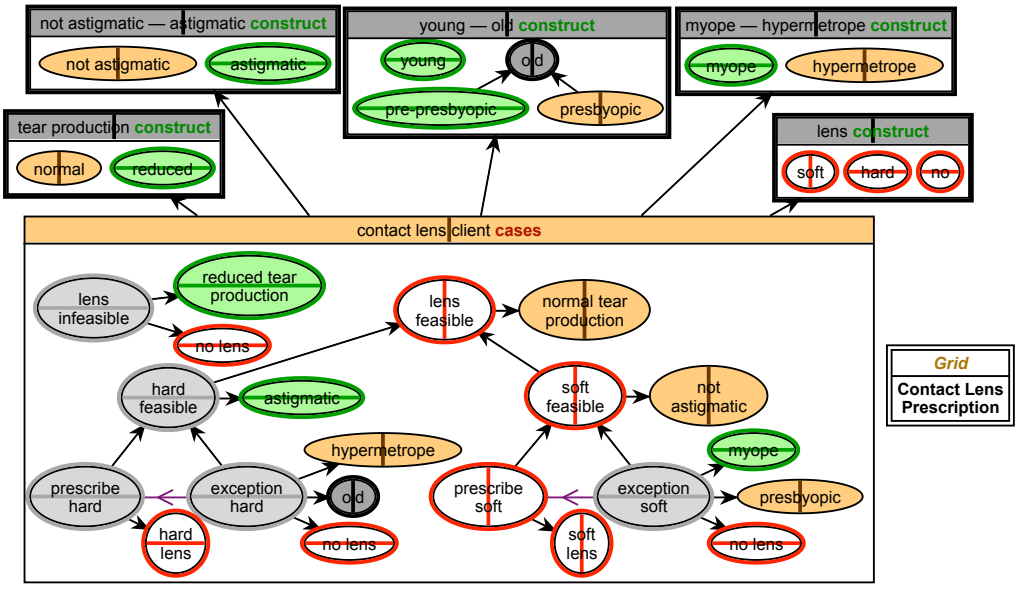



Figure 42: Case-based reasoning net with improved layout—inferences shown

The vertical lines in concepts indicate they are true, horizontal that they are false. The orange background indicates an assertion, green an inference through opposition, grey an inference through irrelevance, and red lines indicate an inference through logical necessity. The ***CNet Manual*** provides detailed information and further examples.

Figure 43 shows the same class and construct specifications exported as a conceptual net supporting rule-based reasoning. The constructs and grid are not shown as they are the same as Figure 42. CNet may again be used to infer anticipations but based on logical definitions rather than abductive case/prototype-based reasoning that may better model human inference.

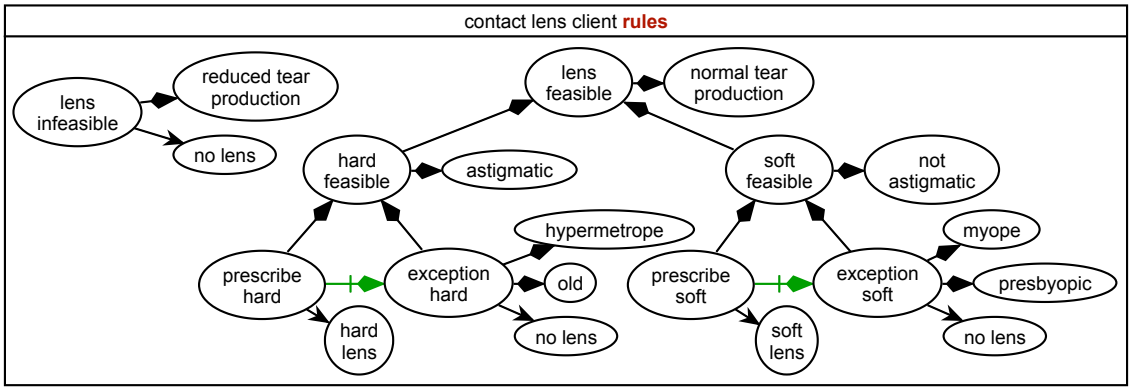



Figure 43: Classes exported as rule-based reasoning net

### *3.4.6 Using the Classes pane analyses with grids where classes have not been specified*

The ***Classify*** capability of the Classes pane is useful as a way of summarizing a grid even when no classes are defined. Figure 44 shows the classifications by constructs of the grid about learning situations used initially to illustrate editing a grid—the elements falling under each construct pole.

Figure 45 shows the classification by elements of the same grid—the construct pole under which each element falls.

Figure 46 shows the constructs and elements in the *learning situations* grid exported as a conceptual net.

Each element is represented as a state of affairs representing an experience of a learning situation that is expressed in the grid, with the element name as an indicial property indicating the type of the experience.

Running CNet, setting the pole of a construct true, and running abductive inference over the cases allows relations between the constructs to be discovered, for example that ***like*** always implies ***involvement*** and that ***dislike*** always implies ***specific content***. These asymmetric relations sometimes, but not always, correspond to the symmetric matches of other analyses, but sometimes provide different insights into the conceptual model implicit in the grid.

### Classification by Constructs

1.1: **involvement:** seminar, practical, library, programmed text, informal interaction
1.2: **remoteness:** lecture, film
2.1: **flexible:** practical, library, video tape, informal interaction
2.2: **rigid:** lecture, tutorial, film, programmed text
3.1: **equipment:** practical, film, programmed text, video tape
3.2: **no equipment:** lecture, tutorial, seminar, informal interaction
4.1: **self-organised:** library, programmed text, video tape, informal interaction
4.2: **staff-organised:** lecture, tutorial, seminar, film
5.1: **small group:** tutorial, library, programmed text, video tape
5.2: **large group:** lecture, seminar, practical, film
6.1: **variable content:** seminar, library, informal interaction
6.2: **specific content:** lecture, tutorial, film, programmed text, video tape
7.1: **like:** seminar, practical, library, informal interaction
7.2: **dislike:** lecture, tutorial, film, programmed text, video tape

Figure 44: Classification of elements by constructs

### Classification by Elements

1: **lecture:** remoteness, rigid, no equipment, staff-organised, large group, specific content, dislike
2: **tutorial:** rigid, no equipment, staff-organised, small group, specific content, dislike
3: **seminar:** involvement, no equipment, staff-organised, large group, variable content, like
4: **practical:** involvement, flexible, equipment, large group, like
5: **film:** remoteness, rigid, equipment, staff-organised, large group, specific content, dislike
6: **library:** involvement, flexible, self-organised, small group, variable content, like
7: **programmed text:** involvement, rigid, equipment, self-organised, small group, specific content, dislike
8: **video tape:** flexible, equipment, self-organised, small group, specific content, dislike
9: **informal interaction:** involvement, flexible, no equipment, self-organised, variable content, like

Figure 45: Classification of constructs by elements

### 3.4.7 *Classes as ideal elements or compound constructs in grids*

The intersects specifying classes may be also be represented, in part, as ***ideal elements*** or as ***compound constructs***. The subordination relations between classes may be expanded and included in the ideal elements, but the preference relations may not, and are included as notes. Figure 47 shows the contact lens grid displayed with its classes as ideal elements.

The ***Analyze*** row in the Classes pane may be use to specify that all classes, or selected classes, are including in grids being analyzed. This inclusion is only temporary for the purposes of the analysis. The ***Case*** and ***Rule*** checkboxes determine whether the classes are represented by elements or constructs, respectively. The ***Add to Grid*** button makes it permanent. Further examples are given in the appropriate analysis sections.

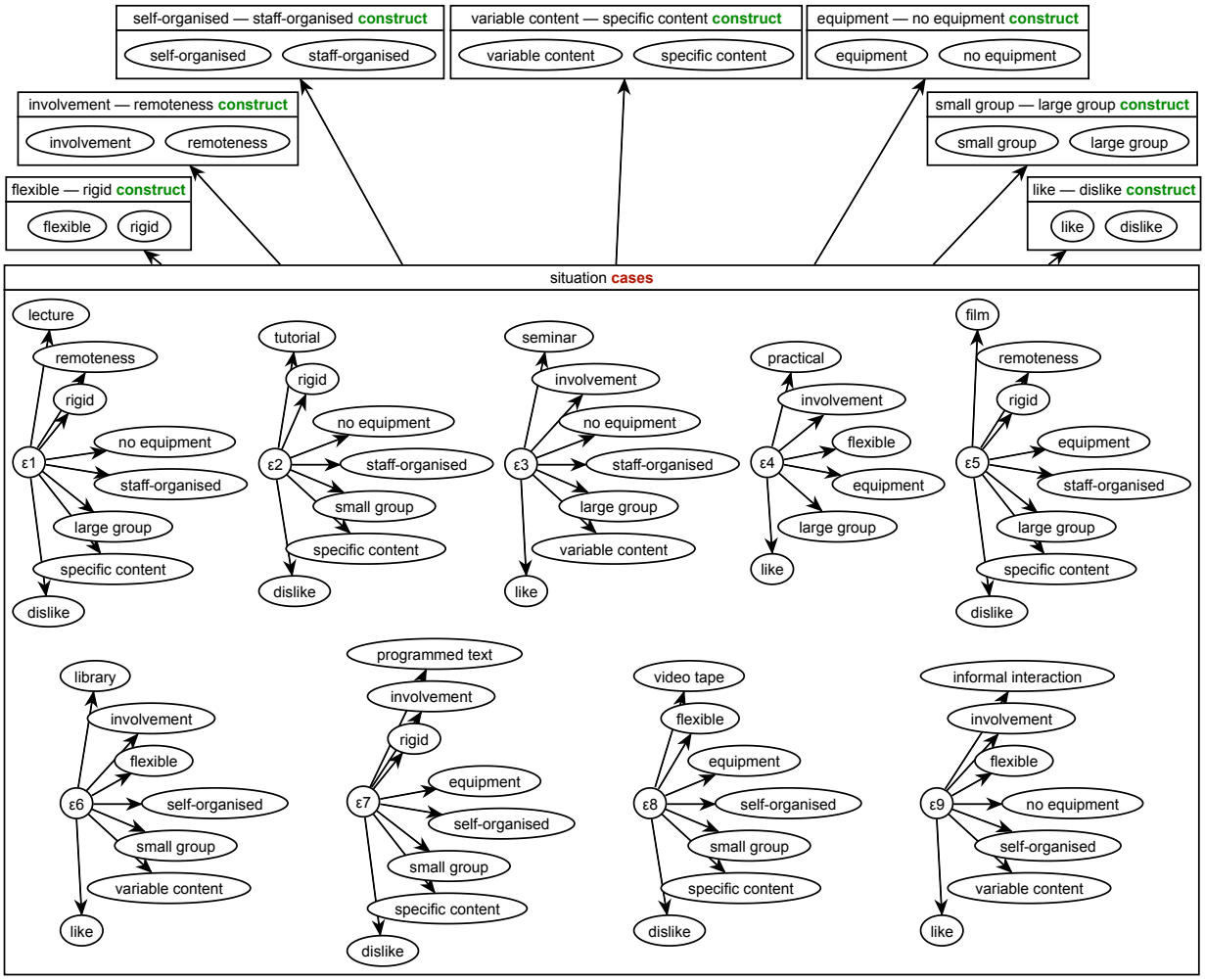



Figure 46: Conceptual net for *learning situations* grid

## 3.5 Items pane

RepGrid makes provision for additional items of data to be added to a grid, stored with it, and edited. These items may be viewed and edited through the ***Items*** pane. Clicking on the ***Items*** tab in the ***House Choice*** RepGrid window brings up a pane listing the items by name and value (Figure 48).

The first five items shown are automatically generated when a new grid is created: a unique identifier string and the date, time, and location (the IP of the machine on which the grid was created), and the status of the grid indicating whether is new or derived from an existing grid.

The next four items are fields entered in WebGrid to control the styling of the web pages generated in eliciting and analyzing this grid. It is sometimes convenient to edit these fields in RepGrid.

The final three items keep track of the RepGrid window position and size, and the column positions of the tables on the ***Elements*** and ***Constructs*** panes.


Figure 47: Grid with classes represented as ideal elements

Items may be deleted by selecting the item row and pressing the ***delete*** key. Names and values may be edited by clicking in the cells and editing the text. Additional fields may be added by clicking on the ***Add*** button and entering the item name and value.

Multi-line values may be edited by double-clicking on the item number which brings up a multi-line editor as shown below for item six which defines some styles in the header of a WebGrid page (Figure 49).

### *3.5.1 User-defined items*

You can define your own items to use in storing additional data in a grid file, and it is common for elicitation scripts to collect additional data and store it in this way. RepGrid will not allow such items to have names that correspond to its own reserved names, and to avoid possible clashes user-defined items should commence with an underline, _, character.

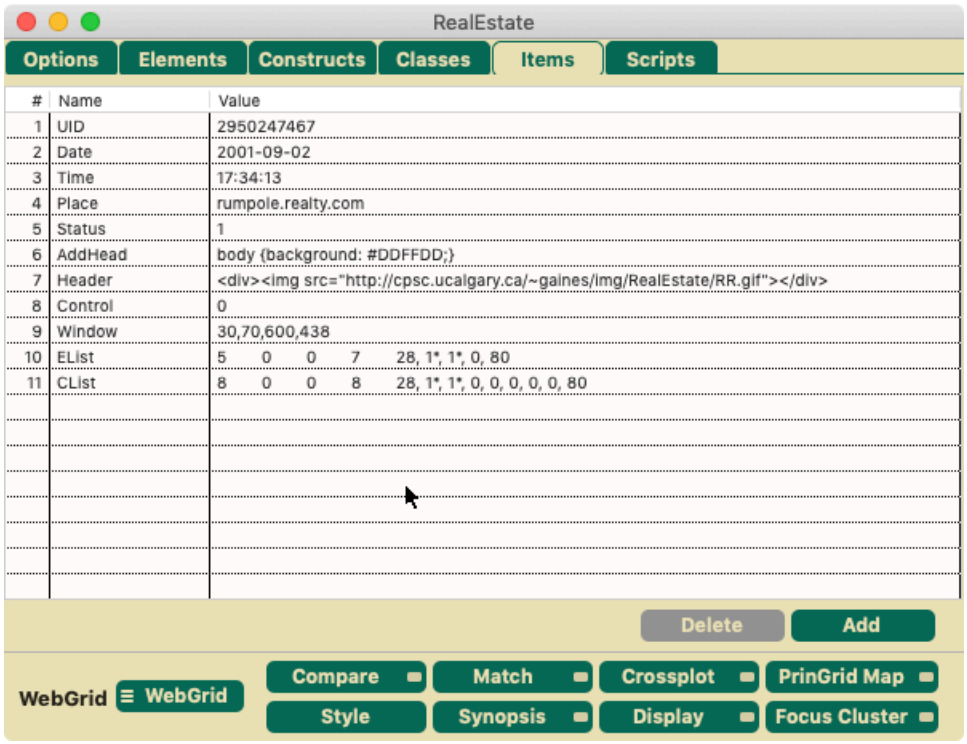



Figure 48: RepGrid ***Items*** pane showing additional items included with grid data

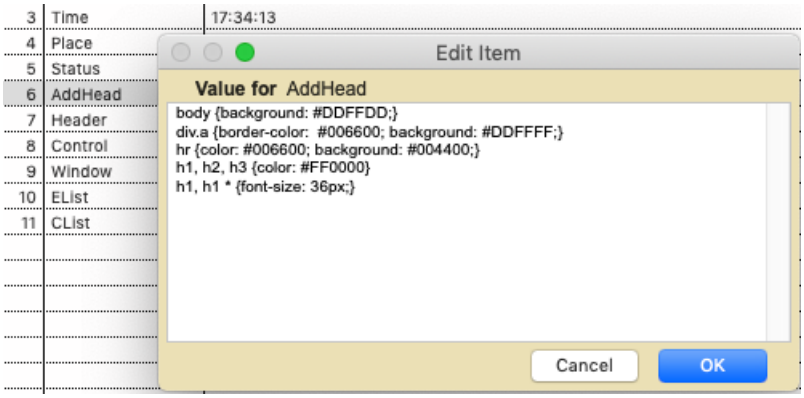



Figure 49: RepGrid ***Items*** pane editing a multi-line value

## 3.6 Scripts pane

RepGrid has an open architecture through the capability to run scripts written in ***RepScript*** that have full access to read and modify the grid data (see RepScript manual). The scripts are text files that users can develop to support their own analyses, such as cognitive complexity measures, or to import and export grid data in other formats.

Clicking on the ***Scripts*** tab in the RepGrid window brings up the ***Scripts*** pane that allows such scripts to be run (Figure 50). The script button/popup menu at the top right enables a script to be chosen and run. It is dual-purpose in that clicking in the menu symbol on the left (or right-clicking/CTL-

clicking anywhere in the button) brings up a menu which enables one to select the script named in the button, and clicking in the script name causes the script to run.

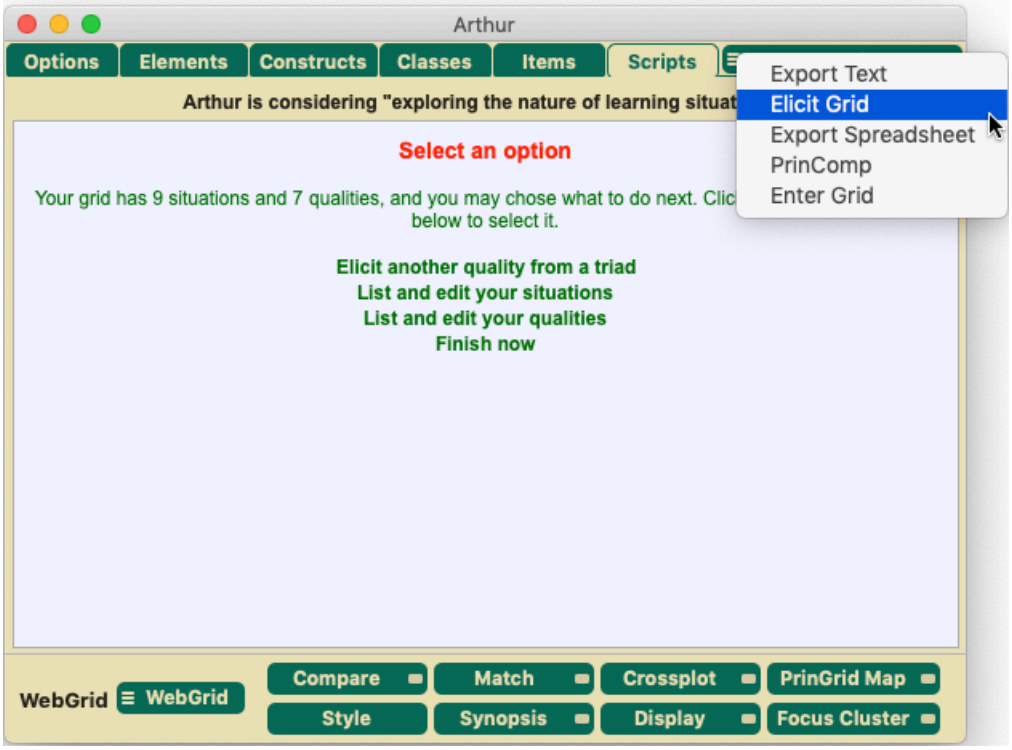



Figure 50: RepGrid *Scripts* pane and scripts menu

Just under the tabs is a line of text that can be set under script control, for example, to indicate the purpose, or status of the script. Under that is an interactive text pane that may be programmed by a script to conduct an interactive dialog, including clickable menus, or used to display status information or results. The pane is shown running the ***Elicit Grid*** script and offering various elicitation options.

**A script may be terminated at any time by pressing the *esc* key.**

Scripts have full access to all data in the grid, can access files and interact with users. Hence they may be used to import and export grids in any specified format, enter grids, elicit them interactively, edit them, and so on. Scripts are text files which users may modify and create their own customized scripts. In particular they may translate them into other languages. Rep Plus uses a Unicode text representation so that non-Roman scripts may be used. The display and analysis tools have been designed to use only the text entered into the grid so that they also support graphic output in the language being used.

The initial set of scripts currently supplied with RepGrid include ***Enter Grid*** for rapid entry of grid data, ***Elicit Grid*** for conversational elicitation of a grid, and scripts to export grids in various formats. They provide basic facilities for grid entry, elicitation and export, and also serve as exemplars for those developing custom scripts.

The supplied scripts are within the Rep Plus application directory ***GridScripts,*** and the name of the script is what appears on the popup menu. User scripts may be put in the ***GridScripts*** directory in the ***Rep Plus*** directory created automatically in the ***Documents*** or ***MyDocuments*** directory. If certain scripts are intended for use only with certain types of grids they may be placed in a ***GridScripts*** within the directory containing the scripts, and will only appear in the scripts menu of those grids.

# 4 Grid entry, elicitation and export scripts

The scripts supplied with RepGrid accessible through the ***Scripts*** pane are designed to support rapid entry of existing grid data, conversational elicitation of new grids, and export of grids in other formats.

## 4.1 Enter Grid

The ***Enter Grid*** script allows existing grid data to be entered rapidly. The script is programmed to:-

- Request missing fields in the ***Status*** window such as the user's name and the purpose of the elicitation
- Ask the user to enter the elements
- Ask the user to enter the constructs and their ratings on the elements
- Offer the user the option to edit and enter more elements or constructs, or to finish the entry process

The figures below illustrate the ***Enter Grid*** script being used to enter the ***Arthur*** grid used to illustrate this manual. The first screen shows the name and purpose being entered as text terminated by the ***return*** key (Figure 51).

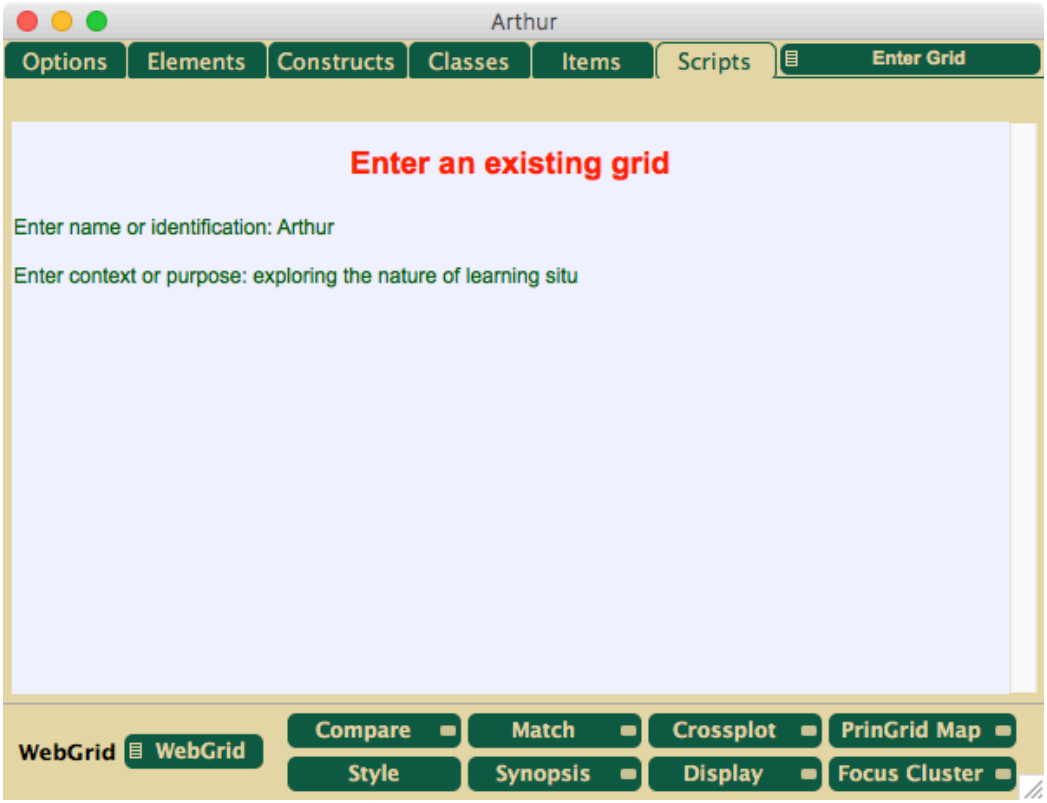



Figure 51: ***Enter Grid*** script entering name and purpose

The entry process is similar to that of a text editor. The ***delete*** key may be used to erase text, and text may be selected, copied and pasted.

The ***Enter Grid*** script may be run immediately with a new grid, or with one where the name and purpose have been entered on the ***Status*** pane, and/or the elements have been entered on the ***Elements*** pane. **If the rating scale is other than 1 to 5 or the terms for elements and constructs are different then these should be set up in the *Status* pane before the script is run.** In the example below elements have been termed ***situations*** and constructs ***qualities.***

The script offers the facility to edit elements, constructs and ratings, and this may also be done through the ***Elements*** and ***Constructs*** panes. The ***Display*** button provides a convenient way of checking the data entered against its source to ensure that it is correct.

After the purpose has been entered, the element names are entered (Figure 52). Keying ***return*** with an empty name terminates the element entry and proceeds to the constructs and ratings entry.

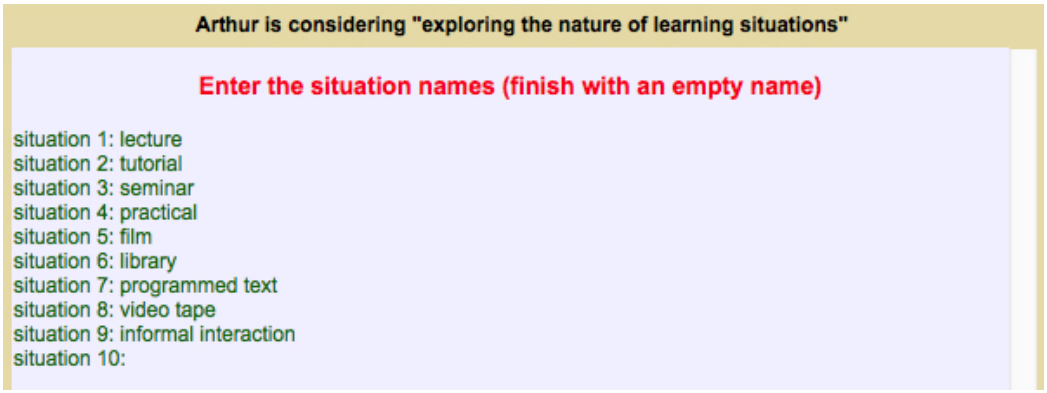



Figure 52: ***Enter Grid*** script entering element names

Figure 53 shows the pole names of the first construct being entered. Keying ***return*** after the right pole name proceeds to the ratings entry for that construct.


Figure 53: ***Enter Grid*** script entering pole names of the first construct

Then the ratings on each element are requested (Figure 54). They can be typed in as numbers followed by ***return***, or by clicking the mouse and selecting a rating from a popup menu.

When all the elements have been rated on the first construct, the pole names of the second construct are requested, and so on, until all the constructs and ratings have been entered. Construct entry is terminated when an empty pole name is entered, and the grid editing options are listed (Figure 55).

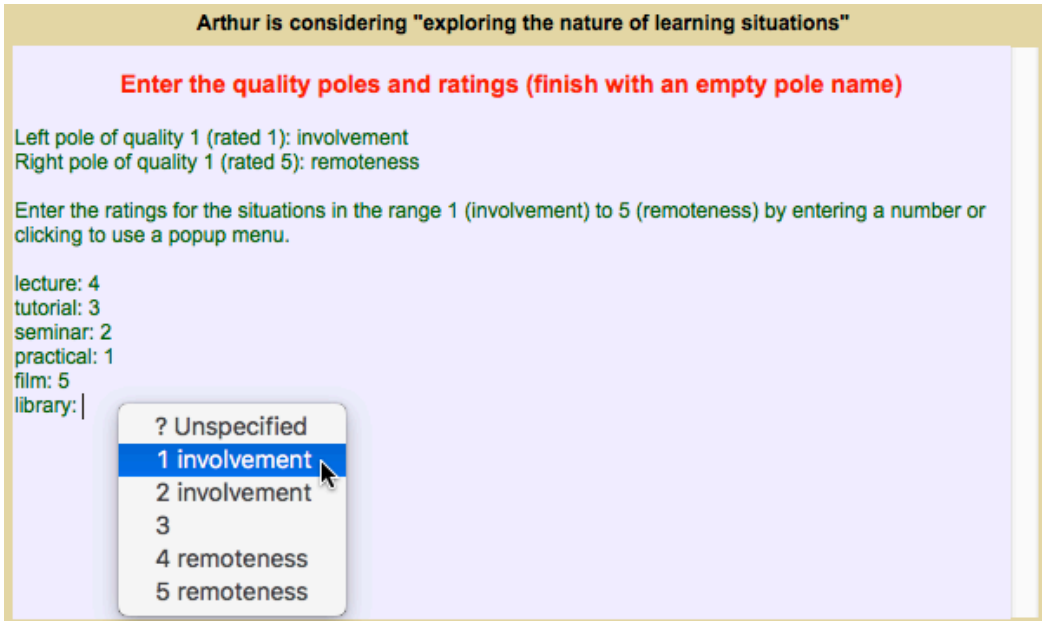



Figure 54: ***Enter Grid*** script entering pole names of the first construct

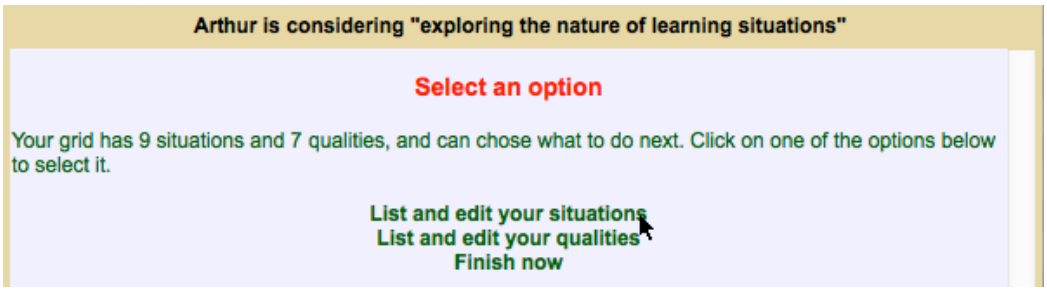



Figure 55: ***Enter Grid*** script options after grid entry is complete

The ***Display*** button may be used to display the grid to check that it has been entered correctly. If not, one the list of options displayed may be clicked to list the elements or constructs may be listed and allow the names and ratings to be edited. Clicking on ***List and edit your situations*** shows the elements and provides options to add more or to edit an element name and ratings (Figure 56).

Clicking on the element ***practical*** displays its name and its rating on each construct (Figure 57).

Clicking on the option to delete the element displays a warning and options to do so or cancel (Figure 58).

Clicking on the element name makes it available for editing as shown below (Figure 59).

Clicking on a construct makes the rating on the selected element available for editing (Figure 60).

Similarly, the option in Figure 55 to list and edit the constructs may be chosen. They are then listed for editing (Figure 61).

Clicking on a construct displays it for editing (Figure 62). The editing procedures for the pole names and the ratings are essentially the same as those described above for the elements.

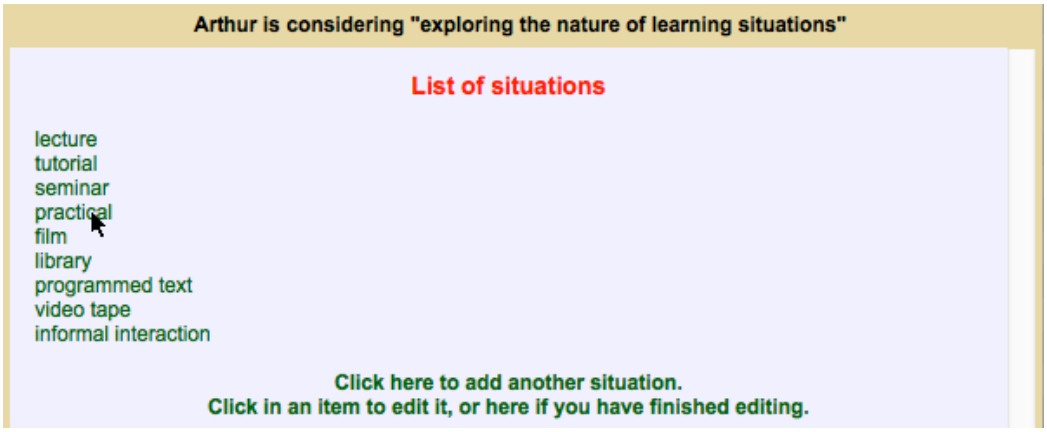



Figure 56: *Enter Grid* script listing elements

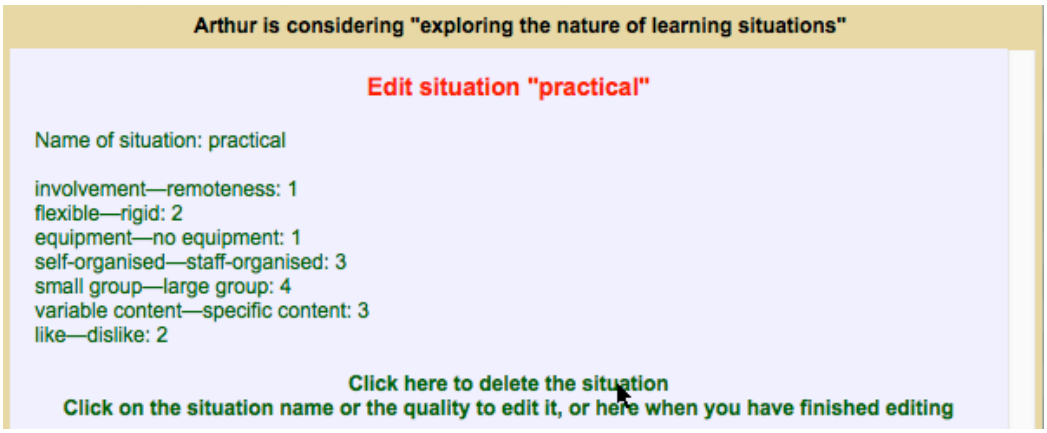



Figure 57: *Enter Grid* script editing a selected element

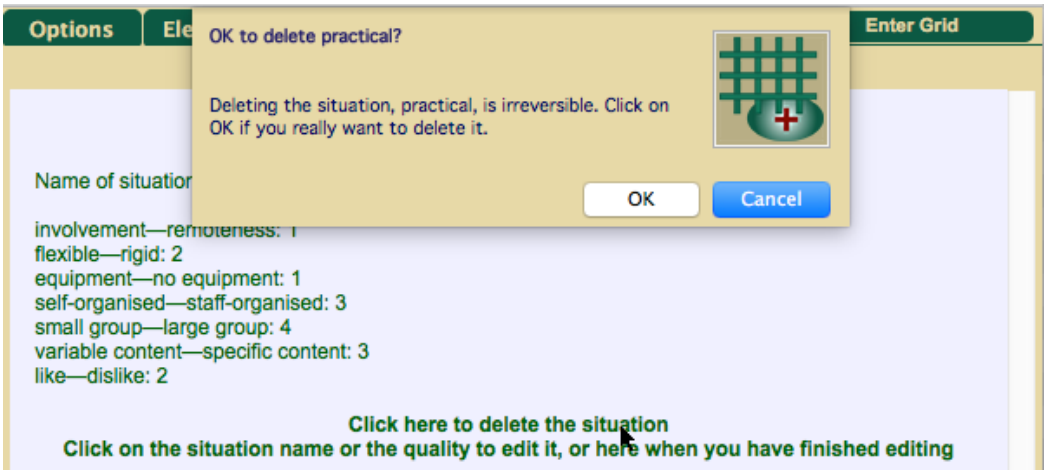



Figure 58: *Enter Grid* script deleting the selected element

The *Enter Grid* script may be run at any time with an existing grid and will open with the editing options of Figure 55 enabling the grid to be edited conversationally as described above as an alterna-

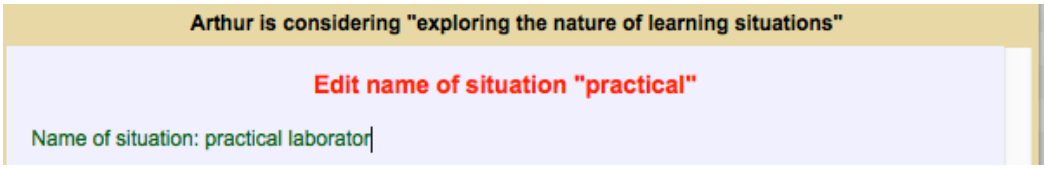



Figure 59: *Enter Grid* script editing an element name

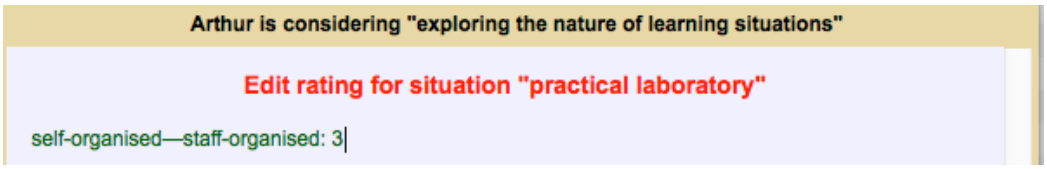



Figure 60: *Enter Grid* script editing a rating on a constructt

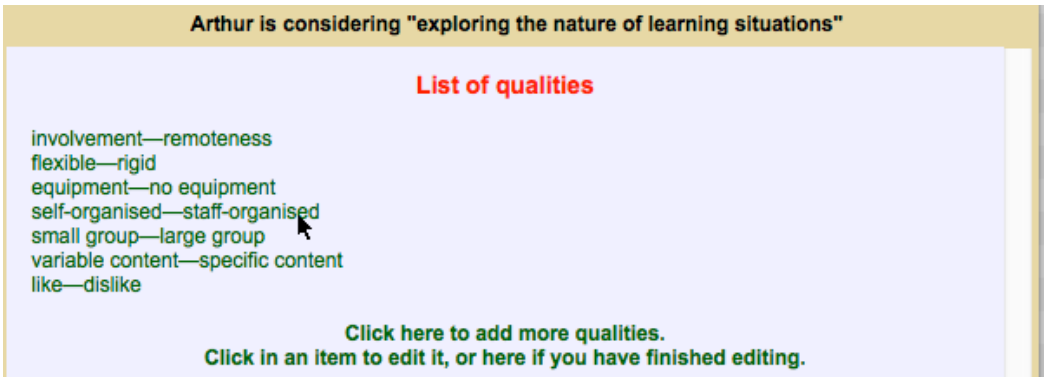



Figure 61: *Enter Grid* script listing constructs

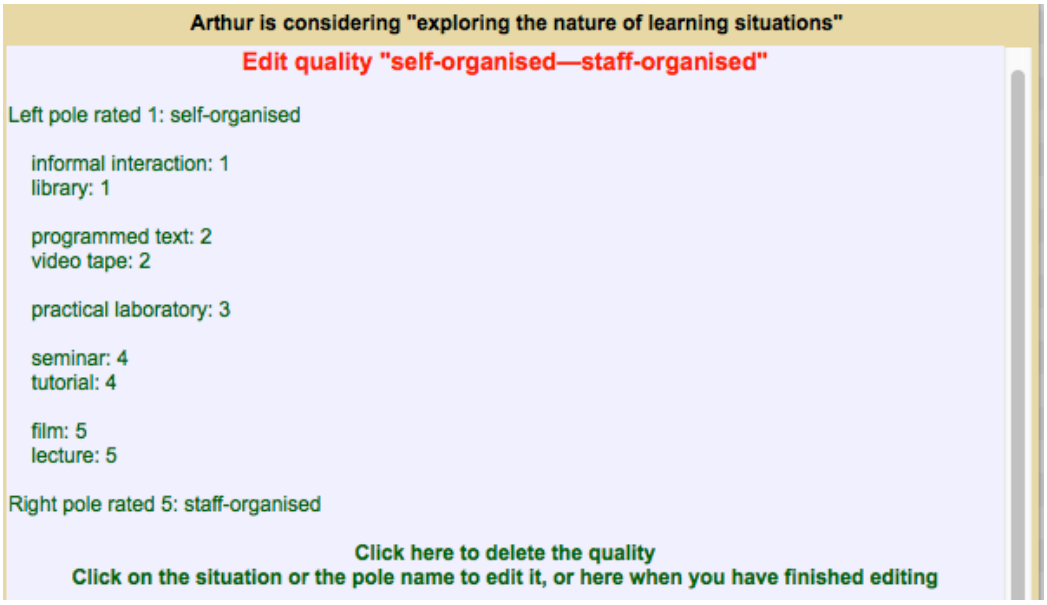



Figure 62: *Enter Grid* script entering pole names of the first construct

tive to editing through the ***Elements*** and ***Constructs*** panes. The editing is synchronized between the scripts and the panes so that it is possible to switch back and forth between them if desired.

## 4.2 Elicit Grid

The ***Elicit Grid*** script supports conversational elicitation of conceptual grids by emulating the Shaw's (1980) interactive repertory grid elicitation program, "PEGASUS".

The script is programmed to:-

1. Request fields that are empty in the ***Status*** window such as the user's name and the purpose of the elicitation
2. Ask the user to enter six or more elements
3. Elicit constructs from triads of elements until there are four
4. Check element and construct matches and offer the user the opportunity to enter more constructs or elements to break the matches
5. Offer the user the option to elicit more constructs from triads, edit and enter elements or constructs, or to finish the elicitation
6. Ask the user to rate the elements on any given constructs when the elicitation process is finished
7. Modify the elicitation process appropriately to elicit ratings for ***exchange*** grids in which the elements and constructs are given but the ratings are open, ***elements*** grids in which the ***elements*** are given, and ***constructs*** grids in which the constructs are given.

Once steps 1 through 3 are complete the scripts loop between steps 4 and 5.

At any time during the elicitation the user can click on the analysis facilities available through the ***Display, Synopsis, Focus, PrinGrid***, and so on, buttons to display or analyze the grid, and then continue the elicitation.

The ***Elicit Grid*** script operates in a similar way to the ***Enter Grid*** script except that, as shown below, the interaction is more tutorial with and greater explanation feedback of matches to suggest appropriate element and construct elicitation.

Clicking on the script menu option ***Elicit Grid*** brings up an initial explanatory dialog (Figure 63).

The initial elements are entered in much the same way as for the ***Enter Grid*** script (Figure 64).

The element elicitation section of the script can be edited to accommodate more specific requests for particular types of element as, for example, might be appropriate to a ***role grid*** requesting that family members and friends be entered, a core competency grid requesting that employees with different types and levels of skills be entered, or a market research grid requesting that products of certain categories and qualities be entered.

The ***Elicit Grid*** script differs from the ***Enter Grid*** script primarily in that it elicits constructs using triadic elicitation in which the user is asked in which way two elements are alike and differ from a third, and in the feedback of element and construct matches to prompt the elicitation of further constructs and elements to reduce the matches (Shaw, 1980).

Three elements are selected by the script and the user is asked to consider in what way two are alike and different from the third, and to click on the one that is different (Figure 65).

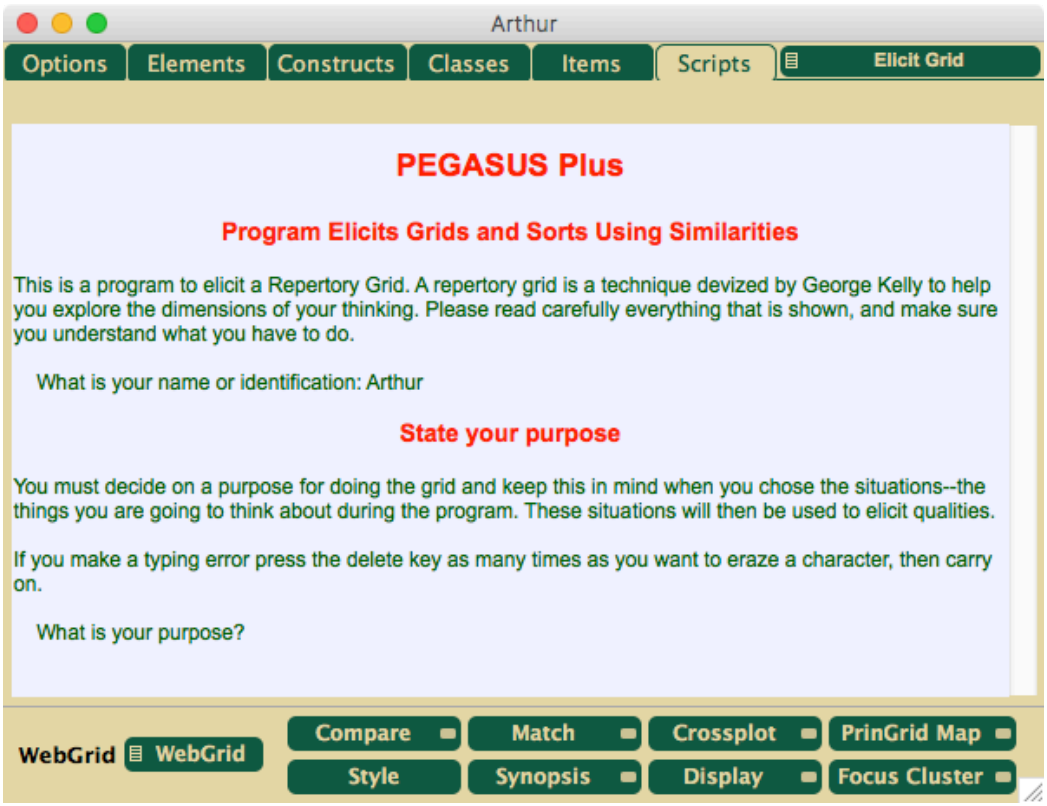



Figure 63: Initial *Elicit Grid* script requesting name and purpose

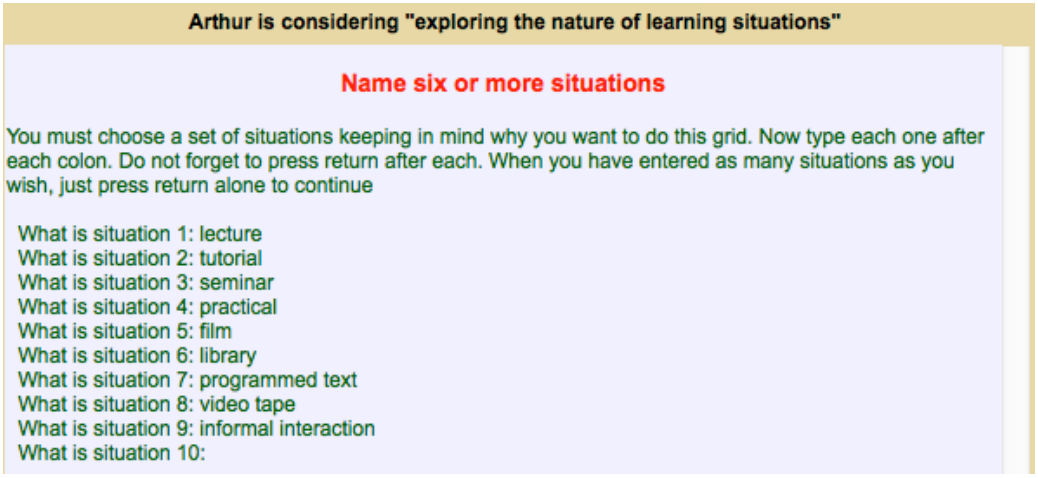



Figure 64: *Elicit Grid* script entering element names

When the user clicks on an element in the triad they area asked to enter the construct pole names construct pole names for the distinction being made (Figure 66).

They are then shown the the three elements of the triad rated appropriately and asked to rate the remaining elements on the elicited construct (Figure 67).

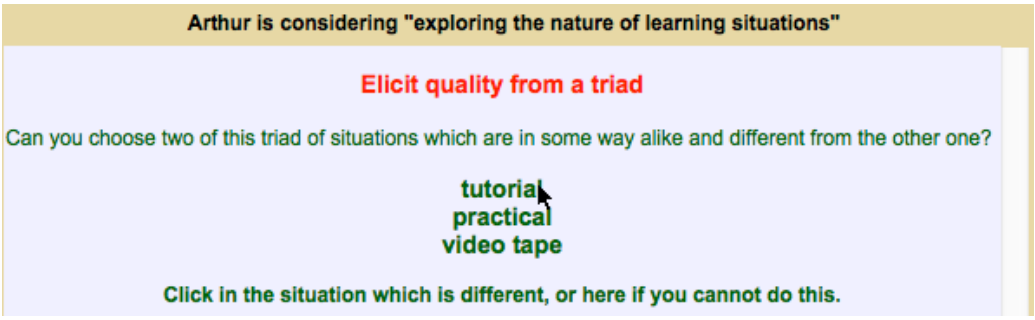



Figure 65: *Elicit Grid* script eliciting a construct from a triad of elements

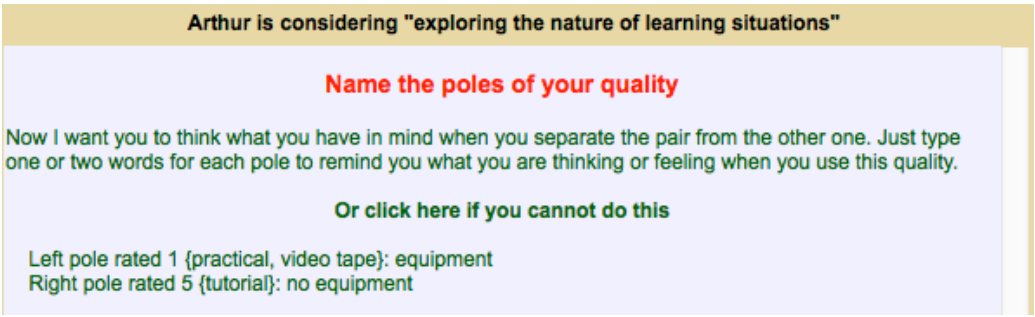



Figure 66: *Elicit Grid* script entering construct pole names

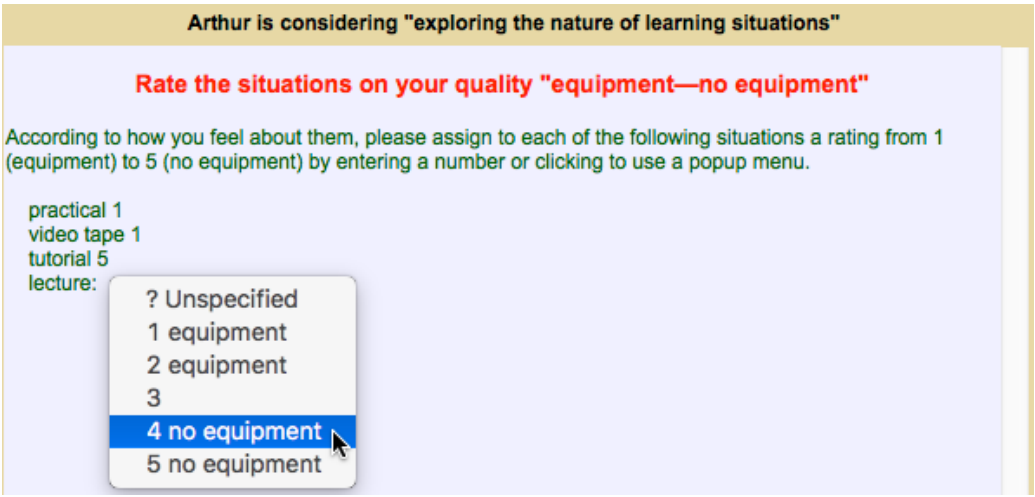



Figure 67: *Elicit Grid* script entering element ratings

When all the elements have been rated the screen in Figure 68 is shown which makes the pole names and ratings available for editing as has been shown for the *Edit Grid* script. Since the process is now one of elicitation rather than data entry it is likely that the user will not be content with the ratings as initially entered and will edit them at this point as illustrated in the previous section.

Construct elicitation continues with further explanation and another triad of elements (Figure 69).

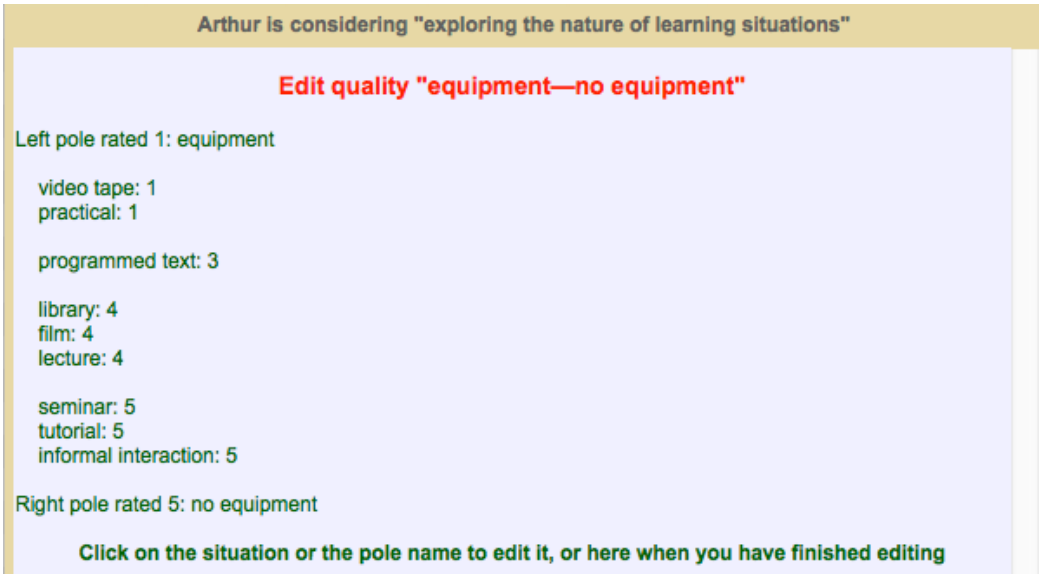



Figure 68: *Elicit Grid* script entering element ratings

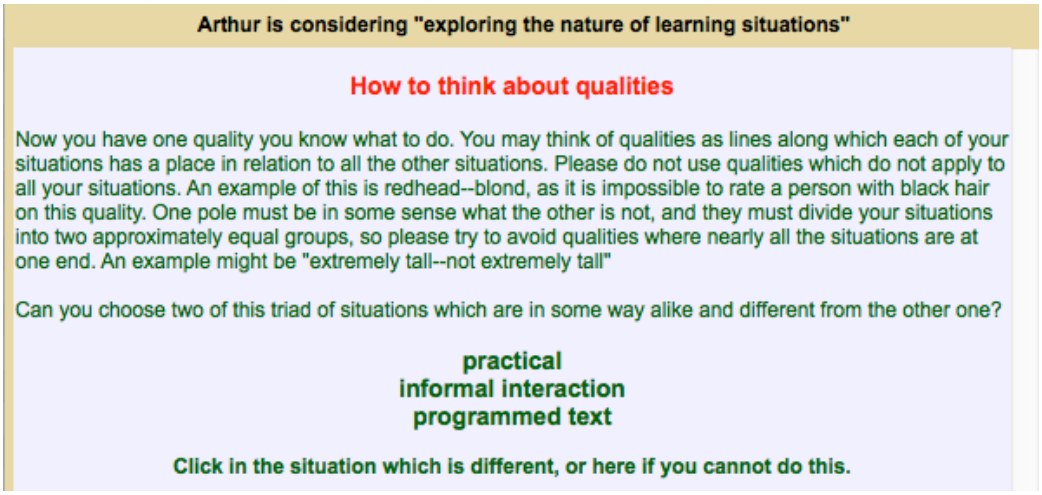



Figure 69: *Elicit Grid* script eliciting a construct from a second triad of elements

When four constructs have been elicited through triads the script tests for matches between constructs and between elements and, if it finds any above eighty per cent, asks the user to enter an element or construct to reduce the match (Figure 70).

The user clicks on ***self-organized and rigid,*** and enters a new element, ***programmed text,*** that is construed as both and should reduces the construct match (Figure 71).

The new element is then shown already rated appropriately on the matching constructs and the user is asked to rate it on the remaining constructs (Figure 72).

When the new element has been rated the name and ratings can be edited as already shown. The test for matches continues and notes an element match which prompts the elicitation of a new construct (Figure 73).

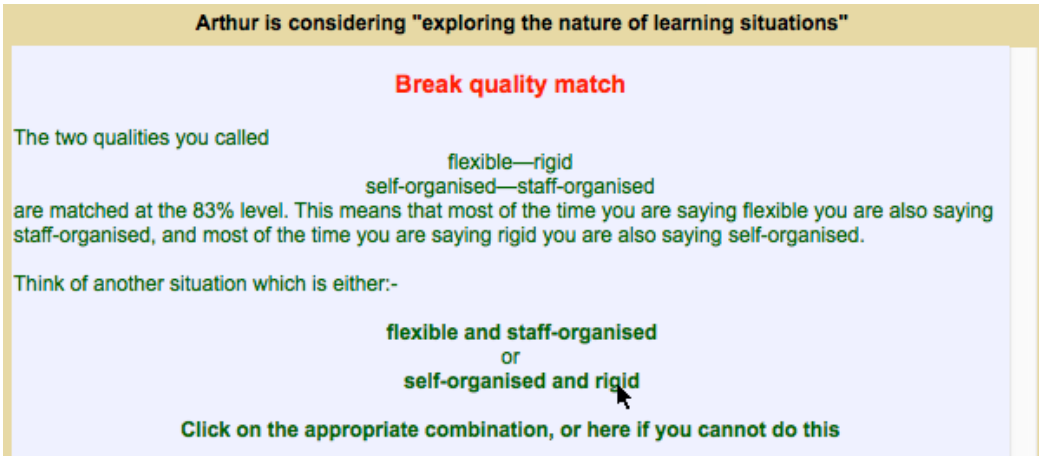



Figure 70: *Elicit Grid* script eliciting a construct from a triad of elements

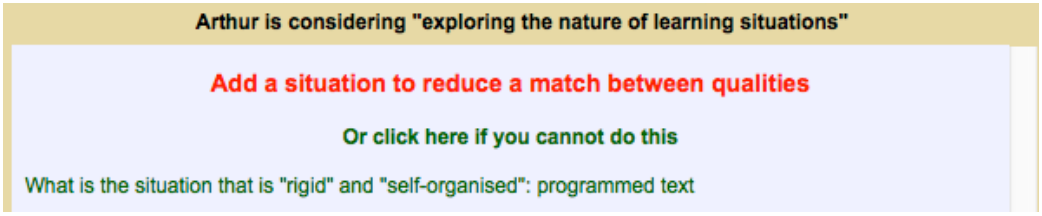



Figure 71: *Elicit Grid* script eliciting a construct from a triad of elements

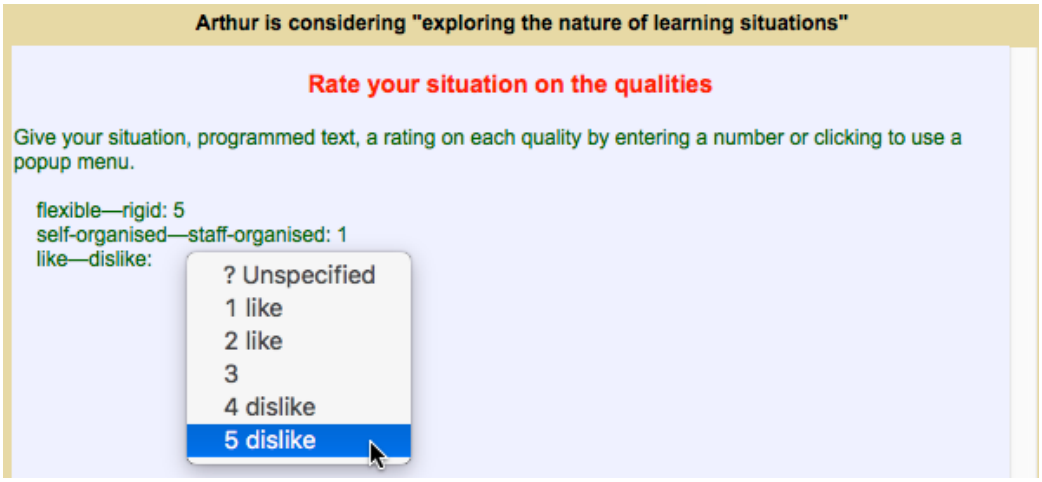



Figure 72: *Elicit Grid* script eliciting a construct from a triad of elements

The new construct is then shown with the matched elements already rated appropriately on the new construct and the user asked to the remaining elements on the new construct (Figure 74).

The process continues until there are no further matches to be displayed and various options for further elicitation are then listed (Figure 75).

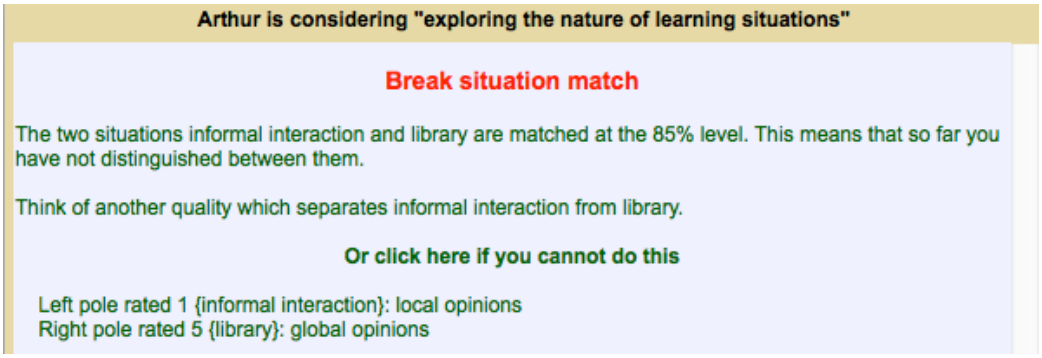



Figure 73: *Elicit Grid* script eliciting a construct from a triad of elements

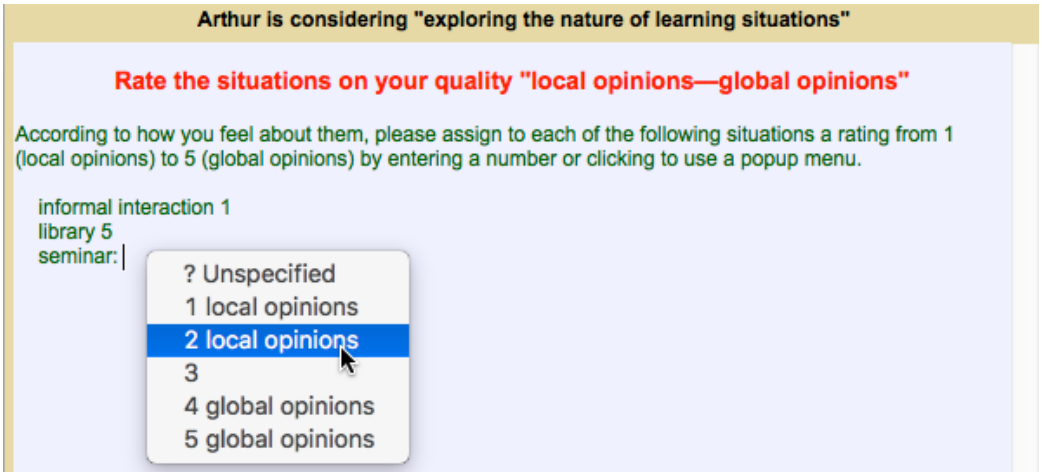



Figure 74: *Elicit Grid* script eliciting a construct from a triad of elements

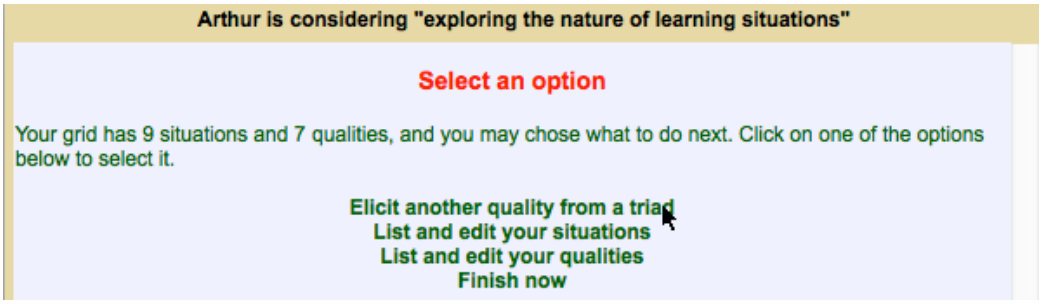



Figure 75: *Elicit Grid* script options

The options are the same as those for the *Enter Grid* script with the addition of one for triadic elicitation. Selecting this now also provides an option for the user to select some or all the elements of the triad (Figure 76).

The user can also use the editing facilities to add and rate elements and constructs directly. The test for matches is applied each time the user enters an item so that further feedback is given if appropriate. This process of elicitation from triads and matches, and entry and editing, proceeds until the user is satisfied that the grid is relatively complete and chooses the option to finish.

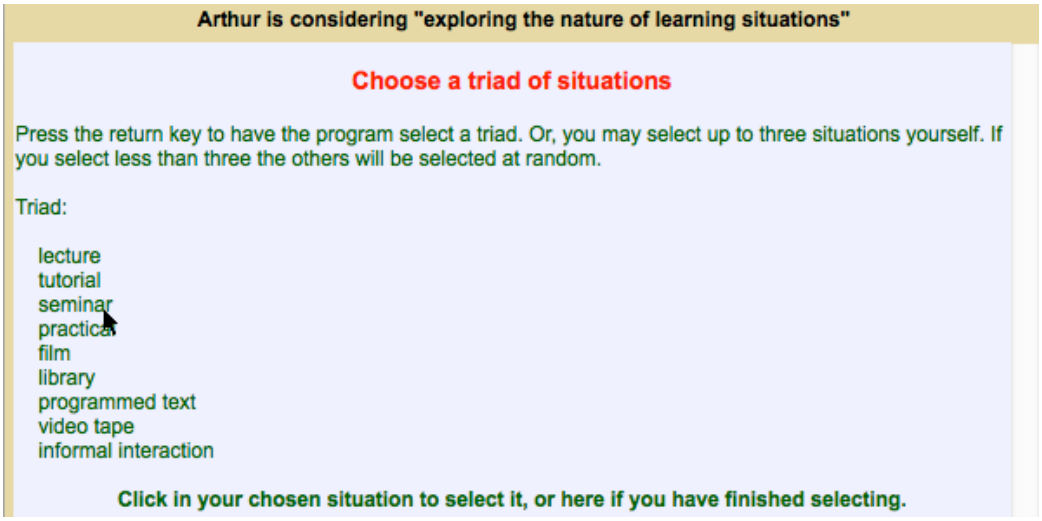



Figure 76: ***Elicit Grid*** script selecting elements for a triad

Before finishing a check is made for unrated constructs such as ***given constructs*** provided by the facilitator and the user is asked to rate the elements on these if there are any.

The ***Display, Synopsis, Focus Cluster, PrinGrid Map*** and ***CrossPlot*** buttons may be used during the elicitation to supply an interim display or analysis of the grid. Viewing this may prompt the entry of further elements and constructs or the editing of the grid.

The ***Elicit Grid*** script can be run with an existing a grid at any time to elicit further elements or constructs and will automatically skip steps 1, 2 or 3, if the grid already contains the appropriate data. For example, one can prepare a grid for use by others with the purpose specified as ***to understand people I know,*** with ***person*** as the term for an element, ***people*** for elements, ***characteristic*** for construct, ***characteristics*** for constructs, and with initial elements ***self*** and ***ideal self.*** Running the ***Elicit Grid*** script set starting with a copy of this grid will result in the user being asked their name, being asked for additional examples of people, having four initial constructs elicited from triads, and then being taken through an elicitation process for further elements and constructs. One can also enter some given constructs such as ***powerful—powerless*** that capture an important aspect of the purpose of the elicitation, and, as already noted, the script will ask the user to rate the elements on these as they finish.

To facilitate comparison of grids the ***Rep Plus Manager*** window allows one to open a copy of an existing grid with the ratings reset to be open (***Exchange***) or with the constructs removed (***Elements***) or the elements removed (***Constructs***). Another user, or the same user at a later time, can fill in the ratings and add constructs and elements to such grids for comparison with the original (§5.6). It first requests the user's name. The ***Elicit Grid*** script supports elicitation commencing with such partial copies of grids. It first requests the user's name and then: with ***Exchange*** grids it asks the user to rate all the elements on each construct in turn; with ***Elements*** grids it proceeds immediately to triadic construct elicitation; with ***Constructs*** grids it requests elements and then asks the user to rate all the elements on each construct in turn. It then offers the normal options.

## 4.3 Export grid data

Two scripts are supplied which allow the grid data to be exported in formats suitable for use in other applications.

The ***Export Text*** script writes the grid data into a text window in the basic grid format described in §9.1.

The ***Export Spreadsheet*** script writes the grid data into a text window in the tab-delimited grid format described in §9.2.

The data in the text window can then be edited, saved, copied and pasted, or dragged to another application.

## 4.4 Analyze grid data

Scripts have full access to all the information in the grid and may be used to provide additional analysis capabilities. The ***PrinComp*** script is provided as an example of the use of the mathematical and graphic libraries available in RepScript to generate a principal components analysis with graphical and textual output.

## 4.5 Modifying scripts

The scripts supplied are intended as examples that a facilitator can copy and modify to serve specific requirements and user communities.

RepGrid looks for GridScripts directories both in the application directory and in the ***Rep Plus*** directory where the default files are kept as discussed in §2.2. The ***Rep Plus*** directory in the ***Documents*** or ***MyDocuments*** directories is intended for scripts developed by the facilitator.

For example, one might copy the ***Elicit Grid*** scripts from the Rep Plus application directory, rename them with an appropriate name to appear in the popup menu, and edit the ***Main*** script to make the initial element elicitation more specific to the purpose, for example, by requesting the names of the user's mother and father, good friend, teachers, and so on.

One might also develop highly specific elicitation procedures for specific purposes such as market research, knowledge management or system design requirements elicitation.

It is also reasonably straightforward to translate the existing scripts to support conversational elicitation in languages other than English by replacing the English text with the equivalent in another language. It may be necessary to slightly reorder the output to reflect the different literary style of the target language.