5 Determining the Methodology to Test the Conceptual Graphs Approach

5.01 Introduction

The previous chapters indicated the potential benefits of conceptual graphs to the strategic management accountant. The subject matter of this chapter refines those general arguments into the practical environment in which conceptual graphs may best be presented to the strategic management accountant. The discussion starts by examining the practical considerations involved when the domain knowledge is transferred between the domain expert and the computer, moving onto the particular appropriateness of conceptual graphs. The need for conceptual graphs as computer software is then identified, together with the inadequacy of existing such software. From all these bases the Conceptual Analysis and Review Environment, or CARE for short, is devised. The CARE software employs the more promising conceptual graphs processor, JEHCGP, which is also discussed. CARE is the vehicle through which conceptual graphs are user evaluated in the following chapter.

5.02 The Practicalities of Eliciting and Reviewing Knowledge

With any knowledge-base system there must be some means of sufficiently transferring domain knowledge between the domain expert and the computer. As mentioned in Chapter 1, this practical process is known as knowledge elicitation. Jackson (1990) summarises the experiences with an early knowledge elicitation technique, protocol analysis\(^1\), as follows (page 23):

\[\text{Newell and Simon generated a kind of knowledge representation known as production rules ...}^2. \text{ They also pioneered a technique known as protocol analysis, whereby human subjects were encouraged to think}\]

\(^1\) Detailed by Ericsson and Simon (1984).
\(^2\) Discussed in Chapter 1.
aloud as they solved problems, and such protocols were later analysed in an attempt to reveal the concepts and procedures employed. This approach can be seen as a precursor of some of the knowledge elicitation techniques that knowledge engineers use today\(^3\). These psychological studies showed just how hard the knowledge representation problem was, but demonstrated that it could be addressed in a spirit of empirical enquiry, rather than philosophical debate.

The knowledge acquisition stage has also been referred to as the ‘bottleneck’ in knowledge-based applications development (Feigenbaum, 1977). Hence knowledge elicitation is itself no trivial exercise, and thus needs to be properly considered. Its importance cannot be overlooked: This thesis has already demonstrated the fundamental dangers arising from the domain expert’s knowledge being inadequately computer-encoded.

In their study of failed management accounting expert systems, King and McAulay (1992) suggest two key properties that an expert system should possess to overcome the knowledge elicitation bottleneck. Their first proposal is for a pre-installed library consisting of an established academic body of domain knowledge, from which the domain expert could build a tailored model according to his or her particular circumstances. King and McAulay explain that such knowledge is more thorough than the relatively ‘patchy’ knowledge gained by the expert in the field. Their second proposal is that the domain expert should somehow be able to enter his or her knowledge into the computer via a ‘checklist’, as this is the method by which human experts present their knowledge. In essence, the expert would communicate to the system simply via a list of domain knowledge elements considered as relevant to some problem of interest. Thus, apart from any familiarity it may have with the domain expert, the checklist approach portrays a practical means of freeing the domain expert to concentrate purely on domain knowledge. The domain expert is thereby not burdened with the added computational overload hurdle first identified as fundamental in the early part of Chapter 2. However those checklist-based expert

\(^3\) The role of the ‘knowledge engineer’ was mentioned in Chapter 1.
systems must not be too shallow, as we know from Chapter 1. As a reminder, Leith (1991) aptly makes the point (page 190):

.... the systems which are being called expert systems are no more than checklists, and the explanations they give, no more than reading off the ticks or crosses on the checklist.

Bearing Leith’s criticism in mind, King and McAulay’s proposals are further discussed later. As already established by this thesis, Sangster (1991) discusses the need for human domain expert review in management accounting. Sangster typifies the warning that management accounting knowledge does not lend itself easily to formal models. Thus many studies, such as Back-Hock (1991) and Levary and Madeo (1991), suggest knowledge-based systems should be adaptive in nature. In this portrayal the knowledge in the computer is built up from continual iterations of activity with the human expert, rather than that expert having to accept an outright formal model at the outset. Back-Hock explains one such instance, the ‘product life cycle’ model. The formal knowledge inherent in this model needs to be constantly tailored by the real-world user, so leading to what Back-Hock calls the ‘integrated product life cycle concept’.

From all the above it can be seen that there appears to be a need for both relatively static and dynamic elements of knowledge. However, The Economist magazine (1993, March 20) lucidly reveals that there is little agreement about what business strategy is. The strategic problem domain thereby renders truly static domain knowledge as significantly inappropriate, so the scope for King and McAulay’s first proposal above about a body of established academic knowledge becomes severely limited. All this heightens not only the need for an interactive environment, but also one that the domain expert can be comfortable with given its consequently expected heavy use.

The discussion in this thesis has shown domain experts working directly with expert systems. O’Brien, Candy, Edmonds and Foster (1992) reveal the advantages of this situation. O’Brien et al. refer to
such systems as ‘End-User Knowledge Manipulation Systems’, or EUKMS for short. They state (page 4):

Minimally, the definition of EUKMS is the provision of software support which enables a knowledge worker\(^4\) to encode, refine and manipulate domain knowledge, in a machine-usable form, without the continuing mediation of the knowledge engineer. EUKMS diminish the conceptual gap between a knowledge worker’s expertise and a machine-tractable representation of that knowledge, by providing a medium of communication which is founded upon domain concepts and terminology. Our concern extends beyond the provision of tools which knowledge-workers can use, but do not otherwise augment the cognitive communication between user and system. A general goal for the design of EUKMS is to provide an interactive knowledge acquisition tool or environment with which knowledge can be elicited, encapsulated and encoded, using domain compatible external knowledge representations which are meaningful to the user. The computational representation required by the target system is independent of the external knowledge representation, and should be concealed from the knowledge worker.

The processes of eliciting and reviewing the kinds of knowledge that the strategic problem domain demands means that, in practice, an expert system would best support the strategic management accountant by such systems being highly interactive with that domain expert. This integration between human user and computer could best be achieved by allowing domain experts to directly state and query elements of domain knowledge in user-familiar terms. As the interaction developed, the human-computer dialogue could extend beyond superficial checklists of knowledge. Therefore King and McAulay’s second proposal above, about the advantages of a domain expert entering a checklist of that expert’s knowledge, would thus also be effectively satisfied without making the expert system too trivial. This is especially because the user would be interacting with domain knowledge alone.

\(^4\) ‘Knowledge worker’ is their term for domain expert.
5.03 Clarifying The Appropriateness of Conceptual Graphs

Although the matter has already been referred to extensively in this thesis, we now turn to emphasise how conceptual graphs could assist in the above practicalities of eliciting and reviewing knowledge. To begin with, a psychological study by Novak and Gowin (1984) demonstrated how the structured diagrammatic nature of their simple ‘concept maps’ elicited key concepts and relationships even to young children. Examples, illustrating the essential contours of these maps, appear in Figure 5.01.

Sowa (1991a) explains however that such concept maps, unlike the equally diagrammatic conceptual graphs, cannot express all of logic and natural language for serious analysis. Given Sowa’s argument, the tough problem domain of business strategy would certainly require such levels of analysis. There is a particular significance in the above psychological basis for strategic problems. This is because, as already identified, the model must also enable strategic management accountants to build and review their own knowledge bases in view of the high level of continually shifting tacit knowledge that strategic problems embody. Together with their advanced technical power, conceptual graphs’ above argued user-readability should best enable the domain expert to integrate with knowledge-based systems beyond the checklist level. The following discussions expound upon the dimensions of this relationship.
To start with, consider the following factual illustration. During the formulation of Polovina and Delugach (1993), its authors debated the choice whether to employ ‘quantity’ or ‘units’ in that paper’s conceptual graphs modelling ‘supply and demand’. These same graphs appeared in the early part of Chapter 4, where supply and demand was also dealt with. It was discovered that a case could be made for either, merely according to individual inclination. As a result the authors decided that a deep semantic analysis of ‘quantity’ or ‘units’ outright mattered substantially less than what those terms symbolise psychologically to each user through his or her own conceptual graphs models. As corporate knowledge bases evolve there could nonetheless be some standardisation effort amongst the users to gain the benefits of wider scale knowledge. This would, as a consequence, involve standardising terms as well as graphs. From the arguments about strategic knowledge above, this would present a real challenge to those participants. Nevertheless, all would still be as agreed by a consensus of users, not as dictated by some well meaning knowledge engineer. In this way, the benefits of King and McAulay’s proposal about the benefits of an established academic library of knowledge may emerge.

Figure 5.01: Actual examples of Novak and Gowin’s concept maps, in outline (Source: Novak and Gowin (1984), pages 2 and 41 respectively).
Then there is the scope in conceptual graphs for hierarchical analysis, as already explained throughout this thesis so far. Although conceptual graphs are modular anyway, the use of the hierarchy allows even more compact graphs at a range of abstractions. As a result, the hierarchy should yield more user-readable structures. Further support for the success of hierarchical analysis is evidenced by the existing commercial software ‘IdeaFisher’\(^5\), which is essentially based on this form of analysis alone. However, the tricky nature of strategic knowledge may preclude sufficiently neat hierarchies. Thus how effectively the hierarchy aids user-readability will be of particular interest during the user evaluations of Chapter 6.

In Chapter 2 the accountant’s flowchart problem, allowing the accountant to draw badly structured diagrams, was discussed. Reference was made accordingly to standard program construction texts such as Stone and Cooke (1987). It will be found from such sources that the problem arises because those flowcharts are inherently procedural in nature. The declarative nature of conceptual graphs, however, causes the accountant to avoid drawing badly structured graphs automatically by simply sidestepping this obstacle. Furthermore the route of reasoning in the accountant’s flowchart has to start at the very beginning and stop at the very end. Should a given query fall exactly into this control pattern, then arguably the accountant’s flowchart can be seen to be satisfactory. Should the query, however, start or end somewhere in between then this procedural limitation becomes apparent. Hence the domain expert’s attempts at interpretation using the accountant’s flowchart with such scenarios can become extremely difficult. Again the indomitable nature of strategic knowledge is highly unlikely to be routed so narrowly\(^6\).

\(^5\) IdeaFisher runs on the Apple Macintosh, and is available from ‘Camelot’ (Tel: 0800 565656).

\(^6\) The flowchart is an algorithm. Kowalski (1979b) shows that “algorithm = logic + control’. Thus it is the control provided by the accountant’s flowchart that would be found wanting in respect of strategic knowledge.
Regarding the matter of inference, we saw that conceptual graphs can be more succinct than deploying if-then rules. As explained from Chapter 2, inferencing in conceptual graphs employs negative contexts, adapted by Sowa (1984) from the logic of Charles Sanders Peirce. As a reminder, the rule “If P then Q” could also be expressed as “not (P and not Q)”. This transformation may appear to be initially counter-intuitive to non-logicians, but Peirce relied on converting other logical relationships into AND and NOT form to obtain the visuality of his logic. To bridge the gap, Sowa (1984) proposed explicit relations in conceptual graphs such as one for ‘if-then’. Therefore in the linear notation:

\[ [P] \rightarrow \text{(implication)} \rightarrow [Q]. \]

Here the graph elements ‘\(\rightarrow\) (implication) \(\rightarrow\)’ denote the antecedent P and the consequent Q, hence ‘If P then Q’. In Peirce-based form the above implication essentially reads:

\[ \neg [[[P]] \neg [[[Q]]]]. \]

or, in the more convenient ‘(....)’ form discussed from Chapter 2:

\[ ([P] ([Q])). \]

As noted in Chapter 2, Farques et al. (1986) go further and dismiss Peirce logic completely, replacing this aspect of conceptual graphs with ‘if-then’ rules on the grounds that Peirce logic is unnecessarily complicated. However, as already shown, Peirce logic uncovers even the simple ‘modus tollens’ relationships obscured somewhat by the shallow procedural nature exhibited from chains of ‘if-then’ implications. We can further see that even Sowa’s ‘compromise’ form shown above does not truly help, yet such explications can be seen to be vital for modelling strategic knowledge.

Furthermore, as well as conceptual graphs basis in domain expert-familiar structured diagrams, the method of negation in Peirce is arguably similar to that in the accountant's bookkeeping model, where figures are negated by surrounding them in brackets.
For example the complementary double entry of ‘£3,000’ is ‘(£3,000)’ (Lee, 1986). The conceptual graphs inferencing advantage over if-then rules also reveals benefits in the conceptual graphs-based system’s output, because of the potentially greater number of ways a system could output an if-then based explanation than conceptual graphs. Thus there is less risk of such software replying in a sequence that the user might not expect.

Lastly, structured diagrams are further demonstrated by Hammer and Janes (1990) and Schwartz (1992) to be notably useful in the type of interactive human-computer integrated environment detailed above. The particular usefulness of conceptual graphs in that integrated environment are encouraged by Slagle, Gardiner and Han (1990), Loucopoulos and Champion (1990), and Champion (1991). In employing protocol analysis during the knowledge elicitation phase of their study, Slagle et al. note (page 37):

> While protocol analysis is difficult, conceptual structures\(^7\) offer a good mechanism to facilitate that analysis and create a knowledge specification.

Although we have mainly come across the above discussions beforehand in this thesis, these extra arguments have now focused and further underpinned the potential value of the conceptual graphs-based interface to the strategic management accountant. The remainder of this chapter details the interface’s design in computer software.

### 5.04 The Need for Conceptual Graphs Software

Although accepting that conceptual graphs should be software-based from all the arguments presented so far, it is briefly worth stating why this is so explicitly. The case really centres around whether it is sufficient simply to draw graphs manually and the human user seek insights merely by inspecting the result. After all, this is how the

\(^7\) i.e. conceptual graphs.
accountant’s flowchart and the many other graphical notations discussed in this thesis are employed in practice.

In answer to the above it can be seen that the particularly complex nature of strategic decision making causes this problem domain to lie well beyond human cognitive limits, as explained in Chapter 1. Thus a software knowledge-based system tool would be highly beneficial to the strategic management accountant. Apart from the strong evidence suggesting that this tool should be conceptual-graph based, manually having to convert any notation into machine-comprehensible form can only serve to introduce a delay that can significantly undermine the human-computer integration requirement stressed earlier. This is because, as Murray and McDaid (1993) indicate, in the visually-based environment the system and the user would work together in real-time for integration to be ultimately effective.

5.05 Existing Conceptual Graph Software, e.g. CAMES

Now that the need for conceptual graphs in software has been set out, attempts were first made to seek out any such existing suitable software. As The Department of Computer Studies, Loughborough University, had an active group working on this very software, it was decided that the most sensible search should be one that started locally. Hence the existing conceptual graph processor, written in prolog, of Smith (1988) and then Heaton (1989) was first examined. This software was simply called ‘cgp’, which stands for ‘conceptual graphs processor’. However cgp was dismissed on the fundamental grounds that the user had to instruct the software as to which graph operations actually needed performing. For instance, the domain expert would have to tell cgp that graph ‘a’ is to maximally join with graph ‘b’. Then the result, graph ‘c’, would be instructed by the expert to try being projected into by graph ‘d’. Should graph ‘d’ have successfully projected, then ‘d’ would be told by the expert to deiterate from the compound graph ‘e’. Then the expert would tell the result ‘f’ to double negate itself to assert the result ‘g’. Clearly this represented a too-high level of computational overload on the domain expert, as established above. In the instance, the expert is
merely interested in the domain knowledge of whether ‘g’ can be asserted from the the graphs ‘a’ and ‘b’ given ‘e’. It is up to the software, not the human user, to determine the computational knowledge remainder.

The conceptual graphs-based expert system CAMES, which stands for Client Admin Expert System, was then investigated (Smith, 1991). This expert system, a subsequent incarnation of Smith’s work on cgp above, was in commercial use at the blue-chip news company ‘Reuters’. As part of its activities, Reuters supplies financial information to dealers, banks and brokers worldwide. Reuters also market to their subscribers complete trading room systems and interfaces to Reuters’ own computers, as well as services and equipment supplied by various competitors. It is in this role that CAMES is employed in that it checks, when those subscribers order further equipment, that the new kit is properly compatible with their existing setup.\(^8\)

However, CAMES did not offer the full facilities of conceptual graphs theory. Specifically, it excluded the Peirce-based inferencing. Instead it relied on refutation which, as implemented in CAMES, meant that if a false graph could project onto a graph representing a subscriber’s setup with the new equipment then that setup would be rejected as invalid. As we know from Chapter 2, if any graph is false then automatically so are all its specialisations. We saw that if it is false that Clyde is an elephant, then anything involving Clyde the elephant will also be false. For instance, given that falsity, then ‘Clyde is an elephant and works in a circus’ must also be false. Hence small false graphs can falsify any larger graph that contains that false small graph as a generalisation. In CAMES small false graphs, called ‘schematic rules’, are projected onto potentially vast subscriber’s resulting configuration graphs. The alternative of checking for just valid configurations would involve potentially enormous projections, as all of the client’s graphs would have to project into ranges of true

\(^8\) Thus it is somewhat like the expert system ‘R1’ that configured VAX computers (McDermott, 1982).
configuration graphs. Therefore CAMES can be seen to be computationally efficient.

To illustrate the above, a Reuters marketing policy might be that “No two products shall supply the same service”\(^9\). This stipulation ensures that a subscriber does not pay for two products that would only supply the same service. The schematic rule reflecting this is:

\[
\text{[Service]} - (\text{support}) \gets [\text{Product: } *a] \\
\text{[Product: } *a] - (\text{support}) \gets [\text{Product: } *b].
\]

Now given the subscriber graph:

\[
\text{[Subscriber: UK00001]} - (\text{has}) \rightarrow [\text{MONR+FF: } #1] \\
\text{[MONR+FF: } #1] - (\text{support}) \rightarrow [\text{Money-Rates: GGMONR}] \\
[\text{Money-Rates: GGMONR}] - (\text{support}) \rightarrow [\text{Financial-Futures: CGFF}], \\
\text{[Subscriber: UK00001]} - (\text{has}) \rightarrow [\text{FFA: } #2] \\
[\text{FFA: } #2] - (\text{support}) \rightarrow [\text{Financial-Futures: CGFF}].
\]

and the appropriate hierarchical relationships, the above stipulation would successfully project into the client graph and result in:

\[
\text{[Financial-Futures: CGFF]} - (\text{support}) \gets [\text{MONR+FF: } #1] \\
[\text{MONR+FF: } #1] - (\text{support}) \gets [\text{FFA: } #2].
\]

Thus it can be seen that ‘FFA: #2’ is a superfluous piece of equipment. In Peirce-based form the equivalent would be:

\[
([\text{Service}]} - (\text{support}) \gets [\text{Product: } *a] \\
[\text{Product: } *a] - (\text{support}) \gets [\text{Product: } *b]).
\]

giving:

\[9\] The illustration is adapted from Smith (1991).
Chapter 5

([Financial-Futures: CGFF]-
  (support) <- [MONR+FF: #1]
  (support) <- [FFA: #2]).

and thereby:

([Subscriber: UK00001]-
  (has) -> [MONR+FF: #1]
  (support) -> [Money-Rates: GGMONR]
  (support) -> [Financial-Futures: CGFF],
  (has) -> [FFA: #2]-
  (support) -> [Financial-Futures: CGFF]).

From all the above, it may seem that CAMES appears more attractive than rule-based systems. However, CAMES still does not sufficiently encode the human expert’s knowledge. This is because CAMES assumes that any configuration is true by default. In the complex domain of business strategy this assumption might simply be too rash. Even stating the default as unknown could be expected to offer no help. The strategic management accountant may likely want to distinguish between truth, falsity and unknown. Where CAMES is placed evidences its limitation in the strategic sphere, as it can be seen to be in one of those conveniently well-defined problem domains criticised in Chapter 1. Added to these significant doubts were proprietorship obstacles in obtaining CAMES, as it was Reuters’ own in-house software. In view of all these concerns, it was decided to reject the CAMES expert system.

Surprisingly there was little else that could be considered as suitable, even further afield. However during the course of the above efforts I developed a close working relationship with a fellow researcher, John Heaton, who had enhanced cgp as mentioned above (Heaton, 1989). John was now devising a new cgp as part of his own PhD research in the Computer Studies Department, Loughborough University. That software looked more promising and, because of my

10 Excepting for any conceptual graphs processor that was being kept completely secret, this state of affairs was confirmed by events at the conceptual graphs workshop user evaluation that appears in the next chapter.
close contact with John, there was the opportunity to exercise some influence in respect of that new cgp’s design.

5.06 JEHCGP: A More Promising Conceptual Graph Processor

To begin with, the new cgp software still retained the name ‘cgp’. It was decided, however, that for this thesis’ purposes the new software should be referred as something that distinguished it from the earlier cgp. Hence I gave it the working title ‘JEHCGP’. This term was arrived at by simply prefixing cgp with John Heaton’s initials.

JEHCGP had automated the computational overload that cgp had suffered from, as discussed above. The user merely told and asked JEHCGP in terms of conceptual graphs. Upon the entry of each graph, JEHCGP evaluated it. Apart from syntactic checks, in essence JEHCGP compared the newly entered graph with those graphs in its knowledge-base. For user-asserted graphs, JEHCGP would report that either the graph was unknown to it or that it thought the opposite was true. In the latter case the knowledge-base had determined that a true graph was false or vice versa\(^\text{11}\). For a user-queried graph JEHCGP would explain why it was true or false. Alternatively if the graph could not be determined as either true or false, the software would reply that the graph was simply unknown. All in all, JEHCGP therefore offered solutions to the issues raised throughout this chapter.

Although the details of JEHCGP remained unpublished for some while, its full functionality eventually appeared in Heaton (1992). JEHCGP’s

\(^{11}\) To state that the knowledge-base, rather than the inference engine, determined the result may be seen as conflicting with the traditional model of expert systems. This statement, however, is deliberate. It has already been shown that conceptual graphs render the distinction between facts and rules as artificial. Hence, by extension, so can be the distinction between the knowledge-base and the inference engine. As far as the user is concerned it is the very graphs’ structure that causes the reasoning. Thus yet another aspect of computational overload is made transparent to the user.
various functions are further discussed, as appropriate, throughout the remainder of this thesis. For now, JEHCGP’s essential commands can be illustrated by:

!graph. Add graph to the knowledge-base.

?graph. Query the truth of graph in the knowledge-base.

~graph. Retract graph from the knowledge-base. Omitting a graph retracts the whole knowledge-base\textsuperscript{12}.

<filename. Load a knowledge-base.

>filename. Save the current state of the knowledge-base.

*. List the current state of the knowledge-base.

^. Quit JEHCGP.

Examples of commands with graphs are:

!['central office': Leeds].

!([‘central office’: *x]; ([office: *x]-
 (characteristic)-›[‘higher purchase cost’])).

?[‘improved service level’].

?([‘improved service level’]).

~[‘central office’: Leeds].

~([‘central office’: *x];
 ([office: *x]-
 (characteristic)-›[‘higher purchase cost’])).

\textsuperscript{12} Subsequently changed to ‘-graph.’.
Note that, in the above, the more convenient ‘\( (\ldots) \)’ are used in place of ‘\( \neg [\ldots] \)’ to denote negative contexts in the linear form. To see how far JEHCGP would be usable by the domain expert, a pilot study was conducted. The details, which also elaborate on the above, and outcome of that study are discussed next\(^{13}\).

5.07 Events at a Usability Pilot Study Based on JEHCGP

This pilot study examined the usability of conceptual graphs employing JEHCGP, which at that time was implemented on Loughborough University Computer Studies Department’s Hewlett-Packard UNIX computer known as ‘indigo’. The pilot study was conducted on the two days March 11, 1992 – March 12, 1992. The study’s subjects were Susan Heggie, Chris Hinde and Chris Messom.

The subjects were given and asked to understand the paper “Enriching Cognitive Mapping: A Technical Comparison between COPE and Conceptual Graphs based on an ‘Office Location’ example” (See Appendix A/01) and the introductory instruction sheet “Pilot Study based on the attached ‘Office Location’ paper” (See Appendix A/02) the day before (on the 11th).

At the start of the study itself (on the 12th), the subjects were given the paper “Conceptual Graphs Pilot Study: Instructions and Tutorial” (See Appendix A/03) and asked to work through it using JEHCGP.

The results were:

1) All the subjects managed to get to grips with the problem and the way it was modelled in conceptual graphs. They were able to both appreciate the deficiencies in the COPE, and find ways of improving the existing conceptual graphs knowledge-base using conceptual graphs.

2) However all the subjects expressed severe reservations about the quality of the interface itself. In particular JEHCGP’s interface,

\(^{13}\) That discussion is also duplicated in Polovina (1992b).
which could only handle the linear form of conceptual graphs, came in for harsh criticism. Essentially the subjects felt the linear form was too difficult to follow. The subjects agreed the display form of conceptual graphs, in a suitable WIMPS environment, would be a great improvement. They felt that the results would be much more meaningful if this study was re-conducted using the display form throughout instead.

3) One subject criticised the ‘UNKNOWN’ outcome of certain queries. This arose from the fact that cgp would produce this same outcome whether the graph a) existed in the knowledge-base but could not be proved, or b) did not exist in the knowledge-base at all. For instance, given that neither a local or central office was chosen beforehand, the query ‘?['improved service level']’, which existed but was not provable, would produce the same ‘UNKNOWN’ answer as the query ‘?['office site']’, which is totally absent from the knowledge-base. The subject felt a decision-maker would be asking ‘what do I do now?’. That subject felt a more interactive response than merely ‘UNKNOWN’ was needed, to include looking for similarly spelt concepts.

Therefore it became evident that, to test properly the usability of conceptual graphs, a graphical interface in an interactive WIMPS environment would be required. One subject suggested that the problem presented for the study was sufficiently trivial that it need not be modelled in conceptual graphs but in prolog alone. However such triviality was necessary to commence the study at a reasonably user-understandable level from which further details could be built. Moreover the ultimate purpose underlying this study is to evaluate if the familiarity of pictorially-based structured diagrams respected by business information professionals such as accountants (See, for example, Sizer 1989 and Woolf 1990) can be combined with the power of conceptual graphs. Overall, the work was therefore worth continuing.

Initially, as John Heaton was also at the above session as a silent observer, he subsequently ameliorated the above arising ‘UNKNOWN’ problem by enhancing JEHCGP so that it would explicitly output why
the query graph was unknown. The output now showed the graphs that would be needed to prove the query. Hence the above query ‘?['improved service level']’ would now not produce the same ‘UNKNOWN’ answer as the query ‘?['office site']’. In the latter case JEHCGP would repeat the graph back to the user, thus stating that it had no knowledge about that graph whatsoever. In the former case it would show graphs for both a local or central office, thus depicting that JEHCGP would need to know about the existence of these offices beforehand.

The other issues about ‘similarly spelt’ concepts, more meaningful examples, and the graphical form were to be examined thoroughly during the remainder of this thesis. In the next section, a methodology to aid significantly in these tasks is proposed.

5.08 The Conceptual Analysis and Review Environment, CARE

Given all the matters raised in this thesis up to this point, a human-computer integrated framework called Conceptual Analysis and Review Environment, or CARE, is proposed as the most suitable means by which conceptual graphs can be employed by strategic management accountants. Through the software-based CARE approach, the strategic management accountant could integrate with a computer knowledge-based system. For the reasons already identified, CARE should be conceptual graph-based. In summary, these reasons were:

a) we identified that strategic management accountants work with a particularly involved highly qualitative problem domain that requires a technically advanced knowledge-based tool.

b) the advanced technical tool specified in ‘a)’ above must also be user-comprehensible. Thus conceptual graphs could be particularly suitable because:

i) business information professionals such as accountants use structured diagrams in the general course of their work
ii) negative contexts happen to be similar to the way the accountant's bookkeeping model negates numeric values by enclosing them within rounded brackets.

It was discovered from the pilot study above that the linear form of conceptual graphs was too abstruse for users. The thereby discovered ‘command line’ usability limitations exemplified by the linear form augments the reservations of Schneiderman (1987), who asserts the considerable superiority of employing graphical human-computer interfaces instead. The user’s dislike of the linear form, as compared to the graphical display form, of conceptual graphs was also identified in the findings of Slagle, Gardiner and Han (1990). Hence CARE should employ conceptual graphs in their graphical form.

In addition, points of the theory itself must be as clear as possible. Ideally, as expressed throughout this thesis, intricate parts of theory should be handled by the machine and be transparent to the domain expert without making the power of any CARE software too trivial. Allied to this, and in line with a study by Reason (1990), the CARE software should prevent the domain expert from constructing graphs that have any incorrect syntax.

Through the very spirit of human-computer integration, CARE itself should also be adaptive in the light of experiences gained from its exposure with the domain expert. The nature of such changes would be determined by whatever occurs during that interactive process. An example might be where a user models, say, that ‘A’ affects ‘B’, ‘B’ affects ‘C’, ‘C’ affects ‘D’, and ‘D’ affects ‘A’. In some cases this may be a valid recursion but in others it may not. Amey (1968) and Mepham (1981) discuss whether circular arguments occur within the management accounting domain of opportunity costing. Such dilemmas might become significant in conceptual graphs. This is because knowledge in conceptual graphs form can be generalised and specialised at many levels, thus rendering a recursion insufficiently obvious. Should this problem arise from user exposure, it may be appropriate to include some kind of user-warning aid.
Further to the single user-computer interaction scenario, CARE should allow integration to occur amongst many domain experts. A typical CARE modelling situation might begin with the initial graphs being drawn by one strategic management accountant. Subsequently, that expert’s results could then be consolidated with graphs drawn by other such accountants. CARE could be used to check for inconsistencies between the models. This would draw out differing opinions which could then be resolved to obtain a more comprehensive model. Such activity, as it expanded into capturing other problem domains of interest to the relevant organisation, would evolve into that organisation’s corporate knowledge-base. In this regard, CARE need not ultimately be restricted to generalising over strategic problem domains alone. Linking conceptual graphs to databases, spreadsheets, or other knowledge-based systems could be another avenue for CARE. As supported by Parsaye, Chignall, Khoshafian and Wong (1989), such computer-computer integration could significantly elucidate all manner of business problems.

As already stated, CARE was anticipated to be a direct end-user modelling tool without the domain expert requiring to refer matters constantly to the knowledge engineer. In practice, for the user evaluations of the following chapter, it could not be expected that the domain expert should be left completely alone with the CARE software. This is because those experts could not be reasonably asked to supply their valuable time simply to learn about CARE, as well as apply it to some appropriate domain problem. Therefore there would be on hand, for the purposes of the user evaluations at least, a human facilitator to assist in the rapid understanding of CARE by the domain expert. The cognitive mapping approach, as modelled by COPE and discussed in Chapter 3, also employs human facilitators in its practical implementations (Eden, 1991). Given that the facilitator would need to be intimately familiar with CARE, it would be incumbent upon myself to fulfil this role.

Lastly, CARE was about bringing conceptual graphs in their most clear yet candid form to the domain expert. It was therefore decided that facilities such as translating between graphs and natural language
would not be added to CARE. After all, CARE was ultimately being devised to test conceptual graph’s usability. That meant finding out about the direct interrelationship between conceptual graphs and strategic management accountants. As already suggested in this thesis, natural language could easily obfuscate this particular desire despite the view that conceptual graphs translate naturally to and from natural language (Sowa, 1991b).

5.09 The CARE Software, and its Design Issues

We now turn to how the above general bases of the CARE methodology are to be applied. As expected, a major part of the CARE methodology is its software implementation. Devising that software would require the above ideas to be more precisely defined. Hence this section not only discusses the technicalities of the program itself, but CARE’s actual design issues too.

The CARE software was written in prolog using ‘LPA MacProlog’, a prolog compiler for the ‘Apple Macintosh’ computer. This compiler was selected because it offered extensive graphical libraries that took advantage of the Macintosh’s acclaimed WIMPS interface. Furthermore CARE was to be designed so that it provided a human-computer interface for JEHCGP, which was also written in prolog. This was because JEHCGP was criticised essentially for its linear graphs interface alone in the pilot study above. Thus JEHCGP would basically be retained to perform all its conceptual graphs theory operations, as this aspect was generally liked by the pilot study’s participants. As JEHCGP was written in prolog too, it was natural to combine it with the prolog of the CARE interface to provide an arrangement that could be expeditiously prototyped and enhanced cyclically.

The CARE interface program is listed in Appendix A/04. As agreed, over-explicit references to confidential JEHCGP code have been edited

14 NB: Appendix A/04 shows the program code as at version ‘1.3’ rather than as it started, namely version 1.0. Version 1.3 was how the code ended up by the time that all the user evaluations of Chapter 6 were completed.
in Appendix A/04 accordingly. Appendix A/05 imparts a flavour as to the changes that were necessary to JEHCGP itself. As can be seen from Appendix A/05, the CARE interface code was essentially written as a module that was simply linked to JEHCGP. This avoided too many delicate changes being made to JEHCGP itself, thus assuring its integrity remained intact. Appendix A/06 illustrates how the interface, which will be discussed further below, looks. The interface is also shown as part of that discussion. It is not proposed to discuss the program clauses directly, as this low level of detail can be gleaned from Appendix A/04 and Appendix A/05\(^\text{15}\). Rather the operation of the program is described as it affects the user.

\(\text{Figure 5.02 : Example CARE screen (version 1.0).}\)

\(^{15}\) To provide a reference point, however, the program commences with the clauses under the ‘START’ heading. Each heading denotes a MacProlog program window. Hence ‘START’ depicts such a window, ‘BROWSING’ and ‘INPUT’ are others. Further details about LPA MacProlog can be obtained from the programmer’s manuals or ‘Logic Programming Associates’ (LPA) directly (Tel: 081 871 2016, Email: lpa@cix.compulink.co.uk).
The CARE software, meaning the combined interface and JEHCGP engine, is fired up by mouse double clicking on its icon. As it was being devised as a user evaluation prototype, the software was not a ‘stand alone’ program. Hence the LPA MacProlog application files also needed to be present. Once the CARE software is loaded, the user is presented with the screen as shown by Figure 5.02. This figure also appears as Figure A/06.01 in Appendix A/06. The only differences between the start up window and Figure 5.02 are that the ‘offices leads to employees’ window contains no graphs, and that window is entitled ‘untitled drawing sheet 1’ instead. This shows that CARE currently contains no conceptual graphs. It is upon such windows, or ‘drawing sheets’ that graphs are drawn by the user. The functionality of CARE is enacted through the ‘tool pane’ and the ‘menu bar’. The drawing sheet contains the tool pane along its left hand side, whilst the menu bar resides at the very top of the screen. In line with the standard WIMPS interface of the Macintosh, the drawing sheet windows can be scrolled and re-sized. The tool pane operates as follows:

Arrow Tool (shown activated). The arrow tools functions are as follows. To begin with, the arrow tool selects a conceptual graph drawing by the user dragging a marqui over it as required. The arrow tool also drags a selected drawing by positioning the cursor over the upper left-hand corner of the appropriate concept, relation or outer negative context and then holding down the mouse button. A ‘hand’ cursor appears together with a rectangular ‘outline’ of the entire selected drawings to show they are being dragged. To prevent any confusion, graphs cannot be dragged such that they remain on top of other graphs. Hence graphs that are so dragged simply return to their original location\(^\text{16}\). Such tacit error checking, that is without

\(^{16}\) This alternative was decided upon, rather than positioning the graph in the nearest ‘free space’ for example, because it was the simplest to program without creating conceptual graphs syntax difficulties for the user. This was deemed acceptable as the user evaluations discussed in Chapter 6 could also be
an explicit message, also feature elsewhere in the CARE software. In these instances, this approach was decided to be the most succinct method of illustrating the error to the user. Above selected drawings can be cut, copied or cleared via the 'Edit' menu. Incomplete graph elements, such as a selected relation attached to two unselected concepts, cannot be cut or cleared as this would leave a syntactically incorrect graph. This time an error is signalled, as the user would be less certain about what had precisely happened with the drawing. The cut error message is “Sorry, but nothing cut because at least one incomplete structure was selected.” The clear message is the same except that the term ‘cleared’ is substituted for the term ‘cut’.

Eraser Tool. The user erases a drawing with this tool by positioning the cursor over that drawing, in the upper left-hand corner if appropriate, and clicking once. Should an incomplete graph element be erased, its remaining elements are also erased. Again, this prevents incorrect syntax graphs remaining on the drawing sheet.

Concept Edit Tool. Creates a new concept by positioning the cursor on the desired starting point on the drawing pane and, by holding down the mouse button, dragging a marqui to an estimated size of the concept. Then the appropriate text string is typed in. Once complete, a click anywhere outside the concept automatically re-sizes the concept box to fit around the text string neatly. Previously relied upon to highlight any deficiency in this respect, and thus require adoption of the appropriate alternative.

17 Again this choice offered the simplest, yet syntactically uncompromising, option in programming terms prior to user evaluation outcomes revealing the contrary. Another alternative, for instance, could have been to cut or clear all complete structures and leave the incomplete ones behind.
created concepts can be edited by clicking within the upper left-hand of that concept. Any links, such as relation links or negative contexts, attached to that concept are automatically erased. This ensures that, as the edited graph is now essentially a new one, a conscious decision is made by the user as to whether those original links were still valid.

Relation Edit Tool. Performs for relations the same as the concept edit tool above.

Directed Link Tool. Creates a new directed link between concepts and relations by positioning the cursor on the upper left-hand of the first concept and, by holding down the mouse button, dragging a line to the upper left-hand of the relation. Then the mouse button is released and a second line is dragged to the upper left-hand of the second concept. Tacit error checks, in that no link is drawn as a result, prevent these links from being drawn wrongly by the user. For instance, directed links cannot be drawn i) from one concept directly to another without a relation in between; ii) between relations alone.

Negative Context Tool. Creates a negative context by dragging a marqui over a drawing as required. This procedure is illustrated by Appendix A/06. The context is not drawn if it only partially covers a graph, or if it surrounds an incomplete graph element such as an unattached relation. Thus syntactically incorrect conceptual graphs are avoided, in the tacit manner discussed earlier.
Structure Assert Tool . Asserts a conceptual graph drawing into the knowledge-base of JEHCGP by dragging a marqui over a drawing as required. Like the negative context tool above, a partially covered or incomplete graph is not asserted. Unlike the tacit error handling of the negative context, however, the user receives an error message. The message, “Sorry, but nothing entered because at least one incomplete structure was selected.”, thereby removes any ambiguity in the user’s mind with respect to what may or may not have entered into the knowledge-base.

The text string of attempted concept or relation assertions is also checked for syntax errors. Basically this involves existence of too many ‘colons’ in that string. For example, JEHCGP could not determine where the type label ended and the referent begins in the concept:

Management Accountant: Strategic: S. Fred

Hence, on attempting to assert this graph, the error message “Sorry, but nothing entered - a concept had more than one colon (:)!” would occur. Similarly, as the accepted form of conceptual graphs does not support type labels and referents in relations, the error message “Sorry, but nothing entered - a relation incorporated a colon (:)!” is generated. Thus the user is not erroneously lead to believe that relations can be so structured.

When asserted graphs successfully enter JEHCGP, the user is presented with the screen as shown by Figure 5.03 . This figure also appears as Figure A/06.02 in Appendix A/06. This figure shows that JEHCGP was unaware of this graph, so it adds the graph to its knowledge-base. Alternatively, JEHCGP could have responded that it knew the graph already. This would happen when JEHCGP contained the graph already or could derive it by inference. If JEHCGP determined that the opposite was true, it would state this accordingly. In all these other cases, the new graph would not be added to the knowledge-base.
Chapter 5

Figure 5.03: Output from asserting example graphs into knowledge-base.

Note that the output window, ‘cgp - status report’, retains the linear form of conceptual graphs. Whereas the user could simply layout his or her graph drawings to taste and then input them into JEHCGP, the programming involved so that CARE could reply in the display form was seen to be very complex. Furthermore, LPA MacProlog would not permit graphical objects and text to appear in the same window. As a result there would be a real danger that CARE would have to output too many windows on the screen for the user to comprehend. Therefore it was decided to keep JEHCGP’s linear based output, although it was evidenced earlier that the graphical display form was preferable. The effect of this decision could expect to be confirmed or denied by events at the subsequent user evaluations in Chapter 6, and the appropriate modifications made to CARE accordingly.

The way generic referent’s are handled should also be noted from the JEHCGP output, in that JEHCGP changes these ‘*<whatever>’ referents into ‘*’ prefixed sequential numbers. This is to prevent the
erroneous interpretation of such referents, as explained in Chapter 2. In essence, any graphs:

a) having the same generic referent in more than one concept and

b) input simultaneously from the same drawing sheet

are determined to be coreferent. JEHCGP therefore gives them the same sequential number. Otherwise they are given different numeric generic referent markers signifying they are not coreferent. The user is thereby freed from having to worry about the indistinct nature of generic referents. That user would simply need to be aware that suitably marked coreferent concepts are input simultaneously from the same drawing sheet. Those graphs that were not so input would be regarded as non-coreferent even if they happened to share the same generic referent marker\textsuperscript{18}.

A double click on the ‘!’ tool beforehand prevents the graph from being evaluated until the old knowledge is overridden in the ‘CGP’ menu by the menu command ‘Override old knowledge...’. This matter is discussed later.

Structure Query Tool. Queries a graph in the knowledge-base by dragging a marqui over a drawing as required. The actions are similar to the ‘!’ tool above, excepting that the query graphs, by definition, are not added to the knowledge-base. As discussed earlier, JEHCGP responds with an explanation showing the graph to be true, false or unknown. If there is more than one explanation, the user is prompted with a ‘yes/no’ dialog box requesting whether another explanation is required. Clicking on ‘yes’ displays the other explanation until all the explanations are exhausted or ‘no’ is selected. Double clicking on the tool lists the entire knowledge-base.

\textsuperscript{18} The practical difficulties of generic referents are not peculiar to conceptual graphs theory, as evidenced by Pilote (1989).
Structure Retract Tool. Retracts a graph in the knowledge-base by dragging a marqui over a drawing as required. The actions follow the pattern of the ‘!’ and ‘?’ commands above. The graph is not actually retracted until the old knowledge is overridden by ‘Override old knowledge...’ in the 'CGP' menu as discussed below. Doubleclicking on this tool retracts immediately all that is held in the entire knowledge-base, leaving it empty.

Graphic Information Tool. Shows the internal prolog-based form of a drawing. This was of relevance more to CARE’s program maintenance than domain expert use.

The ‘mini viewing pane’ at the bottom left hand corner of the drawing sheet enables the user to move the visible part of the drawing sheet more quickly than through using the scroll bars. The scroll bars use effectively cause the Macintosh to redraw the graphs continually, and thus slow down the visible area movement significantly.

We now move onto discussing CARE’s menu items, other than where they a) do not differ from the usual Macintosh menu functions, b) are essentially irrelevant LPA MacProlog menu functions, or c) have been mentioned already. Each menu item also supported a standard Macintosh ‘⌘’ ‘hot key’ equivalent in line with good human-computer interaction practice. The menu items are as follows:

Under the ‘File’ menu:

Defaults...: The ‘Graphics’ option should be in ‘MacProlog’ format for graphs to be properly pasted to another drawing sheet.
Quit: Quits CARE. The user is asked if he or she wishes to save the current state of the drawing sheets and knowledge-base. If yes, the ‘Save the current state...’ command, under the ‘CARE’ menu and discussed below, is enacted before the program is quitted.

Under the ‘Edit’ menu:

Paste: When pasting a conceptual graph drawing where the front drawing sheet already contains a graph, a new untitled drawing sheet is automatically generated to take the pasted drawing. Thus the user is discouraged from drawing unnecessarily large graphs. This procedure is meant to ensure the user-readability of the graphs, in accordance with Miller’s seminal study which showed that a human can generally handle only a maximum of between five and nine concepts in his or her mind at any one time (Miller 1956).

Balance: In a linear output graph, by positioning the cursor within that structure and enacting this function, the area between the graph’s matching brackets will be highlighted so enabling their scope to be seen more easily. The ‘Balance’ command was included to help overcome any user difficulty with the residual linear element in CARE, and the command’s effect is expected to be determined from the user evaluation sessions. Appendix B/13 illustrates the ‘Balance’ command in use from the final user evaluation session discussed in Chapter 6.

Under the ‘Window’ menu:

Select Window...: Allows a particular window to be selected.

Under the ‘CARE’ menu19:

19 As a tidying up exercise, some of these menu items were subsequently rearranged either within the same menu bar heading or moved to under the ‘CGP’ menu. Appendix A/04 shows their final position. There should be no difficulty in finding the appropriate code under the ‘MENU HANDLING’ program window of that appendix.
About CARE...: This displays details about the CARE software, such as copyright, trademarks, version number and acknowledgements.

New drawing sheet: Creates a new empty drawing sheet, giving it the title “untitled drawing sheet <n>” where ‘n’ signifies a sequential but previously unused number to distinguish it from other drawing sheet windows. Each drawing sheet contains its own tool bar. Clicking in the small white box at the top left hand corner of the drawing sheet causes the drawing sheet to disappear, but it is merely hidden. Thus it and any graphs it contains can be made visible through the ‘Select Window...’ command under ‘Window’ above. To completely remove a drawing sheet and its contents, ‘Kill front drawing sheet’ is used below.

Rename front drawing sheet...: Renames the title of a drawing sheet window. For instance, the window ‘untitled drawing sheet 1’ could be renamed ‘offices leads to employees’. A renamed drawing sheet name that coincides with an existing drawing sheet name is suffixed by a ‘•’, to avoid window conflict problems. The user thus has the ability to name drawing sheets in familiar phrases.

Kill front drawing sheet: The front drawing sheet, and any graphs it contains, are completely deleted.

Open a previous state...: This opens a previously saved set of drawing sheets and knowledge-base, and also clears the prevailing knowledge-base. Any existing drawing sheets are, however, left untouched. Clearing the present knowledge-base and replacing it with the file-saved one was a feature of JEHCGP. It was decided that this should also occur with CARE, as it would prevent the user from being confused as to what lay in the knowledge-base. Like ‘Rename front drawing sheet’ above, window conflicts are avoided by the suffix ‘•’.

Save the current state...: This saves the current state of the drawing sheets, hidden and visible, and knowledge base, according to a file name and location essentially of the user’s choosing. Both this
command and ‘Open a previous state...’ above operate via standard Macintosh file open/save dialogues.

Find a text string...: Searches the output of ‘cgp - status report’ and then the drawing sheets for a user defined text string, but does not search the internal knowledge-base. Therefore, if required, the knowledge-base needs to be listed onto the output beforehand by double clicking on the '? tool above. For the output part, any existence of the string is highlighted from which the user can stop or go on to look for another instance. In the drawing sheets the string's concept or relation appears selected, together with a ‘yes/no’ dialog box. The user can continue the search by choosing ‘yes’ or stop it by clicking ‘no’. Should the string not be eventually found, the user is informed that “<String> could not be found. Sorry!”.

MacProlog menus: Shows, by being disabled ('greyed out') itself, that all LPA MacProlog menu items not relevant to the operation of CARE are disabled.

Under the ‘CGP’ menu:

Override old knowledge...: Each previously unknown structure added to, or retracted from, the knowledge-base is recorded as episode of knowledge. Unless a retraction, or a structure entered immediately after double clicking on the '!' tool, the structure is also evaluated. If the newly evaluated structure does not make the knowledge-base inconsistent then that structure is also added to the knowledge-base. Otherwise it remains an episode only. However the knowledge-base may be rebuilt to override its previous knowledge in favour of the later contradictory knowledge, together with retracted and double clicked '!' items, by enacting this menu item.

Lattice...: Replaces the JEHCGP command line equivalent for entering lattice relationships directly. Note that in JEHCGP, a lattice not only refers to the type hierarchy but a hierarchy of relations too. Although not in Sowa (1984), the addition of a relation hierarchy in conceptual graphs theory has met with general acceptance. Hence there was no valid reason why it should be excluded from CARE.
JEHCGP, however, displays these lattice relationships in a somewhat non-standard way. Namely, they are displayed as:

Subtype << Supertype. (Or Sub-relation << Super-relation.)

rather than:

Subtype < Supertype.

This was so for reasons inherent to the functioning of JEHCGP. Nonetheless, it was decided that the change was so trivial that there was no need to rectify this in CARE. Another interesting feature of JEHCGP was that hierarchical relationships could be conceptual graphs too. Thus:

Management Accountant << Accountant.

could be stated as:

[type:Management Accountant]<<[type:Accountant].

or

[type:Management Accountant]>>(type:Accountant).

The well argued theoretical basis for this approach can be seen in section 2.9 of Heaton (1992). Without repeating the technical issues here, it was decided that this would be a natural way for the domain expert to deal with hierarchical relationships. This was because the above step would incorporate the hierarchy into the remit of conceptual graphs themselves, and so further reduce the computational overload on that user. Although this was an extension to the accepted conceptual graphs theory, it was worthy of inclusion in CARE.

Type definitions were entered into JEHCGP by means of the ‘double implication’ described in Chapter 2. Relation definitions are also entered in a similar way. For example the graphs:
also describe that:

central office << office.

These alternative methods of describing hierarchical relationships could be entered into JEHCGP via the ‘!’, queried by ‘?’ or retracted by the ‘crossed-out !’ tool commands. Thus the user computationally overloading nature of the ‘Lattice...’ command was effectively rendered as redundant.

Conformity...: Replaces the JEHCGP command line equivalent for conformity relationships. As JEHCGP extracted these conformities in the course of its conceptual graph evaluations anyway, this command became redundant.

Execute a linear form script...: Replaces the JEHCGP command ‘%<file>.’. As this item read in linear form graphs, this facility would not be anticipated to form part of the user evaluations of Chapter 6. Hence it is not discussed further. The nature of this command is discussed in sections 2.8.2 and 3.4 of Heaton (1992).

Engage interactives: Replaces the JEHCGP command ‘%user.’. This command was not anticipated to be employed in the user evaluations. Hence, like ‘Execute a linear form script...’, the command is not discussed further. The command is discussed in section 4.3 of Heaton (1992).

Command line...: This facility executes the remaining JEHCGP commands in their native command line form. These commands, which are essentially not relevant to this thesis, can be seen in Heaton (1992).
Lastly, in all the above functionality and design issues of the CARE software, attention was given towards the criteria of Ravden and Johnson (1989). In particular, their study of practically evaluating human-computer interfaces identified the following nine top level attributes:

- visual clarity
- consistency
- compatibility
- informative feedback
- explicitness
- appropriate functionality
- flexibility and control
- error prevention and correction
- user guidance and support

As stated by Sage (1991), the definition and meaning of these attributes can be inferred from the attributes themselves. Therefore it should be evident that CARE has primarily met the above criteria. The actual extent of conceptual graphs’ achievement, through CARE, along these attributes can expect to be tested in the user evaluations of the following chapter.

5.10 Concluding Remarks

The critical limitations of transferring knowledge between the human domain expert and the computer are well known. Added to this obstacle, the strategic problem domain is known to be particularly complex. Conceptual graphs offer a potential means of tackling this impasse. This is because conceptual graphs are an advanced knowledge-based technique yet contain acute similarities with existing methods, namely structured diagrams and bookkeeping negation, respected by the accountant. From this user-familiar base, CARE, which stands for conceptual analysis and review environment, presents conceptual graphs in a user-friendly methodology for strategic management accountants. Furthermore the advanced knowledge-based decision support software tool that embodies CARE
incorporates sound human-computer integration principles. Thus, through CARE, these experts might be able to employ conceptual graphs in handling their intricate problem domain.

At this point, it is appropriate to confirm all the findings discussed in this thesis so far through user evaluations. This is the subject of the next chapter, that now follows. That chapter evaluates conceptual graphs amongst domain experts who would be adequately representative of the strategic management accountant.