(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 April 2007 (26.04.2007)

PCT

(10) International Publication Number
WO 2007/045013 A1

(51) International Patent Classification:
*G06F 7/00* (2006.01) *G06F 17/18* (2006.01)

(21) International Application Number:
PCT/AU2006/001435

(22) International Filing Date: 3 October 2006 (03.10.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

| | | |
|---|---|---|
| 2005905708 | 17 October 2005 (17.10.2005) | AU |
| 2005222571 | 17 October 2005 (17.10.2005) | AU |
| 11/255,554 | 21 October 2005 (21.10.2005) | US |

(71) Applicant *(for all designated States except US)*: **MIDDLEMARCH HOLDINGS PTY LTD** [AU/AU]; 4th Floor, 313 La Trobe Street, Melbourne, Victoria 3000 (AU).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **SEIDEL, Roland, Geoffrey** [AU/AU]; 62 Temple Road, Selby, Victoria 3159 (AU). **CHANT, Dale, Morris** [AU/AU]; 33 Heath Avenue, Ferntree Gully, Victoria 3056 (AU).

(74) Agent: **SMOORENBURG PATENT & TRADE MARK ATTORNEYS**; PO Box 515, Ringwood, VIC 3134 (AU).

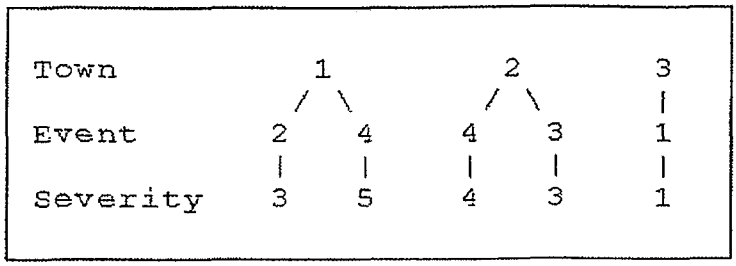
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**
— *with international search report*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

(54) **Title:** A METHOD AND APPARATUS FOR IMPROVED PROCESSING AND ANALYSIS OF COMPLEX HIERARCHIC DATA




(57) **Abstract:** The present invention relates to the field of data analysis. In one form, the invention relates to analysis of data in an analytical database. Preferably, the invention relates to analysis of complex coded data, in particular hierarchical data. A number of aspects of invention are disclosed, including, without limitation, the Storage of hierarchic data, a GUI representation of hierarchic data, hierarchic data convolution and devolution, cross tabulation of complex data, including a segment method, an offset method, a one-level method, and a segment matching method, and a grid construction generator for making hierarchic variables.

WO 2007/045013 A1

# A METHOD AND APPARATUS FOR IMPROVED PROCESSING AND ANALYSIS OF COMPLEX HIERARCHIC DATA

## FIELD OF INVENTION

The present invention relates to the field of data analysis.

In one form, the invention relates to analysis of data in an analytical database. Preferably, the invention relates to analysis of complex coded data, in particular hierarchical data, as often found in survey responses.

It will be convenient to hereinafter describe the invention in relation to hierarchical data, however it should be appreciated that the present invention is not limited to that use only.

## BACKGROUND ART

The discussion throughout this specification comes about due to the realisation of the inventors and/or the identification of certain prior art problems.

The inventors have realised that data which is, for example, representative of more real-life situations can be relatively complex. The existing prior art has difficulties in analysing more complex data. There are a number of techniques used in assigning numeric codes to pre-defined categories so that the process of tabulation can be reduced to counting the number of codes. Furthermore, filtering and weighting are employed in using tabulation as an analytical tool. Simple data is relatively well handled but the handling of complex data – multi-response, incremented and/or, in particular, hierarchical becomes quite difficult.

The inventors have realised that one cause of this difficulty is the nature of the data itself. Although various techniques have been used, they fail to address a fundamental problem in the complexity of the data.

Data complexity may be discussed with reference to simple data, multi-response, incremented, hierarchical, for example.

### Simple

For data such as Gender and Region, where the categories are mutually exclusive, the processing requirements for cross tabulation are relatively simple. It requires a count of the number of times each Gender code (such as 1=Females, 2=Males) and each Region code (such as 1=NE, 2=NW, 3=SE, 4=SW) occur in conjunction for a given case.

**Multi-response**

Data on weather events, however, might be coded as:

c1=rain

c2=hail

c3=snow

c4=wind

c5=heat

A town may have none or all of these so a record for one town may be blank and for another town, with reference to the code above, it may be 1;2;3;4. Yet another town that has had several episodes of hail in one day may record 2;2;4;2 with reference to the code above.

Cross tabulating multi-response data requires iterating over all possible pair combinations.

**Incremented**

Each event may have an increment or value associated with it like the amount of rain, the speed of wind. This may be recorded as 1*30; 4*55 to mean 30mm of rain and winds of 55 kph, using table 1 above. When tabulated, the specified increment for that instance of that code is added to the aggregate, instead of the default increment of 1.

**Hierarchic**

Complex data sets can often have a natural hierarchy. There are many examples:

- Doctor/Patients/Prescriptions
- Departments/Computers/Installed Software
- Field trials for a pharmaceutical, Laboratory/Trial#/Test Type/Result
- Market research of brand attributes ratings
- etc.

This sort of data is notoriously hard to analyse. Obvious sorts of questions a researcher might want answered, relative to the examples above, are:

- How many prescriptions does each doctor issue? As a percentage of the number of patients? How many patients have more than one

doctor? Of all issued prescriptions, what proportion are analgesics, antibiotics?

- What is the ratio between the number of computers and the number of installed applications? Which departments have the most spreadsheets? How many applications are installed on a given OS, etc.
- Which laboratories consistently get a pass for a particular test. Which don't? Which tests pass most often? Do the results for one trial differ substantially or significantly from others?
- For a given set of branded products, and a set of attributes, how is each brand rated? For subsets of brands? Is one attribute more/less popular across all brands than others?

Hierarchic data contains information at several levels. Recording the severity of weather events at many towns involves, for example, three levels of coding. In addition to the event codes, the towns could be coded as 1,2,3 etc. and severity of the weather could be coded as 1,2,3 etc. This data is often pictured as a tree or set of trees as illustrated in Figure 1.

It can be extrapolated that for 20 towns, 5 events and a severity scale of 10 there are potentially 1000 different data items to be recorded each day. Furthermore, data at each level of a hierarchic structure can itself be multi-response, incremented and/or simple uncoded quantities. To allow for multiple events of the same type usually involves multiplying the possibilities – 2000, 3000, 5000 – with tension arising between allowing for enough multiple events and not wasting too much space on data storage.

Hierarchic data for one case is essentially an N-node tree of any depth and complexity. Very few systems are considered to come close to storing this economically. RDBs (Relational DataBases) may use several linked tables, card image and other flat forms must provide space for every possible branch combination even though very few may be used.

Another difficulty is that, although commonly referred to as 'trees', what is really needed is a 'forest' – a collection of trees. For survey data, the root node is

often conceptual, comprising the variable itself. A common example in market research is brand/attribute/rating. For example:

Q12a. Please rate each of the following statements for each brand on a scale of 1 to 10, where '1' means 'do not agree', and '10' means 'agree very much'.

| | TimTams | Monte Carlo | Salada |
|---|---|---|---|
| Is a healthy product | | | |
| Good value for money | | | |
| Has an excellent reputation | | | |
| Available at many retail outlets | | | |

Table 1

For a single respondent, the grid could be filled out as:

| | TimTams | Monte Carlo | Salada |
|---|---|---|---|
| Is a healthy product | 2 | 1 | 4 |
| Good value for money | 4 | 7 | 10 |
| Has an excellent reputation | 9 | 6 | 8 |
| Available at many retail outlets | 10 | 8 | 7 |

Table 2

The tree representation, including the conceptual root, is illustrated in Figure 2. There are many well-establish algorithms for reading such a tree, but for cross tabulation none are considered entirely satisfactory.

**Cross tabulation**

A general problem identified by the inventors is that cross tabulation algorithms to address issues noted above, especially across an entire hierarchy, are considered relatively slow, clumsy, inefficient and generally inadequate. For cross tabulation, traversal speed is an important factor. Using the prior art methods of address pointers at each node to the child nodes, whether on disk or in RAM, can be CPU-intensive, and makes manually following the data chains through the tree for diagnostic and verification purposes cumbersome and difficult.

Relational Database (RDBM) systems in particular find it extremely difficult, if at all possible, to calculate a full set of possible percentages. Systems derived

from a survey processing tradition are generally somewhat a little better at handling hierarchic data, but are still considered to incur a severe performance penalty when used in conjunction with complex data.

In the prior art, preparation of hierarchic data for analysis by cross tabulation is usually addressed by one of three methods:

1. Split the data up into many parallel variables, where the total number of variables equals the product of the number of categories at each level in the logical hierarchy. A problem with this technique is that there may be hundreds or even thousands of variables, which will often be sparsely populated, requiring individual cross tabulations for each, and the specification of a query on all the data will require each variable to be cited in some way, which can be physically difficult to achieve reliably.
2. Flatten the hierarchy by mapping each of all possible code combinations to a unique code of a new variable, where the number of new codes required is the product of the number of categories at each level in the logical hierarchy. It is considered that this just shifts the problem from needing huge numbers of variables to needing huge numbers of codes with little improvement in waste of space or time.
3. Store each level of the hierarchy as a single variable, delimited in some way so that the codes can be appropriately matched across the hierarchy at cross tabulation time. It is considered that this reduces the space and time wastage but still requires unnecessary duplication (each level must replicate the structure of its neighbour) and leaves the levels, that are logically part of a whole, unlinked. This requires some effort of bookkeeping on the part of the user and opens the risk of making invalid or meaningless juxtapositions.

Another problem identified by the inventors is that analytical outputs across an entire hierarchy either cross tabulated in its own right or against another variable, which should be available easily and quickly, can take a long time to process, require a lot of manual checking, are hard to specify (could take many pages of SQL in the RDBM world) and are often hard to interpret.

**Representation of hierarchic data**

Representing conventional variables to a user is commonly done in a tree display showing the variable as a folder with its codes as children under the folder. Hierarchic data presents another problem identified by the inventors in that there is no conventional way to present them. Unravelling all possible pathways through a data tree may lead to a combinatorial explosion.

**Hierarchic data convolution and devolution**

The inventors have also found that in specifying tables with conventional data, there is a direct relationship between the codes in the specification and the rows and columns: every top code makes one column in the table and every side code makes one row in the table. Hierarchic variables present a problem because they are best represented as a tree structure for specification but the number of rows or columns is not simply related to the number of codes. A variable with three levels having 2, 2 and 5 codes respectively will generate 2x2x5 = 20 rows or columns. This is illustrated in Figure 3.

The matter is further complicated by filter and weight expressions that can appear as an unlimited chain of parents of any variable. Additionally, codeframes may have base expressions that need to be preserved and a flag indicating which codes are to be based. This information also needs to be stored in files as saved tables in a way that both the row/column nesting and the specification tree can be reconstructed.

**Grid construction generator for making hierarchic variables**

The inventors have realised that the use of hierarchic variables are considered to be a good way to analyse data but most data collection systems can't provide it. Usually the data is 'atomised'. Data on records for 20 towns of 5 classes of weather event at 10 severity levels might arrive as 20*5=100 different variables each with a 1-10 code frame or, at worst, 1000 different binary variables. Organising and representing this is possible with prior art construction techniques but is considered to be time consuming and difficult.

Any discussion of documents, devices, acts or knowledge in this specification is included to explain the context of the invention. It should not be taken as an admission that any of the material forms a part of the prior art base or

the common general knowledge in the relevant art in Australia or elsewhere on or before the priority date of the disclosure and claims herein.

An object of the present invention is to alleviate at least one disadvantage associated with the prior art.

Another object of the present invention is to enable data to be relatively transparent and/or relatively easier to present to an end user.

A still further object of the present invention is to make the direct cross-tabulation of hierarchical data possible, relatively fast, relatively straightforward and/or relatively reliable.

**SUMMARY OF INVENTION**

The present invention provides, in one aspect of invention, a data format and/or method of representing hierarchical data such as from a survey response, comprising a string of indicia, the string including indicators of tree depth (level).

The present invention provides, in a second aspect of invention, an analytical tool adapted to provide analysis based on data formatted as herein disclosed.

The present invention provides, in a third aspect of invention, a GUI representable data format and/or method of displaying hierarchical data, comprising at least one first folder, at least one second folder, the second folder being provided within the first folder, each second folder including code(s) related to a corresponding level of the hierarchy.

The present invention provides, in a fourth aspect of invention, a data structure representation and/or method of representing the structure of hierarchical data, comprising the steps of providing a first folder representing a variable, providing at least one second folder, within the first folder, each second folder representing a level, and providing within each second folder, codes for that level.

The present invention provides, in a fifth aspect of invention, a data representation and/or a method of converting data represented in a first format to data represented in a second format, comprising the step of using SRL in the process of converting the first format to the second format.

The present invention provides, in a sixth aspect of invention, a cross-table specification represented in SRL.

The present invention provides, in a seventh aspect of invention, a schema adapted to represent a cross table specification, comprising first indicia representing a variable and second indicia representing code(s).

The present invention provides, in an eighth aspect of invention, a specification representation language as herein disclosed.

The present invention provides, in a ninth aspect of invention, a method of processing data, the method comprising the steps of providing data representing a hierarchy having at least two levels, each level having at least one code and processing the data at each level as a single unit (segment).

The present invention provides, in a tenth aspect of invention, a method of determining a row or column in a table applicable to a given response having a complex data structure, the method comprising the steps of determining the response, determining the structure of the variable, and processing the structure arithmetically to determine the row or column for the response.

The present invention provides, in an eleventh aspect of invention, a method of processing a response, the method comprising the steps of determining the level, and processing a segment(s) only in that level.

The present invention provides, in a twelfth aspect of invention, a method of accommodating a variable by providing segments equivalent to the segments at that level.

The present invention provides, in a thirteenth aspect of invention, a method of arranging variables in a table configuration, the method comprising the steps of selecting the variable, providing a grid structure and slotting the variable into the grid at a desired location.

Other aspects and preferred aspects are disclosed in the specification and/or defined in the appended claims, forming a part of the description of the invention.

In essence, the present invention, with regard to the following aspects of Invention:

1. Storage of hierarchic data

   - Provides a way of storing complex data that is highly space efficient and facilitates rapid processing. What would otherwise be stored in many files with indexing links is stored in a single file, one case per

line. Multi-responses are separated by semi-colons. Increments are preceded by an asterisk. Hierarchic tree structure is indicated with alphas for levels.

2. GUI representation of hierarchic data

- Provides a way of representing hierarchic data that conveys an intuitive understanding of the structure while avoiding the combinatorial explosion of possibilities. The hierarchic structure, which would otherwise be inferred by linked tables or other devices requiring some effort to interpret, is indicated as a tree with Levels as child branches having their codes underneath.

3. Hierarchic data convolution and devolution

- Provides a manner of employing a Specification Representation Language that effectively mediates between the intuitive GUI representation of hierarchic variables, the combinatorial explosion of rows and columns on resulting tables and the storage and retrieval of this information in files.

4. Cross-tabulation of complex data

a) The segment method.

- Provides an approach to defining the unit of data that facilitates processing complex data. Rather than regard a single code as the unit, a segment is all the responses at one level point – effectively one node of a tree of data – and may be just a single code but could be many codes each with increments. The approach to the data is from a different perspective that allows economies of speed and simplicity. For example, by using a larger unit of data, the complexities of multi-response and incremented data have been found to be segregated and relatively easier to deal with.

b) The offset method.

- Provides a very fast arithmetic way of indexing hierarchical data elements that nonetheless allows for multi-responses at any level. This is much faster than following links across tables or similar devices required of other storage techniques and is possible because the data from all levels is stored in the one place.

c) The one-level method.

- Provides a method that increases the efficiency of processing hierarchical data by establishing a 'processing level' and ignoring data at any other level.

d) The segment matching method

- Provides a way of preparing responses from different variables to facilitate rapid processing. During processing, variables may contribute to the top, side, filter and weight components of a table specification. A relatively fast way to process the responses for a particular case is if all components have the same number of segments. This preparation arranges for that even if the four components are all hierarchic and all at different levels. This method enables expanding or collapsing of segments at one level to substantially match the number of segments at another level. The result is at each level, an array of substantially the same length, with data related by parallel indices rather than tree navigation. Effectively, it converts a filter and/or weight result at one level to the equivalent result at another level by referring to the tree structure implicit in the actual response string.

5. Grid construction generator for making hierarchic variables

- Provides a method for combining the combinatorial explosion of simple variables that logically form a structured hierarchic variable. In particular, an intuitive visual way of building the single hierarchic variable from a profusion of simple ones.

The present invention has been found to result in a number of advantages, such as:

- enables hierarchic structures comprising any mix of data types to be tabulated and cross-tabulated under most, if not any, filter and/or weighting condition in a way that is highly computationally efficient;
- produces cross tabulations that are considered functionally complete, in that all logical outputs with respect to any combination of one, some or all hierarchic levels, can be easily obtained, whether within the hierarchy itself, or within a cross tabulation of the complete hierarchy or any one of its levels by any other variable;
- can 'unloop on the fly' meaning many, even thousands, of tabulations can be reduced to one;
- hierarchic data that is often stored in many, even thousands, of variables can be stored in a reduced number, even one;
- specifications (including filtering, weighting and basing conditions) can be expansive;
- speed of processing is increased;
- it becomes reasonable to generate the entire hierarchic table where previously this was too cumbersome, and to process the table speedily given the various inventive methods as herein disclosed;
- economy of storage and specification is improved;
- the data is relatively simpler to handle and interpret;
- greater productivity is achieved with less knowledge required in using the present invention;
- fewer computational resources are required to use the present invention whilst still enabling the handling of relatively complex data; and
- hierarchic variables can be safely and easily assembled from a multitude of component variables.

Further scope of applicability of the present invention will become apparent from the detailed description given hereinafter. However, it should be understood that the detailed description and specific examples, while indicating preferred embodiments of the invention, are given by way of illustration only, since various

changes and modifications within the spirit and scope of the invention will become apparent to those skilled in the art from this detailed description.

**BRIEF DESCRIPTION OF THE DRAWINGS**

Further disclosure, objects, advantages and aspects of the present application may be better understood by those skilled in the relevant art by reference to the following description of preferred embodiments taken in conjunction with the accompanying drawings, which are given by way of illustration only, and thus are not limitative of the present invention, and in which:

Figure 1 illustrates a representation of data;

Figure 2 illustrates a tree representation of a market research survey response;

Figure 3 illustrates a data tree;

Figure 4 illustrates a hierarchical tree representation in accordance with an aspect of invention;

Figures 5, 6 and 7 illustrate how the SRL is used in accordance with an aspect of invention;

Figure 8a illustrates a process of convolution according to an aspect of invention;

Figure 8b illustrates a process of devolution according to an aspect of invention;

Figure 9 illustrates an offset method according to an aspect of invention;

Figures 10a and 10b illustrate a one-level method according to an aspect of invention;

Figures 11a, 11b and 11c illustrate the segment matching according to an aspect of invention;

Figure 12a and 12b illustrate filtering at different levels according to an aspect of invention;

Figure 13 illustrates an example of grid construction according to an aspect of invention;

Figures 14 to 19 illustrate grid construction associated with each row or column being one variable; and

Figures 20 to 23 illustrate grid construction associated with each cell being one variable.

## DETAILED DESCRIPTION

Broadly, there are a number of aspects of invention disclosed, at least some of which are:

1. Storage of hierarchic data;
2. GUI representation of hierarchic data;
3. Hierarchic data convolution and devolution;
4. Cross-tabulation of complex data
   a) The segment method;
   b) The offset method;
   c) The one-level method;
   d) The segment matching method.
5. Grid construction generator for making hierarchic variables.

### 1. Storage of hierarchic data

The data tree illustrated in Figure 1 has three individual trees, one for each day in a weather database. Figure 2 shows similar survey data with the three trees (one for each brand) as branches of a single question. In accordance with this aspect of invention, in general, tree depth indicators are used to store a forest of N-node trees in a string. Using alphas is a convenience, however, any indicia and/or format may be used. If more than 26 levels are needed, case sensitivity can be used to allow 52 levels. If more than 52 levels are needed, wide strings could be used (16 bit characters). If unlimited levels are required, then depth could be indicated by some such system as {1}...{2}...{3}...{4}... etc.

Thus, the information as illustrated in Figure 1 may be stored as a single string, namely:

a1b1c**2**b2c**4**b3c**9**b4c**10**a2b1c**1**b2c**7**b3c**6**b4c**8**a3b1c**4**b2c**10**b3c8b4c**7**

string 1

where

'a' indicates product (TimTams, Monte Carlo, Salada),

'b' indicates one of health, value, reputation, available, and

'c' indicates value (response answer).

The items of rating data may be shown in bold (as above). The three brands at the top level are indicated by a1....a2....a3. Within each brand the four

statements, at the second level, are indicated as a1b1...b2...b3...b4...a2..... Within each statement the actual ratings, at the third level, are indicated as

a1b1**c2**b2**c4**b3**c9**b4**c10**a2b1**c1**... etc. string 2

This representation of the data allows the entire tree to be traversed from left to right in a single pass.

The weather data from Figure 1 may be recorded or stored as:

a1b2c3b4c5a2b4c4b3c3a3b1c1 string 3

where the letters indicate level (a, b, c ...) and the numbers are the data at each node.

Multi-response data is easily accommodated by using a semicolon delimiter (or any other indicia), for example:

a1b2c3;5 string 4

shows town 1 had two event 2s, of severity 3 and 5.

Incremented or value data is easily accommodated by using a preceding asterisk (or any other indicia), namely:

a1b2c3*55;5*73... string 5

Thus string 5 illustrates town 1 had a weather event 2 with details 3 and 5 with associated measurements of 55 and 73, such as a storm (code 2 at level b) with 55 mm of rain (where rain is code 3 at level c) and 73 kph winds (where wind is code 5 at level c)

**2. Representation of hierarchic data**

There is a problem as noted above in representing a tree, such as (but for illustrative purposes only) the tree shown in Figure 3.

In accordance with this aspect of invention, we represent:

- a tree whose root folder represents the variable

- sub folders represents levels in order
- contents of sub folders are the codes at that level

Figure 5 illustrates this, in which a hierarchic variable is shown as a folder with sibling sub-folders for each of the levels, these sub-folders each have as children their own codes. This reflects the tree model for hierarchic data, implies that each level can be treated as an independent normal variable and assists in comprehending the structure of the data.

The GUI representation of Figure 4 is considered to fully describe the tree of Figure 3. Furthermore, the representation of Figure 4 gives access for specification purposes to the entire variable (the root), each of the three levels (.Brand, .Attribute, .Rating), and each of the 2*2*5=20 possible paths. The advantage of this representation is that, if the levels comprised 10 codes each, creating 1,000 possible paths, only 30 leaf nodes need to be displayed for user-selection purposes. The representation in accordance with this aspect of invention has zero redundancy. For example, in Figure 3, Attribute1 has to appear twice in the diagram – this is considered redundant. At the bottom level r1 appears four times. With deeper trees and more branches the redundancy gets worse.

The representation in accordance with this aspect may be referred to as a 'cross-tab specification'.

### 3. Hierarchic data convolution and devolution

Further to the above, the cross-tab specification may be represented in the form of a Specification Representation Language (SRL). The SRL may be used to correlate data between the 'cross-tab specification' style display of variables/levels/codes and a 'table' style display of rows/columns.

Figure 5 illustrates how SRL is used in accordance with an aspect of invention. The cross-table specification 51 can be represented as a table via a process of 'convolution', and a table 52 may be represented as a cross-table specification 51 through a process called 'devolution'. The convolution and devolution is enable through the SRL 53, which is saved, for example as a file or in memory.

Figure 6 provides a further illustration of this aspect. A variable (for example as shown in Figure 4/5) with three levels having 2, 2 and 5 codes respectively will generate via convolution a table having 2x2x5 = 20 rows or columns. To respecify the table the 20 rows have to be disassembled and the tree structure reassembled. This is called 'devolution'.

Figure 7 illustrates SRL. The SRL is provided as a text representation of the full branch for each row/column, and that can be used to reassemble the tree with a 'recursive devolution' algorithm. Figure 7 illustrates a specification on the left represented as a tree with filters, weights, variables, codes, bases and stats – some with percentages (circles with % in) and some not (other circles).

The SRL is a text string that substantially describes a row/column vector in a table in a way that also preserves the branching information from the tree representation of the table specification.

The general form of the lines is:

{xxx}[yyy] .... var[base](%code) string 6

where

- xxx and yyy are one filter and weight respectively in a modifying prefix before the variable
- var is the variable or codeframe whose codes are being tabulated
- base is an expression indicating how the numbers will be percentaged
- code is the code number or other reference being shown in this row/column with the presence of a % sign meaning this row/column can be percentaged.

Some example lines:

[WeightRegion25()]Occupation[cwf](cwf)

[WeightRegion25()]Occupation[cwf](%1)

[WeightRegion25()]{Location(2)}GenMar(%1:1)

[WeightRegion25()]{Location(2)}GenMar(1:avg)

The lines can be assembled manually by reading the branch path down to any leaf node in a specification tree. Filters are written inside {} braces, weights in [] braces. These are the early nodes of a specification and represent a modifying prefix before the variable/code information is met.

For example the prefix {Gender(1)}[WeightRegion25()] means that the first node in this path at the root of the tree was a filter Gender(1) and subordinate to that was a weight node WeightRegion25(). There can be any number of these in any order.

The only unbracketed element of the line is the variable reference that immediately follows the modifying prefix. This may be a simple variable, and to distinguish from hierarchic variables is here called just a codeframe, or it could be a hierarchic variable.

Codes for simple variables are generally the code number but can also be any of two dozen mnemonics like 'tot' for total, 'avg' for average and 'cwf' for the cases-weighted-filtered base count.

Codes for hierarchic variables are the same elements (code numbers or mnemonics) but now one for each level of the variable separated by colons.

In Figure 7, at the Top right is the SRL, shown here as a text representation of the convoluted form of this tree as it is stored in files. Introductory filters are in { }, introductory weights in [ ]. A base expression is in [ ] immediately after the variable name. Hierarchic codes are indicated with the colon-separator syntax. The flag indicating codes to be based is a % as the first thing inside the code bracket. All rows begin [WeightRegion25()] meaning a weight node that is parent to all others. The last six rows continue with {Location(2)} meaning a filter node parent to all the GenMar nodes. The first 6 rows continue with Occupation[cwf] meaning a codeframe (simple variable in this case) using cwf as a base. Although specific indicia and / or formats have been used in this SRL, it is used by way of example only and the SRL may use any indicia and / or format without departing from this aspect of invention.

Thus, in the example illustrated in Figure 7,

[WeightRegion25()]Occupation[cwf](cwf) = plot cwf code not percentaged

[WeightRegion25()]Occupation[cwf](%1) = plot code 1 percentaged

[WeightRegion25()]{Location(2)}GenMar(%1:1) shows hierarchic node 1:1 meaning Gender(1)=Male and Married(1)=Yes

[WeightRegion25()]{Location(2)}GenMar(1:avg) shows hierarchic node 2:avg meaning Female Average.

From this example, it can be seen that each line of the SRL represents a description of a row in the table that is also a description of its path from root to leaf in a tree representation like Figure 3. Further, the collection of SRL lines can be 'devolved' to produce the specification version of the data tree as in Figure 4.

The SRL may be stored and used as required in a convolution and / or devolution process.

In Figure 7, in the bottom right, this information can be seen on screen as the convoluted tree structure where the hierarchic variable has been expanded.

Specification trees may also include function expressions. These are stored with an introductory @ and can be isolated or under codeframes.

{flt(1)}[weight()]@expression

{flt(1)}[weight()]var[base](@expression)

This aspect of invention enables the conversion between a Table Axis Specification Tree into a list of Table row/column vectors as shown in Figure 7. The critical part of the vector for this description is the Specification Representation Language lines that substantially completely describes the row/column vectors and is used for saving to file. The Convolution/Devolution method is embedded in tree-walking and tree-generation algorithms.

**Generating Vectors from a Specification Tree**

The forward process takes a specification tree and generates the vectors that are the rows or columns of a table. Each of these is completely described by a SRL line.

The main driving function, GenerateVectors(), is a recursive routine that starts at the top of the tree gathering nodes until it reaches particular nodes that provoke the generation of vectors, then backs up to the parent of the node.

Nodes collected along the way are Filter and Weight nodes. These are the early branches of a tree and are followed by vector-generating nodes. When a vector-generating node is reached the collection of filter/weight nodes represents the filter/weight prefix in the SRL line.

Nodes that generate vectors are:

Function – generate a single vector for this function

Codeframe – generate a vector for each child node of the codeframe

Variable – generate the convolution of hierarchic levels

The last of these is another recursive function GenerateVariableVectors() that walks down the Tree Representation of Hierarchic Data multiplying out all the possible combinations. This implements the Convolution algorithm.

**Generating a Specification Tree from Vectors**

The reverse process takes SRL lines that may have been loaded from file or delivered from a displayed table and rebuilds the specification tree.

The main driving function, ReadVectorBlock(), is a recursive routine that walks along the bank of SRL lines from the supplied vectors looking for common prefixes. Any contiguous set of vectors with a common prefix is identified as a block and the routine is called again to look for subsequent common continuations of the prefix within the block. Each block, of course, represents an early node in the specification tree. Identification of blocks within blocks is the essential business of the routine, each block generating one intermediate node on the specification tree.

At the end of the SRL lines are codes and functions that will be the leaf nodes in the tree. Functions and simple codes are easily dealt with, generating a single leaf node each, but hierarchic variable nodes that are recorded like (3:2:4) are collected for processing by another recursive function that generates the Tree Representation of Hierarchic Data from the multitude of expanded lines. This implements the Devolution method.

**Convolution Method**

This takes the Tree Representation of Hierarchic Data and generates a vector for each of the multiplied possibilities. It looks for a particular level, starting with the first, and processes the code lines found underneath. These could be all or some of the available codes at this level and could include pseudo-codes that represent statistics like total and average or bases. Refer to Figure 8a, which illustrates an example of the convolution method according to an aspect of invention. In processing a code, if it is not working at the lowest level the routine is called again to start at the next level down. This way, each code is combined with every code at the next level down, they being combined with every code at the next level down again ad infinitum until the lowest level is reached. When processing codes at the lowest level it generates a vector for each then returns so that the cycle can begin again for the next code at the previous level.

Every time a code is processed its value is placed in the correct spot in an array that is used by the vector-generating routine to build the SRL line. The prefix of this has already been assembled and the final step is to write the codes currently in the array in the form (3:2:4) to complete the SRL.

**Devolution Algorithm**

Figure 8b illustrates an example of the devolution method according to an aspect of invention. The devolution method takes a set of SRL lines and generates a Tree Representation of Hierarchic Data. The lines have already been processed to generate the tree up to the hierarchic data point and the block of lines presented to this algorithm all terminate with code references like (3:2:4).

It is understood the generated tree will have an introductory parent node for the variable under which will be a child node for each level. The present method determines which code nodes are added as children of the level nodes.

The method first assembles the skeleton of the variable – the parent node and child level nodes – then builds the codes for each level in turn. The method then inspects the set of terminal code references looking only at the codes for a particular level. Often, there will be duplication here so the method just notes which codes appear at all and then makes a code tree node for each of them in the order they were first met.

**4. Cross-tabulation of complex data**

a) The segment method.

b) The offset method.

c) The one-level method.

d) The segment matching method

Cross-tabulations have a top and side variable and can also be filtered or weighted. For example, a table of Town by Event might be filtered to a particular Severity, a table of Age by Consumption might be filtered to a particular Year and weighted by Gender to ensure a balance of respondents.

Handling complex data here presents many problems: the four components of a specification (top, side, filter, weight) may be at different levels from hierarchic variables, and may be multi-response and / or incremented. The invention includes several cooperating methods to make it possible to handle all such variation in data.

**a) The segment method.**

Other prior art systems generally treat one code response as the basic datum. In the approach of this aspect of invention data like a1;2b3;4c5;6 is treated as three data items called segments. The first segment is a1;2, the second b3;4 and the third c5;6. The segments can also include increments so c3*30;4*55 is also just one segment. Organised as a data tree to show the hierarchy, the unincremented data is

Town: 1;2

Event: 3;4

Severity: 5;6

A segment can also be thought of as a node of a data tree.

This device has made it possible to achieve efficient cross tabulation and other processing of this sort of data regardless of complexity.

**b) The Offset Method**

Figure 9 illustrates an offset method according to an aspect of invention. The fundamental process in cross tabulation is determining in which cell of the table to store the data for the current case. The response in the across variable may be a code 4 (column index 3) and the side variable a code 6 (row index 5). If the side response is multi like 6;8;11 then three rows are involved. This is for simple data but of course the matter is more complicated for hierarchic data.

Note on INDEX, there is always a tension between enumerating things as a human prefers (1,2,3,4...) and as a computer prefers (0,1,2,3...). In a preferred embodiment, Index means zero-based systems. The first row will be at index zero in an array, the fourth row at index 3. This is why in the examples code a4 turns into a 3 in the calculations. The offsets calculated are also indexes so an offset of 4 means the fifth row.

When an entire hierarchic variable is involved, and because speed is paramount, the offset method is made more complicated because it manages two streams of calculations both involving tree structures.

The kernel is the calculation of the offset itself from a hierarchic branch. For example the offset for the response a4b5c6, the sixth line in the fifth block of the fourth major block, could be calculated arithmetically as (((3xNb)+4)xNc)+5 where Nb and Nc are the number of codes at level b and c.

The algorithm assembles these calculations in a way that is fast and allows for unlimited levels. The principle device is that each layer of parentheses is built on the previous in two steps:

multiply offset by size of this level

add response code-1 at this level

Starting from zero the layers are built like this:

offset = 0

offset = offset x Na + 3

offset = offset x Nb + 4

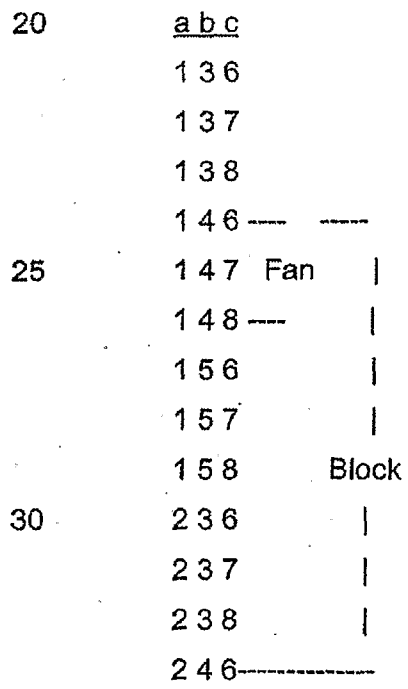
offset = offset x Nc + 5

Since multi-responses are allowed at any level a single response can generate many offsets. For speed these are all built up in parallel and keeping track of which offset is being worked on represents the second stream of calculation. The terms 'fan' and 'block' are employed here.

**Fan and Block**

In the example response a1;2b3;4;5c6;7;8 each of the segments is multi-response and the set will generate 2x3x3 offsets.

The 18 offsets are arranged in a particular order giving rise to terms used in the algorithm. The first offset is for a1, b3, c6 etc.



247

248

Considering the second code in the second segment: b4. When it appears it is in a **'fan'** of three offsets and it next appears a **'block'** of nine offsets later. The fan and block numbers for each level are:

| | | | |
|---|---|---|---|
| Fan | 9 | 3 | 1 |
| | a1;2 | b3;4;5 | c6;7;8 |
| Block | 18 | 9 | 3 |

These numbers are clearly generated by progressively multiplying the number of codes at each level starting with 1. The fan size for one segment is the block size of the next.

In the algorithm these numbers are first generated and stored in an array to facilitate fast processing.

The rest of the algorithm is a nest of loops for each segment, each code, each fan and each block. All offsets are built progressively in parallel. The segment (level) and code contribute to the first stream of calculation using "times size of level, plus code index at this level". The fan and block numbers contribute to the second stream of calculation keeping track of which offset is being updated.

**c) The One Level Method**

A hierarchic variable with three levels Town, Event, Severity might have a response stored as a4b2c4b3c5c6a1;2b3;4c8c5;6b7c9a3b2c1c5c7. In processing this to produce cross-tabs there may be a top or side variable just the Towns, just the Events, just the Severity or the entire variable (all details down to Severity).

In determining the offsets (rows/columns) at which to store data, every segment of data in the string may be considered because a4b2c4 refers to a different row than a5b2c4. Because of the other cooperating methods, this is not usually necessary.

It is only necessary to send to the offset routine segments of the required level. All segments not at the required level can be ignored.

For instance, when an entire hierarchic variable is used the offset routine only needs the leaf nodes because it generates the branch back to the root itself and uses that to calculate the offsets. This is illustrated in Figure 10a.

The response a4b2c4b3c5c6a1;2b3;4c8c5;6b7c9a3b2c1c5c7 arrives at the processing functions divided into seventeen segments (some multi-response) a4 b2 c4 b3 c5 c6 a1;2 b3;4 c8 c5;6 b7 c9 a3 b2 c1 c5 c7.

The One-level method simply scans these sending to the Offset routine only those at the required level. Figure 10b illustrates this.

**d) The segment matching method**

Cross tabulation proceeds by determining in which cell of the table to store the data for the current case. This seems logical for simple data but for complex data that may have many segments the interaction between the segments in the four components of the specification (top, side, filter, weight) needs management.

For instance, for the data shown in Figure 1, if the table is Town by Event then Town has three segments a1...a2...a3 and Event has five segments ..b2...b4...b4...b3...b1. In fact, as Figure 1 shows, the a1 node matches the first two b2...b4 nodes but the last ...a3 node matches only the single last ...b1 node – but this is not at all obvious from the independent strings.

The segment matching method according to this aspect of invention, and as illustrated in Figures 11a, 11b and 11c, is a preparation step in processing the data for one case that collapses and expands segments so that each of the four components has data all with the same number of segments. Processing (where top and side segments are sent to the offset routines to determine in which cells to store data) is then a matter of stepping through the segments in all four components in concert.

Matching up the above 'a' and 'b' segments might mean expanding the 'a's so both strings have five segments:

a1 a1 a2 a2 a3

b2 b4 b4 b3 b1

This preparation is a relatively complicated step but the time it costs is relatively small compared to the time saved in the subsequent processing step. Additionally, it relieves the processing step of another layer of tree navigation making that routine not only faster but easier to design.

Simple data only ever has one segment, even if multi-response or incremented, because simple variables only have one level. Hierarchic data can have many segments so when it appears in any of top, side, filter or weight, the segments must match at the required level to have meaning.

A simple variable always matches anything, the single segment being duplicated as much as required to meet a multi-segment response.

**i) Process Level**

The first step in preparation is to determine a single level to which all four components (top, side, filter, weight) will be aligned.

The top and side variables may be simple (level a) or a single level from a hierarchic variable. If the entire variable is processed it is regarded as being the deepest level. The Process level is the minimum of the top and side levels. For example Town by Severity is processed at level a, Event by Severity at level b, Event by Town at level a.

Aligning the top and side components involves simply noting where the process level segments begin in the top and side responses. For the data to have meaning there must be the same number of these in top and side. Aligning filter and weight components involves additional considerations.

**ii) Filtering at different levels**

If the filter level is deeper than the process level, then a segment will pass the filter test if any child nodes at the filter level pass the test. For instance, from the data pictured in Figure 12a, if the process level is b (Event) then we are interested in which segments at the Event level will make contributions to the table. If the table has the filter Severity(3), meaning 'only show Events that were severity 3', only those events that have a 3 as a child will pass.

The first event segment (2) has one child node that is severity 3 so it passes the test. The second event segment (4) has two child nodes and one of them is a 3 so it also passes the test. The last event segment (1) has two child nodes but neither is a 3 so it fails the test and is said to be 'filtered out'.

When the filter level is shallower than the process level then all child nodes of successful filter nodes will contribute and all child nodes of failed filter nodes will be 'filtered out' and not appear in the table. For instance, from the data pictured in Figure 12b, if the process level is c (Severity) then we are interested in

which of the severity segments will appear in the table. If the table has a filter Event(4) it means 'only show severity for events of type 4'.

The first event segment (2) fails the filter so its single child severity 3 is ignored. The second event segment (4) passes the filter so both its child severity nodes will contribute to the table.

The method that achieves this is shown as a flow chart of Figures 11a, 11b and 11c. The input to that algorithm is the process level, the filter level, the original response (for the above examples a1b2c3b4c5c3a2b4c4b3c3a3b1c1c4) and the filter results at the filter level (e.g. TFTFTFF for the first example). The output is the filter results matched to the process level. (TTFTF for the first example). Effectively, it converts a filter result at one level to the equivalent result at another level by referring to the tree structure implicit in the actual response string.

The method begins by considering the simple cases: a simple variable has only one segment so duplicate it to match segments at the process level; if the filter and process levels are the same they are already matched. Then it branches off to the more intricate routines for deeper and shallower filters.

A similar method is employed to collapse/expand weighting values at different levels.

**5. Grid construction generator for making hierarchic variables**

This aspect of invention provides a visual way of assembling individual variables into a single grid or hierarchic variable. This aspect of invention may be used to convert relatively simple data which is somewhat repetitive into hierarchical data.

Referring to Figure 13, the hierarchic variable named Stack is constructed from 18 simple variables by dropping them into the grid. The grid is sized and labelled by dropping some of these simple variables into edit fields at the top of the form. Clicking 'Generate' produces the 3x6x6=108 set of construction lines that, when run, build the hierarchic stack variable.

```
1:1:1=Stack1(1)
1:1:2=Stack1(2)
...
3:6:5=Stack18(5)
```

3:6:6=Stack18(6)

The benefit is in saving time and effort and in a visual device that makes the task intuitive.

The construction script is a fairly common device, each line meaning, "output a code on the left of the = whenever the data satisfies the filter condition on the right." In normal circumstances the code would be just a single number and here they are obviously hierarchic.

In practice there are two preferred interpretations of the grid. The following indicates the actions taken to achieve each interpretation with a view to making clear the visual, intuitive character of the device. The generation is fairly standard once the meaning of the interpretations is clear.

The first interpretation is characterised by having nothing in the third level edit fields. This signal tells the program to generate filter expressions using the top or side variables supplied. The second interpretation is characterised by having a variable in the third level edit field. This signal tells the program to generate filter expressions from the variables in the cells.

In both interpretations the labels for the levels of the generated variable are automatically taken from component variables (either their description or code labels depending on context) saving a great deal of time and effort.

In essence, the device reduces a great deal of manual effort in which complexity, repetitiveness and sheer mass make fatigue and mistakes likely. The visual character also replaces the cognitive ordeal of maintaining command of a multitude of text lines, with an easily comprehendible visual metaphor.

**5a Each row is one variable or each column is one variable**

Figure 14 shows four variables named Q10 to Q13 each with the same codeframe of five items from Strongly Approve to Strongly Disapprove. What you want is one hierarchic variable with the first level being Party and the second level Approval.

Referring to Figure 15, firstly, drag Q10 into the Top Level Codeframe (dark blue). This will fill out the top cells with the labels from Q10's codeframe and set top cells to 5. Actually any of Q10-13 would do because the codeframe is all that is required.

Secondly, increase Side cells to 4 (pale blue circle) and drop each of Q10-13 in the side cells.

Change the Level Names to properly describe what's there – Approval and Party

Clicking 'Generate' produces the script shown in Figure 16.

If, for example, it is decided that this is the wrong way round, and Party is to be the first level and Approval is to be the second, then click 'Grid' again and match the following as illustrated in Figure 17. This shows you can reverse the orientation of the table to suit.

Clicking Generate gives Figure 18. Note the filter expressions are built from the Side variables. When this script is run the data produced is shown in Figure 19. The top panel shows the data for each case in its compact form and the lower panel shows the first case represented as a tree structure.

**5b. Each Cell is one variable**

In highly atomised field systems where multi-response can't be stored, a grid question arises where each cell has been stored as a separate variable.

In the example shown in Figure 20, Q41-46 hold the rating score given by respondents for two brands against three statements - how attractive is brand 1 etc.

Referring to Figure 21, firstly (dark blue), the variable Q40Attributes dragged to the Side Codeframe field provides the labels for the side of the table (second level of the hierarchic variable being generated) and the variable Q41 dragged to the Cells Codeframe field provides the labels for the third level. Secondly (light blue), setting the Top Level to 2 codes establishes the size of the table and the variables Q41-46 are dragged into the cells as indicated. These provide the names for the filter expressions of the construction lines that will be generated. Thirdly (magenta), two variables dragged into the top cells provide the labels for the top of the table (first level of the hierarchic variable).

Figure 22 shows the resulting construction script. Note the filter expressions are from the variables in the cells. Figure 23 shows the data from the generated variable in its compact form and the first case shown as a tree structure.

While this invention has been described in connection with specific embodiments thereof, it will be understood that it is capable of further modification(s). This application is intended to cover any variations uses or adaptations of the invention following in general, the principles of the invention and including such departures from the present disclosure as come within known or customary practice within the art to which the invention pertains and as may be applied to the essential features hereinbefore set forth.

As the present invention may be embodied in several forms without departing from the spirit of the essential characteristics of the invention, it should be understood that the above described embodiments are not to limit the present invention unless otherwise specified, but rather should be construed broadly within the spirit and scope of the invention as defined in the appended claims. Various modifications and equivalent arrangements are intended to be included within the spirit and scope of the invention and appended claims. Therefore, the specific embodiments are to be understood to be illustrative of the many ways in which the principles of the present invention may be practiced. In the following claims, means-plus-function clauses are intended to cover structures as performing the defined function and not only structural equivalents, but also equivalent structures. For example, although a nail and a screw may not be structural equivalents in that a nail employs a cylindrical surface to secure wooden parts together, whereas a screw employs a helical surface to secure wooden parts together, in the environment of fastening wooden parts, a nail and a screw are equivalent structures.

"Comprises/comprising" when used in this specification is taken to specify the presence of stated features, integers, steps or components but does not preclude the presence or addition of one or more other features, integers, steps, components or groups thereof. Thus, unless the context clearly requires otherwise, throughout the description and the claims, the words 'comprise', 'comprising', and the like are to be construed in an inclusive sense as opposed to an exclusive or exhaustive sense; that is to say, in the sense of "including, but not limited to".

**THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:**

1. A data format adapted to represent hierarchical data such as from a survey response, comprising:

a string of indicia, the string including indicators of tree depth (level).

2. A data format as claimed in claim 1, wherein the indicators are provided for each level.

3. A data format as claimed in claim 1 or 2, wherein the indicators are represented by different indicia and/or indicia format.

4. A data format as claimed in claim 1, further comprising representing multi-response data by a delimiter.

5. A data format as claimed in claim 1, further comprising representing incremented data by a delimiter

6. A data format as claimed in claim 4 or 5, wherein the delimiter is represented by different indicia and/or indicia format.

7. A data format as claimed in any one of claims 1 to 6, wherein the string is a single string.

8. An analytical tool adapted to provide analysis based on data formatted in accordance with any one of claims 1 to 7.

9. A GUI representable data format for displaying hierarchical data, comprising:

at least one first folder and

at least one second folder, the second folder being provided within the first folder

each second folder including code(s) related to a corresponding level of the hierarchy.

10. A data format as claimed in claim 9, wherein the order of the folders represents a hierarchical structure.

11. A data format as claimed in claim 9, wherein the first folder represents the root of the hierarchy, such as a variable.

12. A data format as claimed in claim 9, wherein the at least one second folder represents the levels, preferably in order.

13. A data format as claimed in claim 12, wherein the at least one second folder represents the levels in order.

14. A data format as claimed in claim 9, wherein the code is an attribute of the at least one second folder.

15. A method of representing the structure of hierarchical data, the method comprising the steps of:

providing a first folder representing a variable,

providing at least one second folder, within the first folder, each second folder representing a level, and

providing within each second folder, code(s) for that level.

16. A method as claimed in claim 15 further comprising the step of ordering the folders to represent the hierarchical structure.

17. A method of converting data represented in a first format to data represented in a second format, the method comprising the step of:

using the SRL in the process of converting the first format to the second format.

18. A method as claimed in claim 17, wherein the SRL is stored in a file.

19. A cross-table specification represented in SRL.

20. A schema adapted to represent a cross table specification, comprising:
first indicia representing a variable and
second indicia representing code.

21. A schema as claimed in claim 20, further comprising any one or any combination of:
third indicia representing weights
fourth indicia representing filters
fifth indicia representing hierarchical code
sixth indicia representing flags
seventh indicia representing a base expression.

22. A schema as claimed in claim 20 or 21, wherein each line of the schema describes a row or column (of tabulated data)

23. A schema as claimed in claim 20 or 22, wherein each line of the schema describes a table specification node back to its root.

24. A specification representation language comprising:
a general form:
{xxx}[yyy] .... var[base](%code)
where
- xxx and yyy are one filter and weight respectively in a modifying prefix before the variable
- var is the variable or codeframe whose codes are being tabled
- base is an expression indicating how the numbers will be percentaged
- code is the code number or other reference being shown in this row/column with the presence of a % sign meaning this row/column can be percentaged.

25. A method of processing data, the method comprising the steps of:
providing data representing a hierarchy having at least two levels, each level commonly having at least one code, and
processing the data at each level as a single unit (segment).

26. A method as claimed in claim 25, wherein the data is survey data.

27. A method of determining a row or column in a table applicable to a response having a complex data structure, the method comprising the steps of:
determining the response
determining the structure of the variable, and
processing the structure arithmetically to determine the row or column for the response.

28. A method as claimed in claim 27, wherein the processing comprises:
multiplying offset by size of a level, and
adding response code less one to the offset.

29. A method as claimed in claim 27 or 28, further comprising first initialising the offset to zero and then processing for each level of the structure.

30. A method of processing a response, the method comprising the steps of:
determining the level, and
processing only segments at that level.

31. A method as claimed in claim 30, wherein the processing is in accordance with any one of claims 27 to 29.

32. A method as claimed in claim 30, further comprising the step of:
accommodating a variable by providing segments equivalent to the segments at that level.

33. The method as claimed in claim 32, wherein the variable is a filter, top, side, weight.

34. A method as claimed in claim 32, wherein the segments are additional segments.

35. A method as claimed in claim 32, wherein the segments are collapsed.

36. A method of arranging variables in a grid configuration, the method comprising the steps of:

selecting the variable

providing a grid structure, and

slotting the variable into the grid at a desired location.

37. Apparatus adapted to representing the structure of hierarchical data, said apparatus comprising:

processor means adapted to operate in accordance with a predetermined instruction set,

said apparatus, in conjunction with said instruction set, being adapted to perform the method as claimed in any one of claims 15, 17, 25, 27, 30, 32 or 36.

38. A computer program product comprising:

a computer usable medium having computer readable program code and computer readable system code embodied on said medium for cooperating a data processing system, said computer program product including:

computer readable code within said computer usable medium being adapted to perform the method as claimed in any one of claims 15, 17, 25, 27, 30, 32 or 36.

39. A method as herein disclosed.

40. An apparatus and/or device as herein disclosed.

41. A specification as herein disclosed.

42. A schema as herein disclosed.

43. A data format as herein disclosed.

44. A specification representation language as herein disclosed.

45. An analytical tool as herein disclosed.

Figure 1

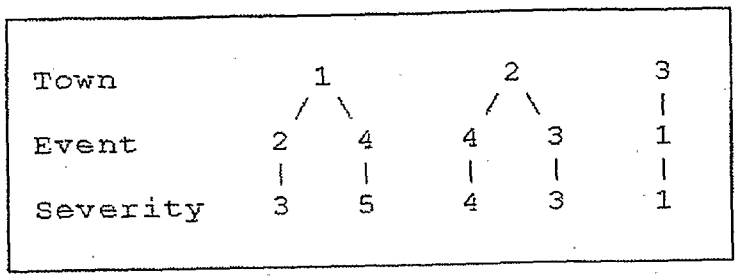



Figure 2

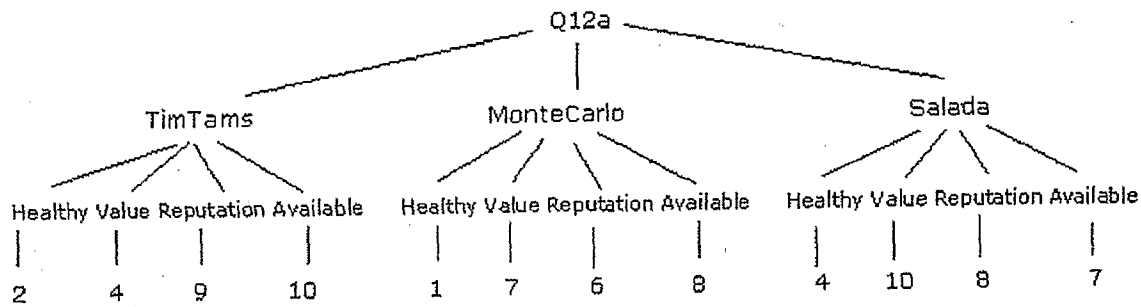
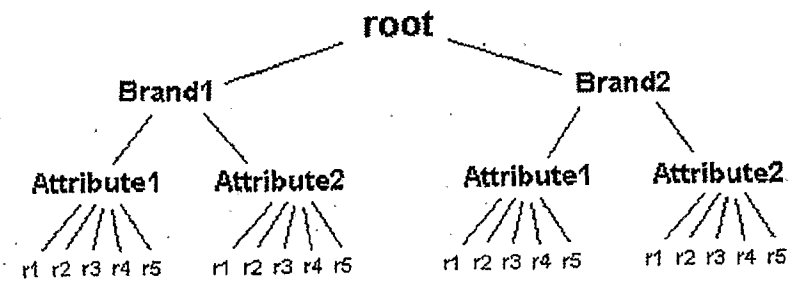
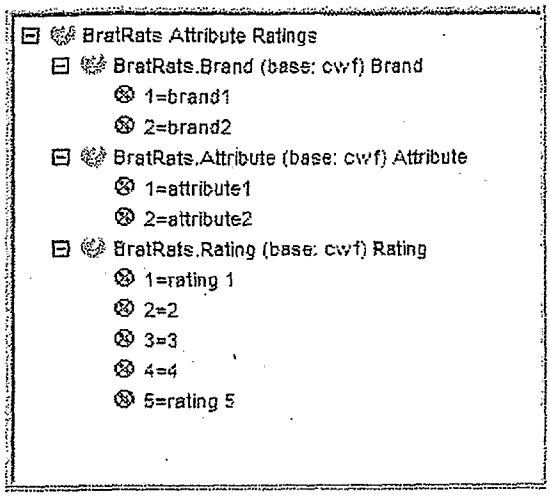


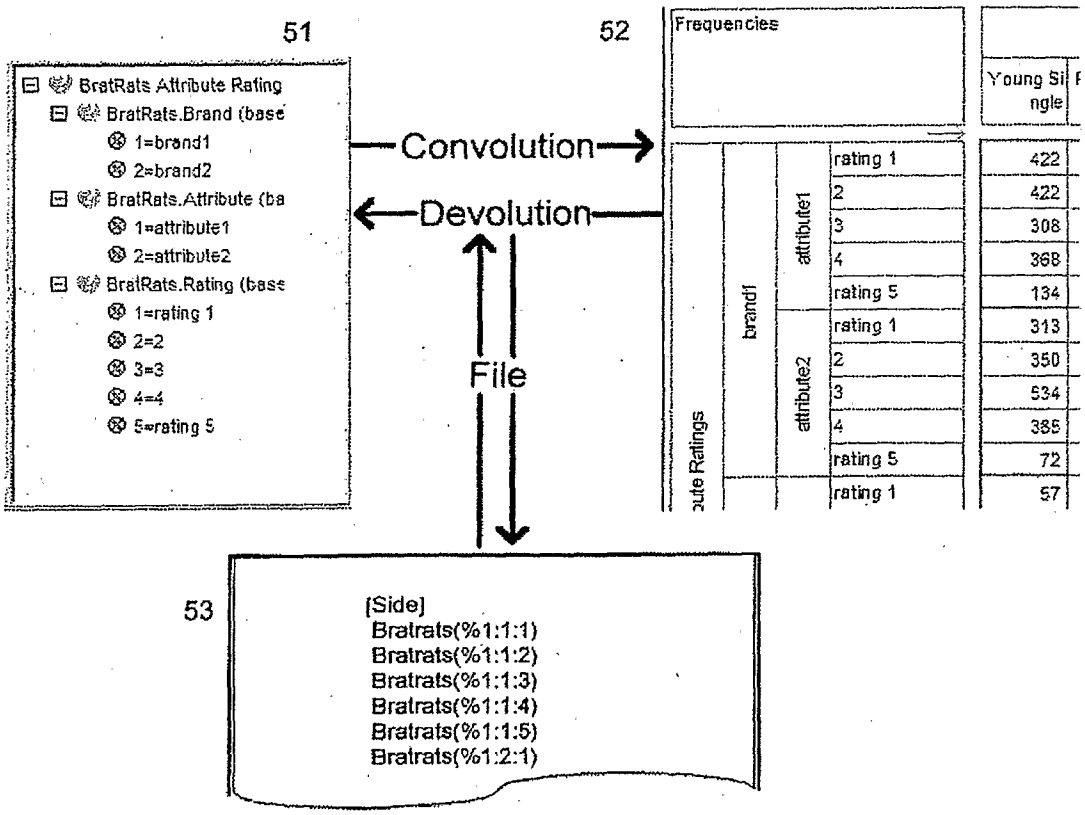

Figure 3



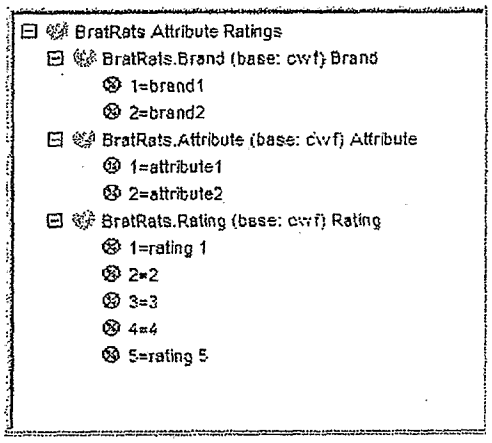
## Figure 4




## Figure 5



## Figure 6




Frequencies

| | | | | Young Single | Pee |
|---|---|---|---|---|---|
| Attribute Ratings | brand1 | attribute1 | rating 1 | 422 | |
| | | | 2 | 422 | |
| | | | 3 | 308 | |
| | | | 4 | 368 | |
| | | | rating 5 | 134 | |
| | | attribute2 | rating 1 | 313 | |
| | | | 2 | 350 | |
| | | | 3 | 534 | |
| | | | 4 | 385 | |
| | | | rating 5 | 72 | |
| | brand2 | attribute1 | rating 1 | 57 | |
| | | | 2 | 470 | |
| | | | 3 | 568 | |
| | | | 4 | 291 | |
| | | | rating 5 | 268 | |
| | | attribute2 | rating 1 | 261 | |
| | | | 2 | 356 | |
| | | | 3 | 377 | |
| | | | 4 | 393 | |
| | | | rating 5 | 267 | |

## Figure 7

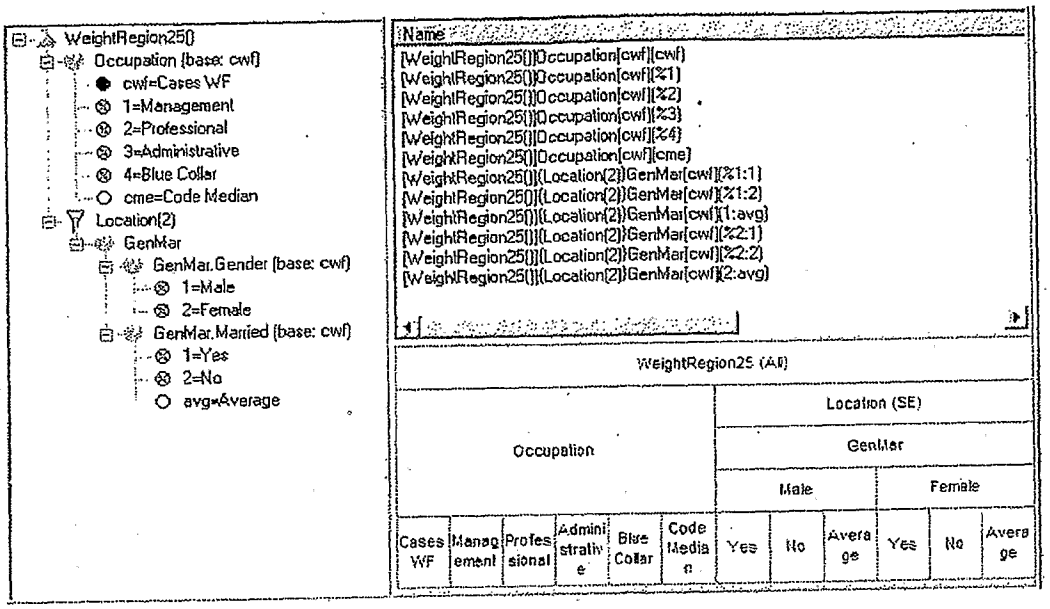

WeightRegion25()
Occupation (base: cwf)
cwf=Cases WF
1=Management
2=Professional
3=Administrative
4=Blue Collar
cme=Code Median
Location(2)
GenMar
GenMar.Gender (base: cwf)
1=Male
2=Female
GenMar.Married (base: cwf)
1=Yes
2=No
avg=Average
Name
[WeightRegion25()]Occupation[cwf][cwf]
[WeightRegion25()]Occupation[cwf][%1]
[WeightRegion25()]Occupation[cwf][%2]
[WeightRegion25()]Occupation[cwf][%3]
[WeightRegion25()]Occupation[cwf][%4]
[WeightRegion25()]Occupation[cwf][cme]
[WeightRegion25()]{Location(2)}GenMar[cwf][%1:1]
[WeightRegion25()]{Location(2)}GenMar[cwf][%1:2]
[WeightRegion25()]{Location(2)}GenMar[cwf][1:avg]
[WeightRegion25()]{Location(2)}GenMar[cwf][%2:1]
[WeightRegion25()]{Location(2)}GenMar[cwf][%2:2]
[WeightRegion25()]{Location(2)}GenMar[cwf][2:avg]
WeightRegion25 (All)
Occupation
Location (SE)
GenMar
Male
Female
Cases WF
Management
Professional
Administrative
Blue Collar
Code Median
Yes
No
Average
Yes
No
Average

**Figure 8a**

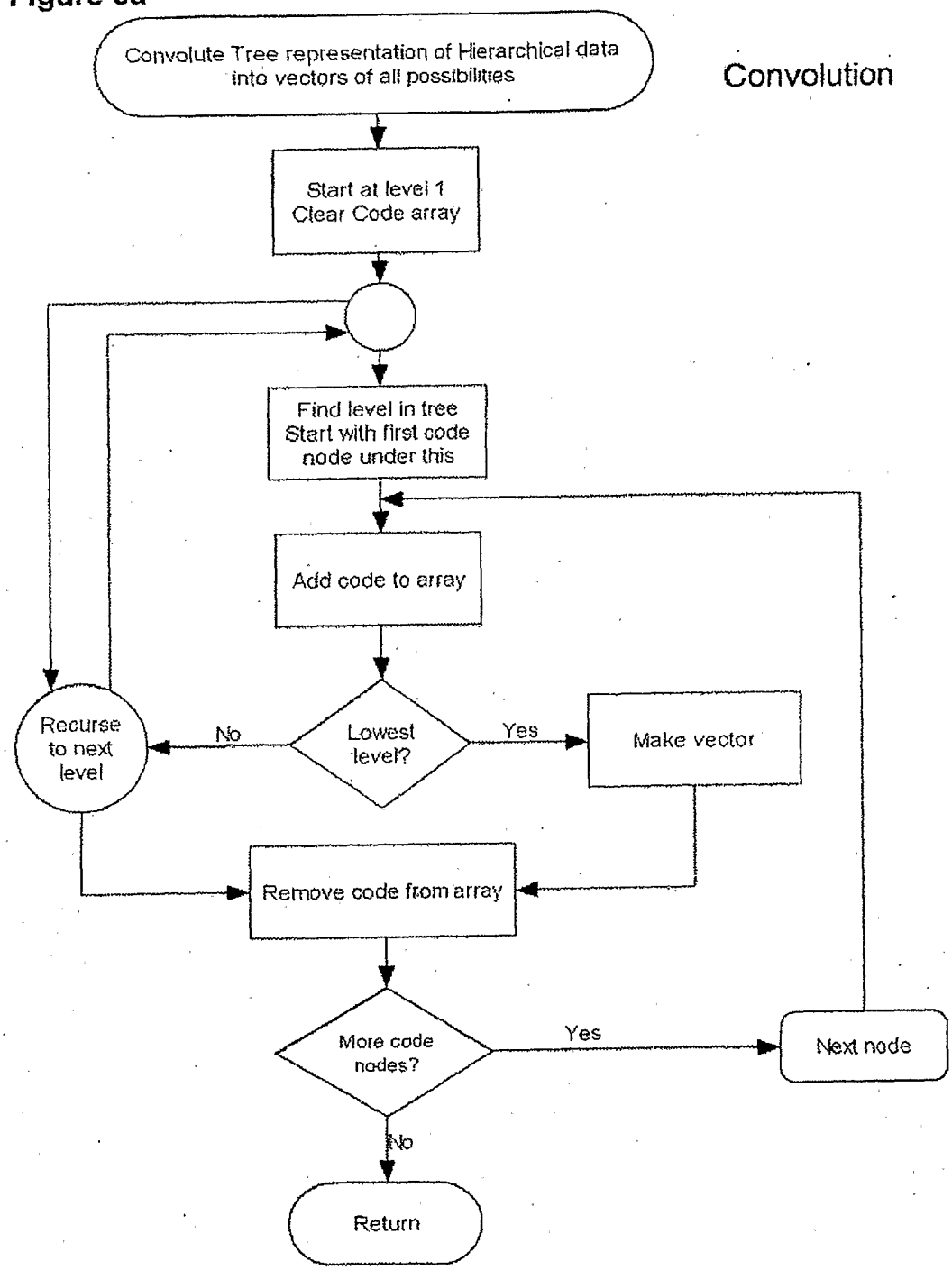

Convolute Tree representation of Hierarchical data into vectors of all possibilities
Convolution
Start at level 1
Clear Code array
Find level in tree
Start with first code
node under this
Add code to array
Recurse
to next
level
No
Lowest
level?
Yes
Make vector
Remove code from array
More code
nodes?
Yes
Next node
No
Return

**Figure 8b**

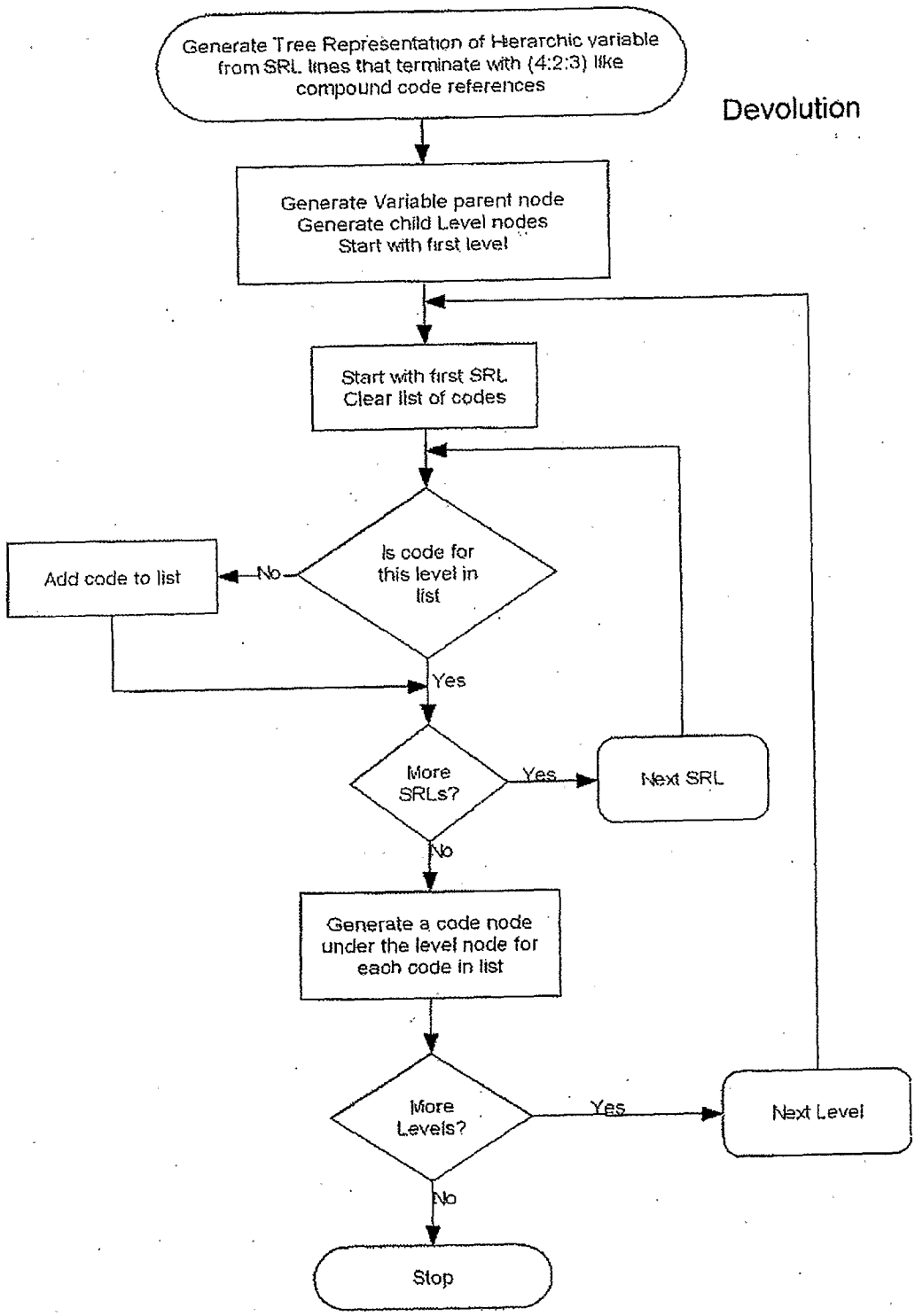

Generate Tree Representation of Herarchic variable from SRL lines that terminate with (4:2:3) like compound code references
Devolution
Generate Variable parent node
Generate child Level nodes
Start with first level
Start with first SRL
Clear list of codes
Is code for this level in list
No
Add code to list
Yes
More SRLs?
Yes
Next SRL
No
Generate a code node under the level node for each code in list
More Levels?
Yes
Next Level
No
Stop

Figure 9

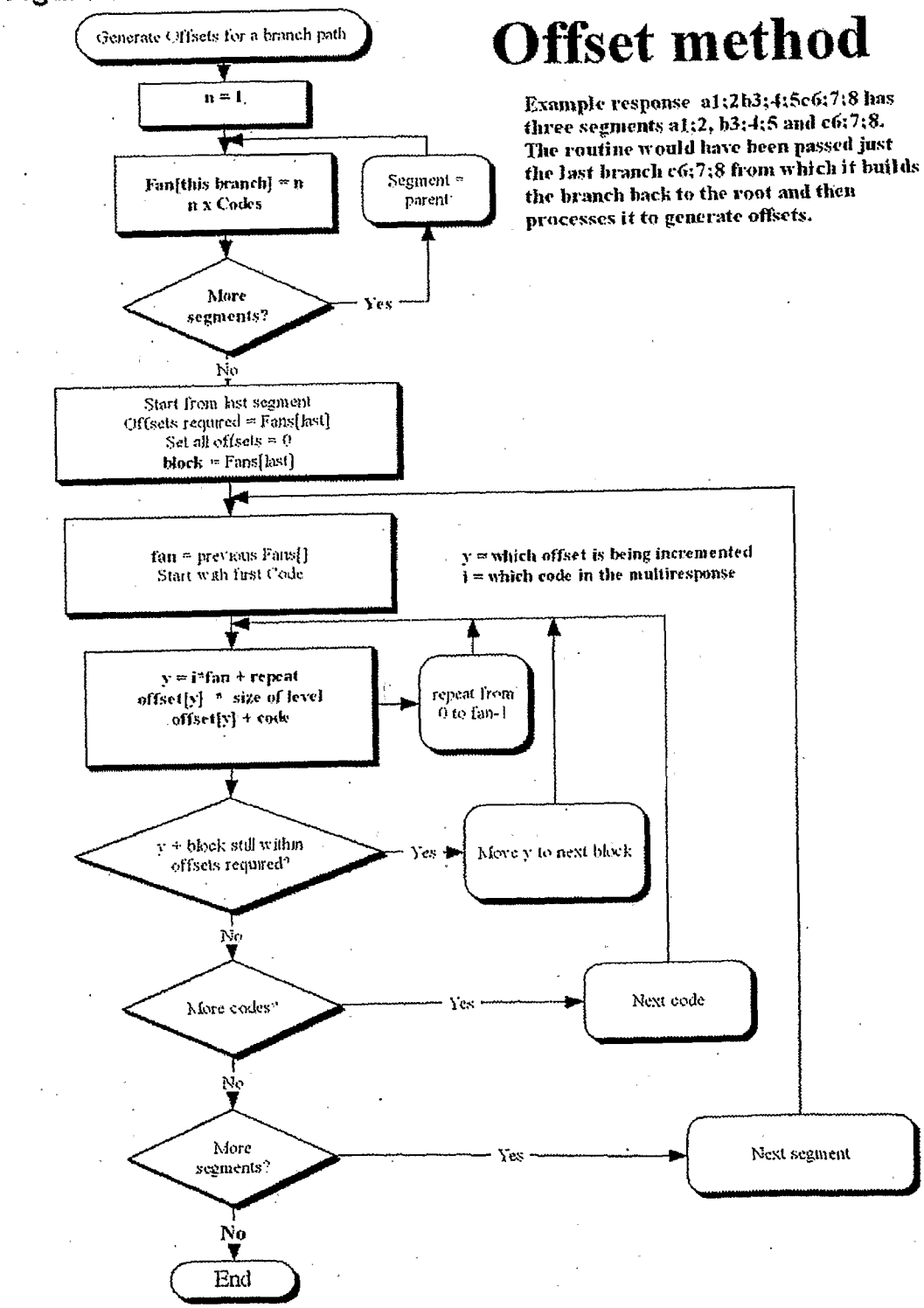

Offset method
Example response a1;2b3;4;5c6;7;8 has three segments a1;2, b3;4;5 and c6;7;8. The routine would have been passed just the last branch c6;7;8 from which it builds the branch back to the root and then processes it to generate offsets.
Generate Offsets for a branch path
n = 1
Fan[this branch] = n
n x Codes
Segment = parent
More segments?
Yes
No
Start from last segment
Offsets required = Fans[last]
Set all offsets = 0
block = Fans[last]
fan = previous Fans[]
Start with first Code
y = which offset is being incremented
i = which code in the multiresponse
y = i*fan + repeat
offset[y] * size of level
offset[y] + code
repeat from 0 to fan-1
y + block still within offsets required?
Yes
Move y to next block
No
More codes?
Yes
Next code
No
More segments?
Yes
Next segment
No
End

**Figure 10a**

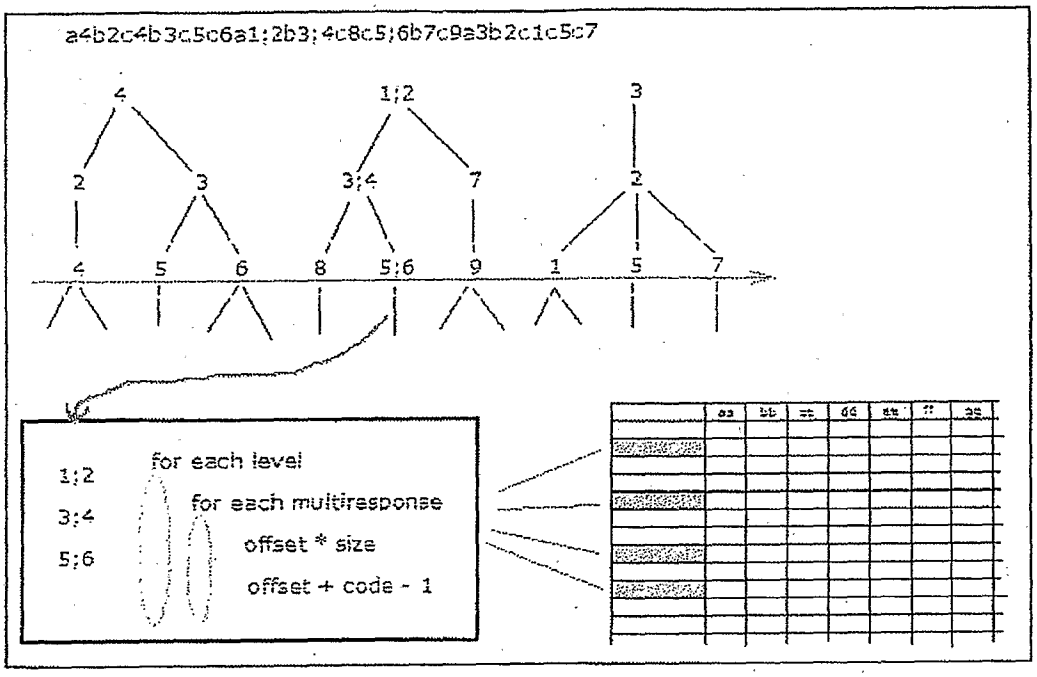

a4b2c4b3c5c6a1;2b3;4c8c5;6b7c9a3b2c1c5c7
4
2
3
4
5
6
1;2
3;4
7
8
5;6
9
3
2
1
5
7
1;2
3;4
5;6
for each level
for each multiresponse
offset * size
offset + code - 1

**Figure 10b**

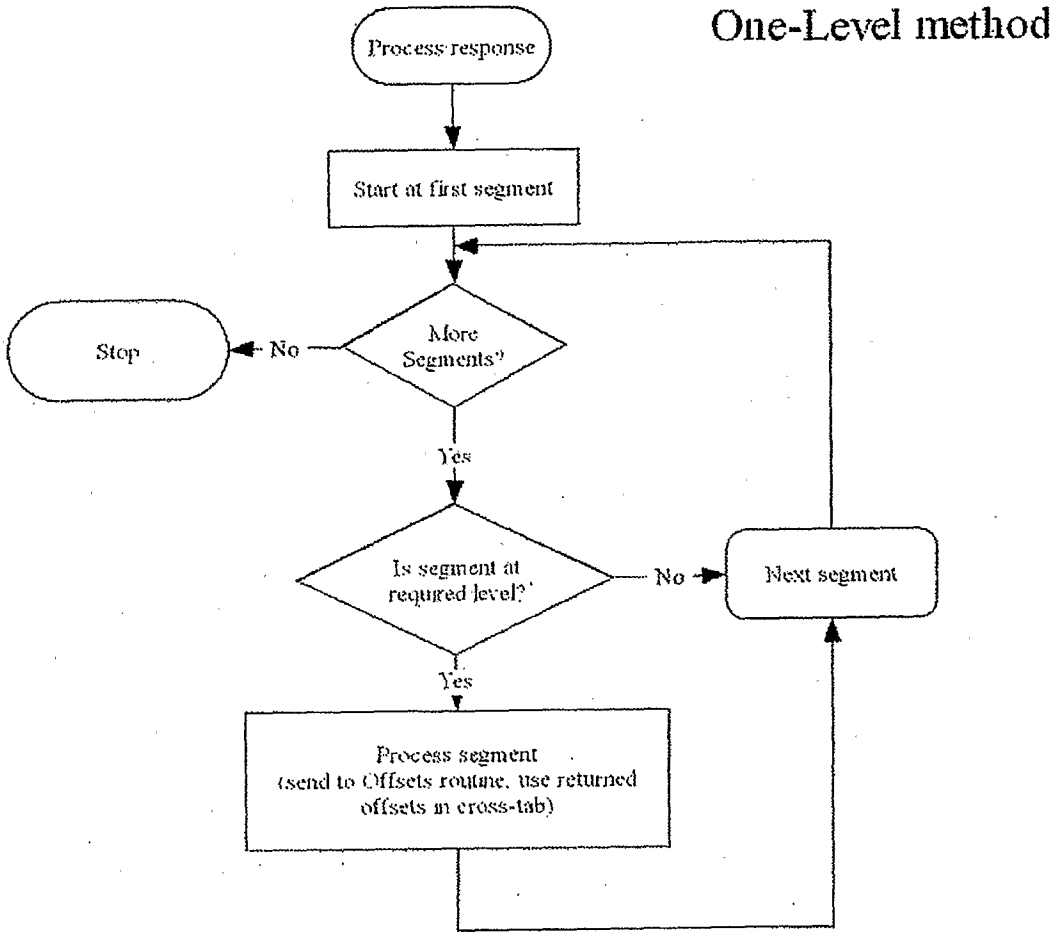
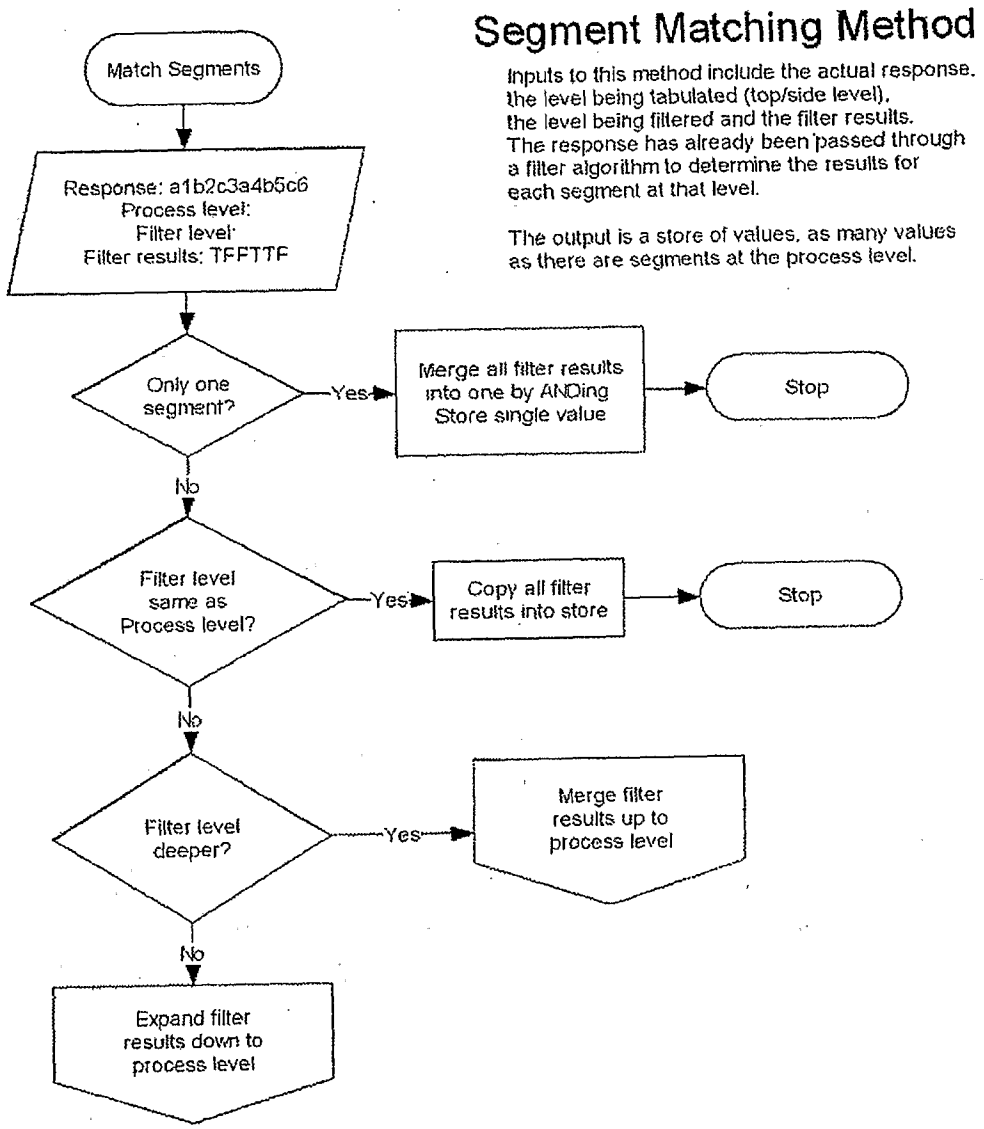
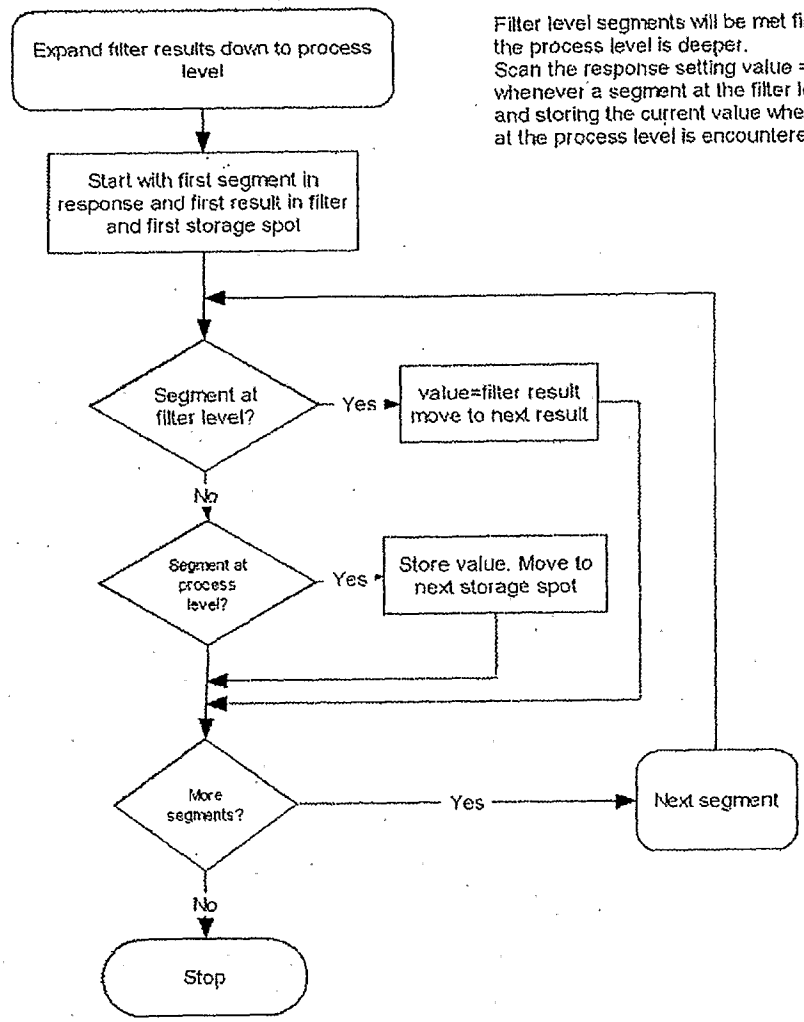

One-Level method
Process response
Start at first segment
More Segments?
No
Stop
Yes
Is segment at required level?
No
Next segment
Yes
Process segment (send to Offsets routine, use returned offsets in cross-tab)

**Figure 11a**


Segment Matching Method
Inputs to this method include the actual response, the level being tabulated (top/side level), the level being filtered and the filter results. The response has already been passed through a filter algorithm to determine the results for each segment at that level.
The output is a store of values, as many values as there are segments at the process level.
Match Segments
Response: a1b2c3a4b5c6
Process level:
Filter level:
Filter results: TFFTTF
Only one segment?
Yes
Merge all filter results into one by ANDing
Store single value
Stop
No
Filter level same as Process level?
Yes
Copy all filter results into store
Stop
No
Filter level deeper?
Yes
Merge filter results up to process level
No
Expand filter results down to process level

## Figure 11b


Expand filter results down to process level
Start with first segment in response and first result in filter and first storage spot
Segment at filter level?
Yes
value=filter result move to next result
No
Segment at process level?
Yes
Store value. Move to next storage spot
More segments?
Yes
Next segment
No
Stop
Filter level segments will be met first because the process level is deeper.
Scan the response setting value = next filter result whenever a segment at the filter level is encountered and storing the current value whenever a segment at the process level is encountered

## Figure 11c

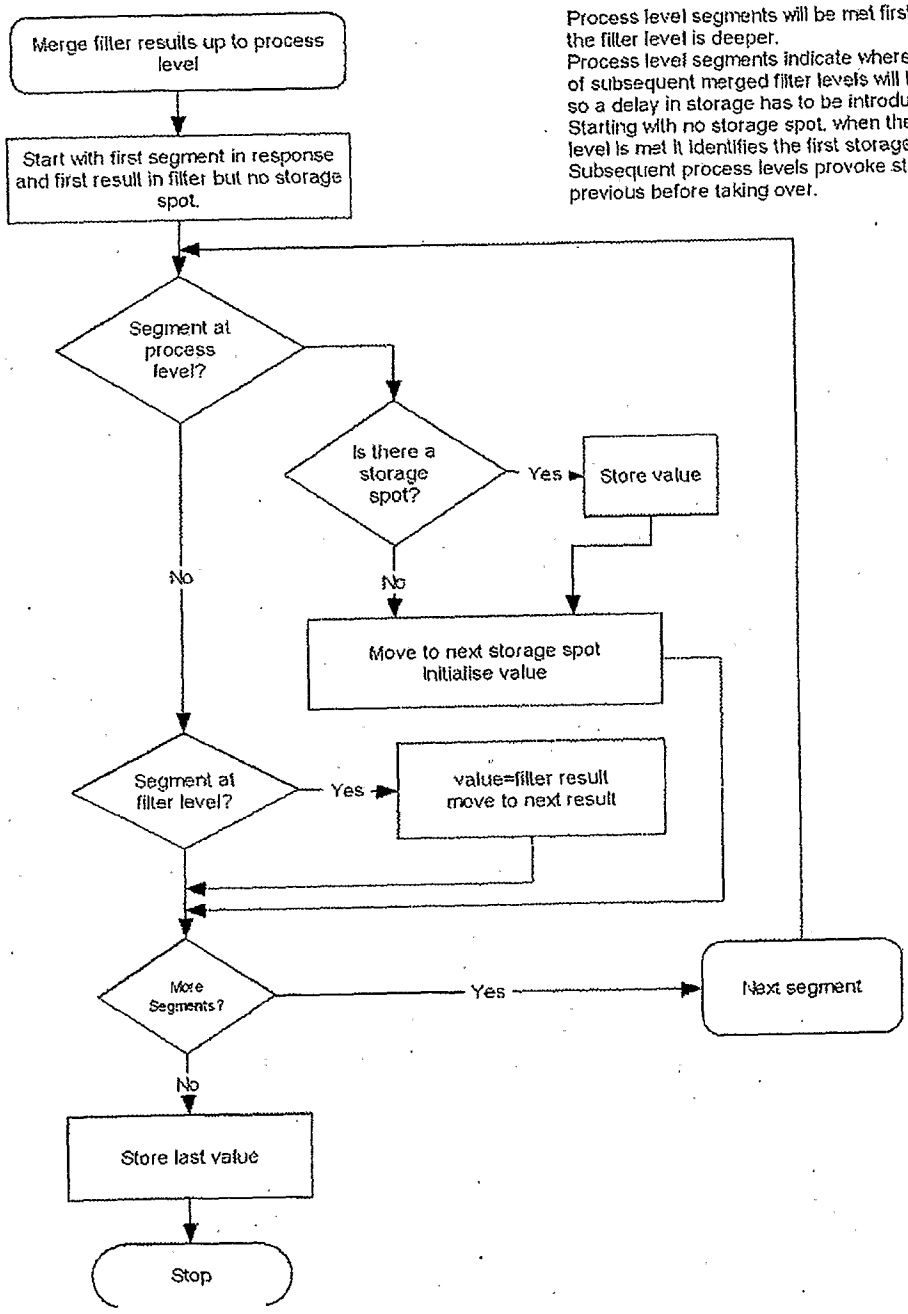

Merge filter results up to process level
Start with first segment in response and first result in filter but no storage spot.
Process level segments will be met first because the filter level is deeper.
Process level segments indicate where the value of subsequent merged filter levels will be stored so a delay in storage has to be introduced.
Starting with no storage spot, when the first process level is met it identifies the first storage spot.
Subsequent process levels provoke storage in the previous before taking over.
Segment at process level?
Is there a storage spot?
Yes
Store value
No
No
Move to next storage spot
Initialise value
Segment at filter level?
Yes
value=filter result
move to next result
More Segments?
Yes
Next segment
No
Store last value
Stop

Figure 12a

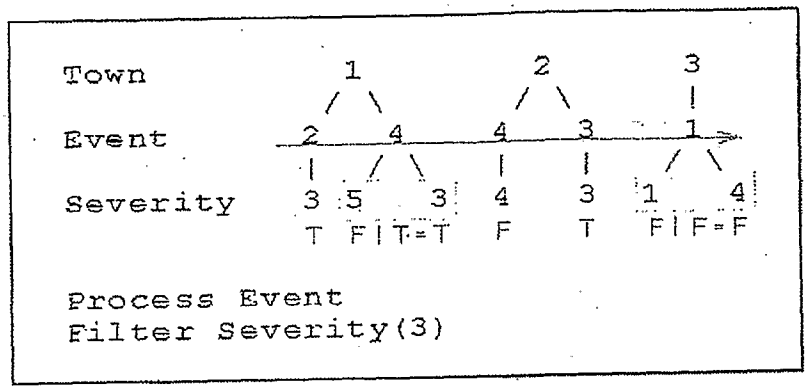
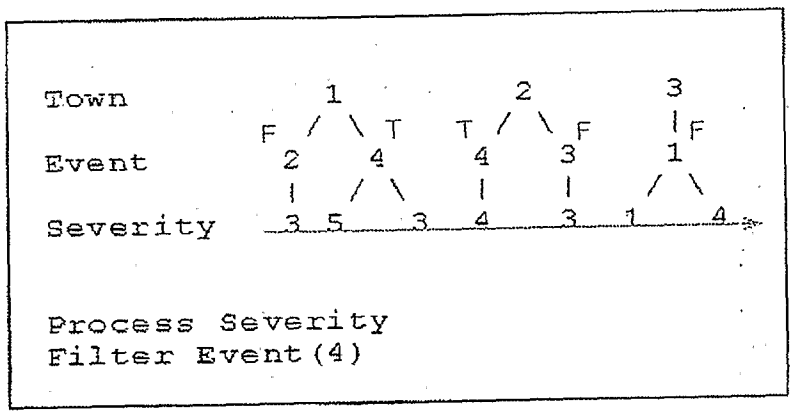
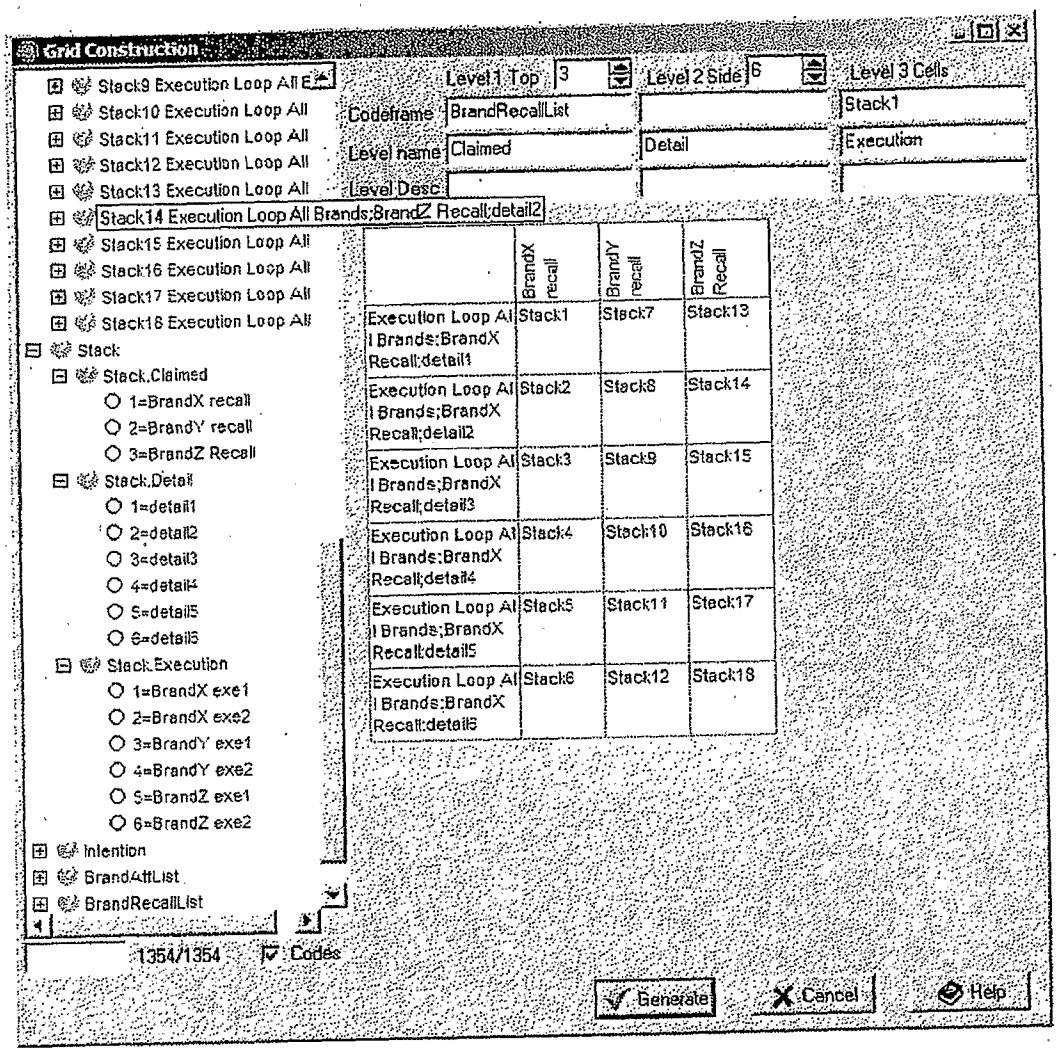



Figure 12b



## Figure 13


Grid Construction
Stack9 Execution Loop All E
Stack10 Execution Loop All
Stack11 Execution Loop All
Stack12 Execution Loop All
Stack13 Execution Loop All
Stack14 Execution Loop All Brands;BrandZ Recall;detail2
Stack15 Execution Loop All
Stack16 Execution Loop All
Stack17 Execution Loop All
Stack18 Execution Loop All
Stack
Stack.Claimed
1=BrandX recall
2=BrandY recall
3=BrandZ Recall
Stack.Detail
1=detail1
2=detail2
3=detail3
4=detail4
5=detail5
6=detail6
Stack.Execution
1=BrandX exe1
2=BrandX exe2
3=BrandY exe1
4=BrandY exe2
5=BrandZ exe1
6=BrandZ exe2
Intention
BrandAttList
BrandRecallList
1354/1354
Codes
Level 1 Top 3
Level 2 Side 6
Level 3 Cells
Codeframe BrandRecallList
Stack1
Level name Claimed
Detail
Execution
Level Desc
BrandX recall
BrandY recall
BrandZ Recall
Execution Loop All Brands;BrandX Recall;detail1 Stack1 Stack7 Stack13
Execution Loop All Brands;BrandX Recall;detail2 Stack2 Stack8 Stack14
Execution Loop All Brands;BrandX Recall;detail3 Stack3 Stack9 Stack15
Execution Loop All Brands;BrandX Recall;detail4 Stack4 Stack10 Stack16
Execution Loop All Brands;BrandX Recall;detail5 Stack5 Stack11 Stack17
Execution Loop All Brands;BrandX Recall;detail6 Stack6 Stack12 Stack18
Generate
Cancel
Help

## Figure 14

| | Strongly approve | Approve | Don't care | Disapprove | Strongly disapprove |
|---|---|---|---|---|---|
| Q10 Conservative party | | | | | |
| Q11 Radical party | | | | | |
| Q12 Socialist party | | | | | |
| Q13 Independents | | | | | |

## Figure 15

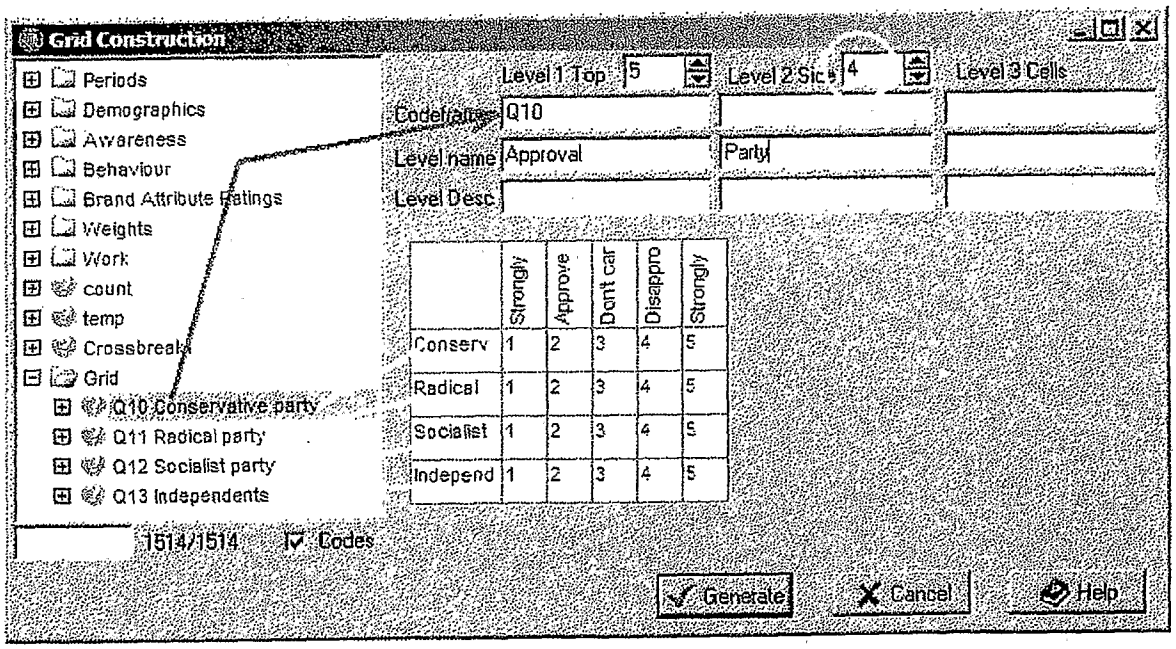
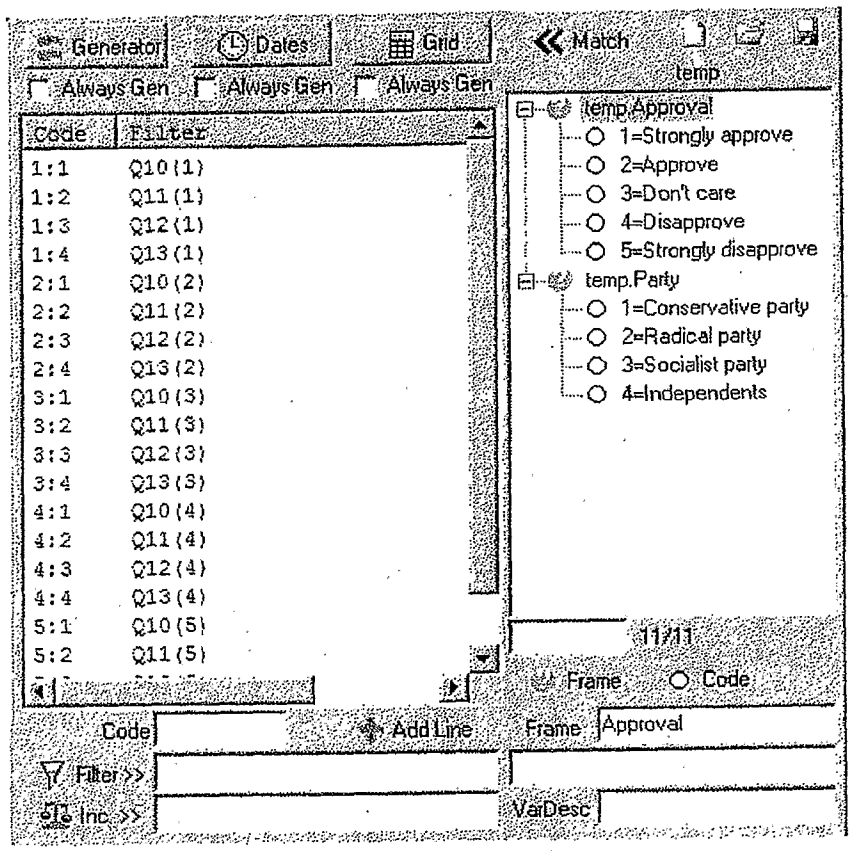


**Figure 16**


Generator
Dates
Grid
Match
temp
Always Gen
Always Gen
Always Gen
Code
Filter
1:1 Q10(1)
1:2 Q11(1)
1:3 Q12(1)
1:4 Q13(1)
2:1 Q10(2)
2:2 Q11(2)
2:3 Q12(2)
2:4 Q13(2)
3:1 Q10(3)
3:2 Q11(3)
3:3 Q12(3)
3:4 Q13(3)
4:1 Q10(4)
4:2 Q11(4)
4:3 Q12(4)
4:4 Q13(4)
5:1 Q10(5)
5:2 Q11(5)
temp.Approval
1=Strongly approve
2=Approve
3=Don't care
4=Disapprove
5=Strongly disapprove
temp.Party
1=Conservative party
2=Radical party
3=Socialist party
4=Independents
11/11
Frame
Code
Code
Add Line
Frame
Approval
Filter>>
Inc.>>
VarDesc

## Figure 17

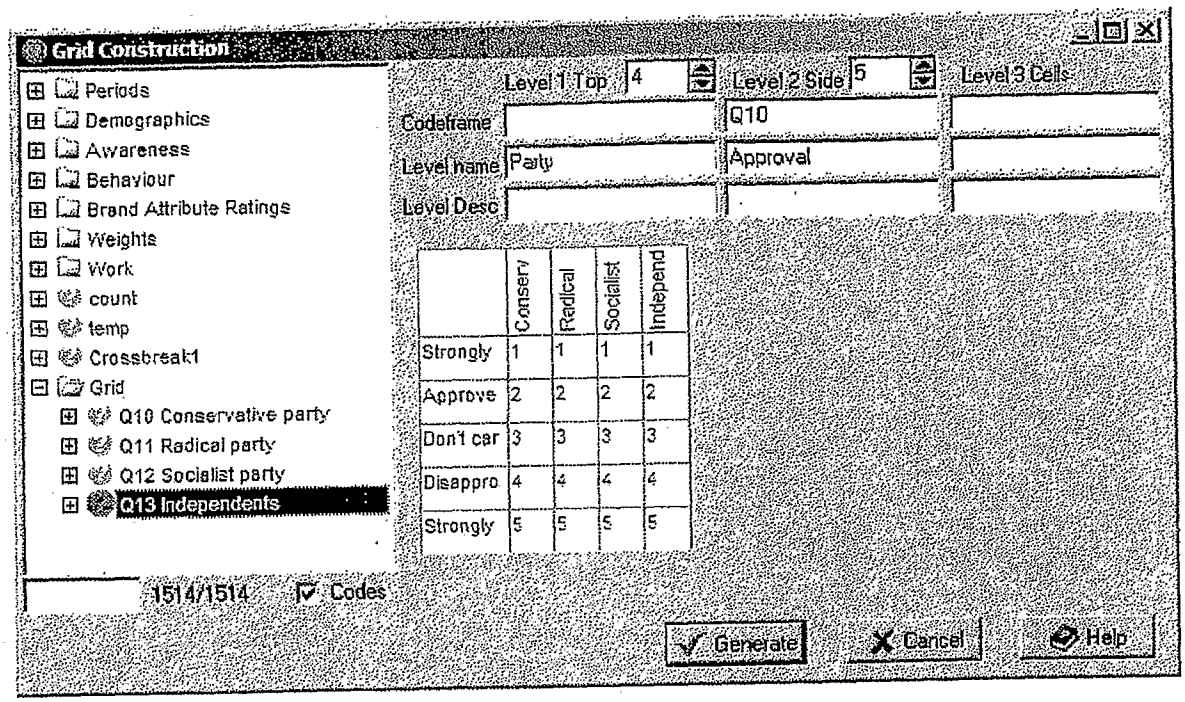
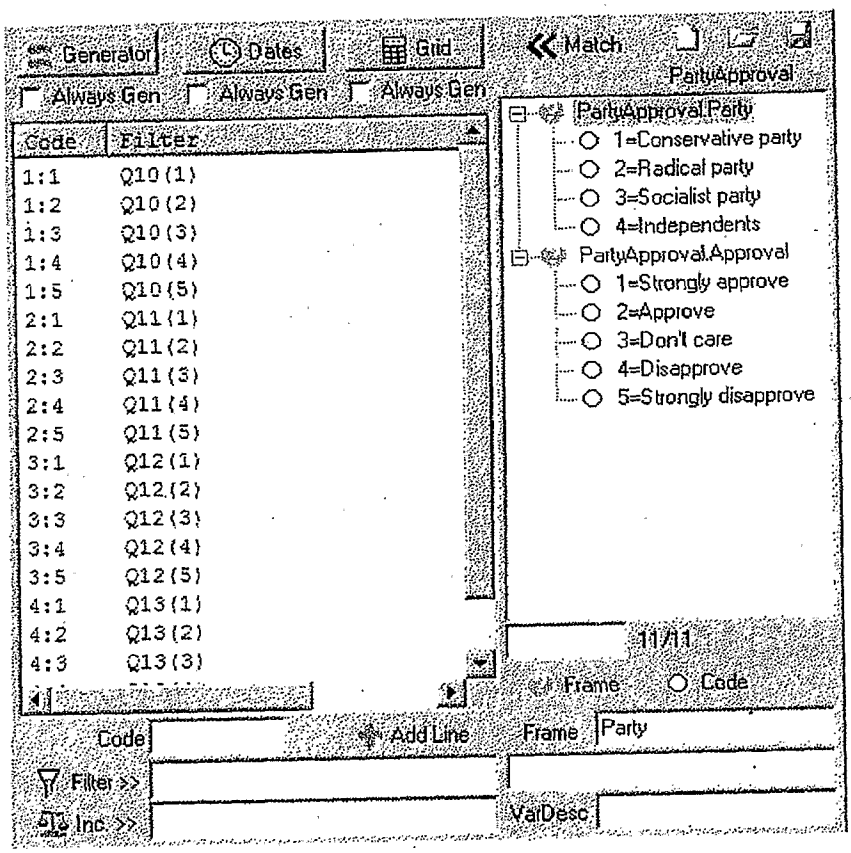
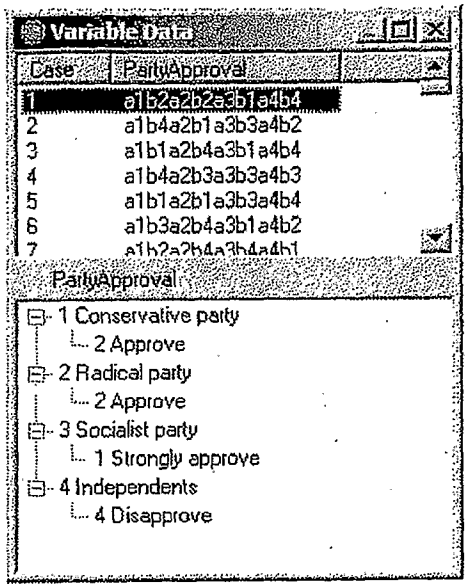

Grid Construction
Periods
Demographics
Awareness
Behaviour
Brand Attribute Ratings
Weights
Work
count
temp
Crossbreak1
Grid
Q10 Conservative party
Q11 Radical party
Q12 Socialist party
Q13 Independents
1514/1514
Codes
Level 1 Top 4
Level 2 Side 5
Level 3 Cells
Codeframe
Q10
Level name
Party
Approval
Level Desc
Consery
Radical
Socialist
Independ
Strongly 1 1 1 1
Approve 2 2 2 2
Don't car 3 3 3 3
Disappro 4 4 4 4
Strongly 5 5 5 5
Generate
Cancel
Help

**Figure 18**


Generator
Dates
Grid
Match
Always Gen
Always Gen
Always Gen
PartyApproval
Code
Filter
1:1 Q10(1)
1:2 Q10(2)
1:3 Q10(3)
1:4 Q10(4)
1:5 Q10(5)
2:1 Q11(1)
2:2 Q11(2)
2:3 Q11(3)
2:4 Q11(4)
2:5 Q11(5)
3:1 Q12(1)
3:2 Q12(2)
3:3 Q12(3)
3:4 Q12(4)
3:5 Q12(5)
4:1 Q13(1)
4:2 Q13(2)
4:3 Q13(3)
PartyApproval.Party
1=Conservative party
2=Radical party
3=Socialist party
4=Independents
PartyApproval.Approval
1=Strongly approve
2=Approve
3=Don't care
4=Disapprove
5=Strongly disapprove
11/11
Frame
Code
Code
Add Line
Frame
Party
Filter >>
Inc >>
ValDesc

## Figure 19




## Figure 20

| | Value for money | Lasts a long time | Attractive |
|---|---|---|---|
| Brand1 | Q41 | Q42 | Q43 |
| Brand2 | Q44 | Q45 | Q46 |

## Figure 21

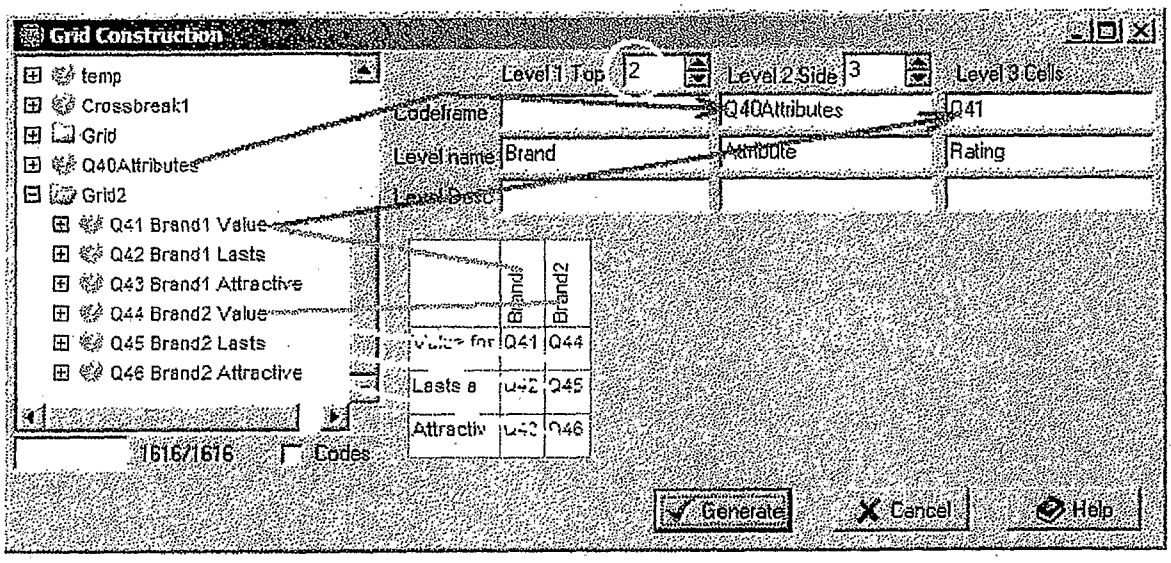

Grid Construction
temp
Crossbreak1
Grid
Q40Attributes
Grid2
Q41 Brand1 Value
Q42 Brand1 Lasts
Q43 Brand1 Attractive
Q44 Brand2 Value
Q45 Brand2 Lasts
Q46 Brand2 Attractive
1616/1616
Codes
Level 1 Top
2
Level 2 Side
3
Level 3 Cells
Codeframe
Q40Attributes
Q41
Level name
Brand
Attribute
Rating
Level Desc
Brand1
Brand2
Q41
Q44
Q42
Q45
Q43
Q46
Generate
Cancel
Help

**Figure 22**

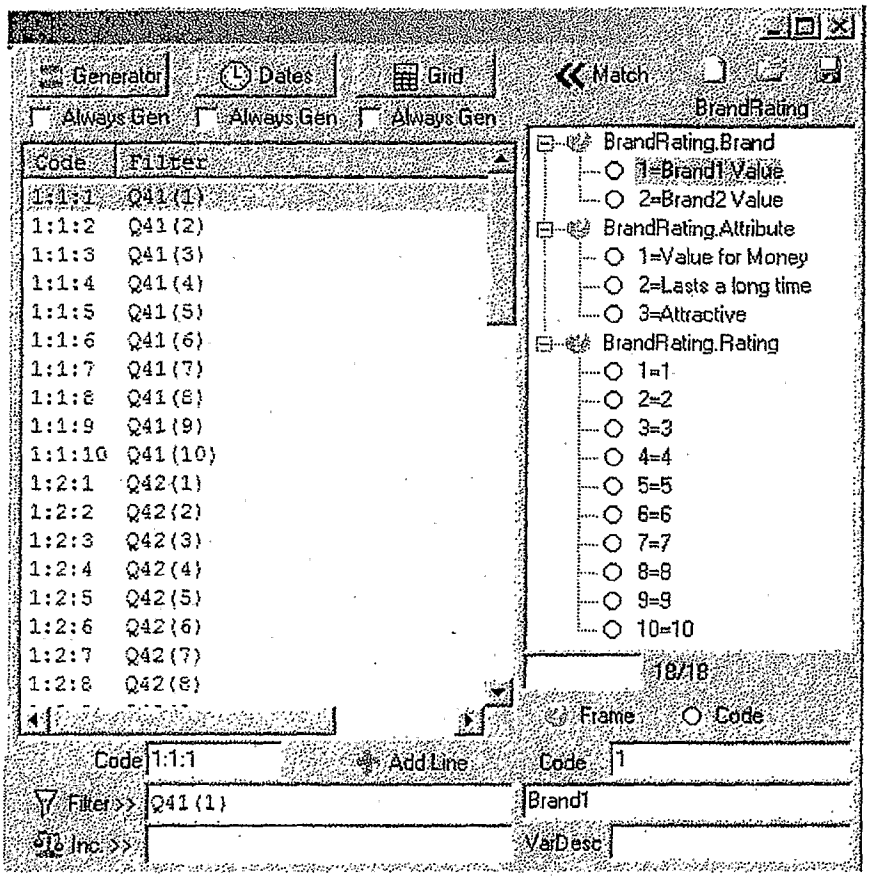
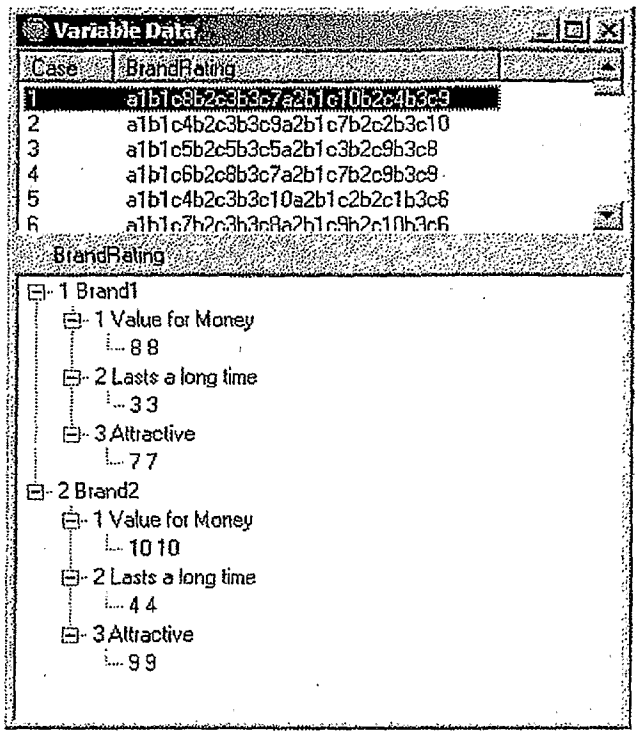

Generator
Dates
Grid
Match
Always Gen
Always Gen
Always Gen
BrandRating
Code
Filter
1:1:1 Q41(1)
1:1:2 Q41(2)
1:1:3 Q41(3)
1:1:4 Q41(4)
1:1:5 Q41(5)
1:1:6 Q41(6)
1:1:7 Q41(7)
1:1:8 Q41(8)
1:1:9 Q41(9)
1:1:10 Q41(10)
1:2:1 Q42(1)
1:2:2 Q42(2)
1:2:3 Q42(3)
1:2:4 Q42(4)
1:2:5 Q42(5)
1:2:6 Q42(6)
1:2:7 Q42(7)
1:2:8 Q42(8)
BrandRating.Brand
1=Brand1 Value
2=Brand2 Value
BrandRating.Attribute
1=Value for Money
2=Lasts a long time
3=Attractive
BrandRating.Rating
1=1
2=2
3=3
4=4
5=5
6=6
7=7
8=8
9=9
10=10
18/18
Frame
Code
Code 1:1:1
Add Line
Code 1
Filter>> Q41(1)
Brand1
Inc. >>
VarDesc

Figure 23


Variable Data
Case
BrandRating
1 a1b1c8b2c3b3c7a2b1c10b2c4b3c9
2 a1b1c4b2c3b3c9a2b1c7b2c2b3c10
3 a1b1c5b2c5b3c5a2b1c3b2c9b3c8
4 a1b1c6b2c8b3c7a2b1c7b2c9b3c9
5 a1b1c4b2c3b3c10a2b1c2b2c1b3c6
BrandRating
1 Brand1
1 Value for Money
8 8
2 Lasts a long time
3 3
3 Attractive
7 7
2 Brand2
1 Value for Money
10 10
2 Lasts a long time
4 4
3 Attractive
9 9

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2006/001435

A. CLASSIFICATION OF SUBJECT MATTER

Int. Cl.

***G06F 7/00*** (2006.01) ***G06F 17/18*** (2006.01)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
USPTO, DWPI: Keyword include (Data, Hierarchical, indicator, depth, represent) and similar terms

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | DIGGINS, C. 'Trees and Labelled S-Expressions'. 3 November 2005 (retrieved on 10 January 2007) Retrieved from the internet <URL:http://www.artima.com/weblogs/viewpost.jsp?thread=135216> See whole document | 1-7 |
| X | US 7003722 B2 (Rothchiller et. al) 21 February 2006 See whole document | 1-7 |
| A | US 2003/0220914 A1 (De Angelis et. al) 27 November 2003 See whole document | 1-16, 25-26, 37-38 |

☐ Further documents are listed in the continuation of Box C ☒ See patent family annex

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 10 January 2007 | 17 JAN 2007 |
| Name and mailing address of the ISA/AU<br>AUSTRALIAN PATENT OFFICE<br>PO BOX 200, WODEN ACT 2606, AUSTRALIA<br>E-mail address: pct@ipaustralia.gov.au<br>Facsimile No. (02) 6285 3929 | Authorized officer<br>M.G. KRAEFFT<br>Telephone No : (02) 6283 2455 |

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2006/001435

**Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)**

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a)

**Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)**

This International Searching Authority found multiple inventions in this international application, as follows:

See additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:
1-16, 25-26, 37-38

**Remark on Protest**

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2006/001435

**Supplemental Box**
(To be used when the space in any of Boxes I to VIII is not sufficient)

**Continuation of Box No: III**

This International Application does not comply with the requirements of unity of invention because it does not relate to one invention or to a group of inventions so linked as to form a single general inventive concept.

In assessing whether there is more than one invention claimed, I have given consideration to those features which can be considered to potentially distinguish the claimed combination of features from the prior art. Where different claims have different distinguishing features they define different inventions.

This International Searching Authority has found that there are different inventions as follows:

- Claims 1-16, 25-26, 37-38 are directed to a hierarchical data format, analytical tool, apparatus and methods of representing and processing the data. It is considered that the data format with codes indicating hierarchical levels comprises a first distinguishing feature,

- Claims 17-18, 24, are directed to a specification representation language (SRL) and a method of using the SRL to convert data from one format to another. It is considered that the specification representation language comprises a second distinguishing feature,

- Claims 27-29 are directed to a method of determining a row or column applicable to a response having a complex data structure. It is considered that the method of determining a row or column applicable to a response having a complex data structure comprises a third distinguishing feature,

- Claim 36 is directed to a method of arranging variables in a grid. It is considered that method of arranging variables in a grid comprises a fourth distinguishing feature.

NOTE: Claims 37 and 38 could be linked to any of the inventions.

PCT Rule 13.2, first sentence, states that unity of invention is only fulfilled when there is a technical relationship among the claimed inventions involving one or more of the same or corresponding special technical features. PCT Rule 13.2, second sentence, defines a special technical feature as a feature which makes a contribution over the prior art.

Each of the abovementioned groups of claims has a different distinguishing feature and they do not share any feature which could satisfy the requirement for being a special technical feature. Because there is no common special technical feature it follows that there is no technical relationship between the identified inventions. Therefore the claims do not satisfy the requirement of unity of invention *a priori*.

## INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
**PCT/AU2006/001435**

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

| Patent Document Cited in Search Report | | Patent Family Member | | | | | |
|---|---|---|---|---|---|---|---|
| US | 7003722 | AU | 2004200627 | BR | 0400362 | CA | 2458499 |
| | | CN | 1558348 | EP | 1452974 | JP | 2004265405 |
| | | KR | 2004007757 | MX | PA04001934 | PL | 365651 |
| | | RU | 2004105879 | US | 2004172590 | US | 2006117251 |
| | | ZA | 200401487 | | | | |
| US | 2003220914 | | | | | | |

Due to data integration issues this family listing may not include 10 digit Australian applications filed since May 2001.

END OF ANNEX