From Analysis to Interactive Exploration: Building Visual Hierarchies from OLAP Cubes

Svetlana Vinnik and Florian Mansmann

University of Konstanz, P.O.Box D188, 78457 Konstanz, Germany
{vinnik|mansmann}@inf.uni-konstanz.de

Abstract. We present a novel framework for comprehensive exploration of OLAP data by means of user-defined dynamic hierarchical visualizations. The multidimensional data model behind the OLAP architecture is particularly suitable for sophisticated analysis of large data volumes. However, the ultimate benefit of applying OLAP technology depends on the "intelligence" and usability of visual tools available to end-users.
The explorative framework of our proposed interface consists of the navigation structure, a selection of hierarchical visualization techniques, and a set of interaction features. The navigation interface allows users to pursue arbitrary disaggregation paths within single data cubes and, more importantly, across multiple cubes. In the course of interaction, the navigation view adapts itself to display the chosen path and the options valid in the current context. Special effort has been invested in handling non-trivial relationships (e.g., mixed granularity) within hierarchical dimensions in a way transparent to the user.
We propose a visual structure called *Enhanced Decomposition Tree* to use along with popular "state-of-the-art" hierarchical visualization techniques. Each level of the tree is produced by a disaggregation step, whereas the nodes display the specified subset of measures, either as plain numbers or as an embedded chart. The proposed technique enables a stepwise descent towards the desired level of detail while preserving the history of the interaction. Aesthetic hierarchical layout of the node-link tree ensures clear structural separation between the analyzed values embedded in the nodes and their dimensional characteristics which label the links. Our framework provides an intuitive and powerful interface for exploring complex multidimensional data sets.

1 Introduction

With rapid evolvement of *data warehouse* technology in the last decade huge volumes of data have become available for analysis and exploration. Data warehouses integrate data from heterogeneous sources into a single repository for comprehensive analytical processing. Apart from generating standard reports, the users are able to gain deeper insights into the data by means of dynamically formulating and verifying their hypotheses about it. Arranging the data into a multidimensional space is especially beneficial for decision support due to the potential of retrieving the data subsets of interest in the form exactly satisfying the users' information needs. Furthermore, multiple coordinated views of the same data set help to dynamically explore it and uncover the "hidden gems", such as outliers, peculiar patterns, trends or clusters.

Data warehouses increasingly adopt the *multidimensional data model* which was designed to meet the challenges of the online analytical processing (OLAP) [5] by providing efficient execution of queries that aggregate over large amounts of detailed data [14]. This model uses numerical *measures* as its analytical objects, with each measure uniquely determined by its *dimensions* and therefore treated as a point in a multidimensional space [3]. Depending on the expected type of queries the data can be organized into *hypercubes* with a measure (or multiple measures) as the value under analysis stored in the cube's cells, the measure's determining dimensions as the cube's axes and the dimensions' values as the coordinates of respective measure cells.

Desired data view can be retrieved from the cubes by applying OLAP operations, such as *slice-and-dice* to reduce the cube, *drill-down* and *roll-up* to perform aggregation and disaggregation, respectively, along a hierarchical dimension, *drill-across* to combine multiple cubes, *ranking* to find the outlier values, and *rotating* to see the data grouped by other dimensions [14].

The standard interface for exploring OLAP data is a *Pivot Table*, or *Cross Tab* [10], which is a 2-dimensional spreadsheet with associated totals and subtotals. Pivot Table allows to nest multiple dimensions within the same axis. This technique is adequate for displaying the query results in a straightforward fashion but it fails to show the selected values in a larger context and is thus a rather poor option for complex data exploration. Advanced OLAP tools overcome the limits of the cross tab interface by offering a multitude of powerful visual alternatives for retrieving, displaying, and interactively exploring the data. Continuous efforts are put into providing new approaches to visual exploration of the hypercube data, such as hierarchical visualizations (decomposition trees, chart trees, tree-maps etc.), multiscale views, interactive scatter-plots, etc. described in the next section.

2 Related Work

The work related to ours in one way or another can be sub-divided into three major groups, namely, multidimensional data modeling, visualization techniques, and explorative interfaces.

2.1 Multidimensional Modeling and Data Warehouse Design

Modeling challenges arise whenever dimensional hierarchies contain irregularities preventing their straightforward mapping to balanced dimensional trees as required for OLAP operations. A proposal to transforming such data into summarizable structures, transparently to the user, can be found in [15]. Implications of unbalanced hierarchies on the logical data warehouse design are explained in [12]. Most of the data warehouse research, however, is concerned with performance issues and is orthogonal to the scope of this paper.

2.2 Visualizing OLAP Data

Besides the classical visualization techniques, such as Pivot Tables [10] and 2-dimensional plots and charts, familiar to any OLAP analyst, a wide variety of more comprehensive visual frameworks for incremental exploration and navigation in large multidimensional data volumes have emerged. Hierarchy-aware visualization techniques applicable in the OLAP context can be grouped into the following categories:

– *Geometric* (Scatterplots, Landscapes, Hyperslice, Parallel Coordinates)
– *Icon-based* (Chernoff Faces, Stick Figures, Color Icons, TileBars)
– *Pixel-oriented* (Recursive Pattern, Circle Segments)
– *Hierarchical* (Dimensional Stacking, Worlds-within-Worlds, Treemap, Cone Trees, InfoCube)
– *Graph-Based* (Straight-, Poly- and Curved-Line, DAG, Symmetric, Cluster)
– *Hybrid* techniques which arbitrarily combine any of the above.

Applicability of any particular technique or their combination depends largely on the analysis needs and the level of user expertise. An overview of the above techniques with respect to OLAP data can be found in [13].

Conventional node-link trees in a classical aesthetical view [17] and in a variety of more compact layouts (*hyperbolic*, *balloon*, *radial*, etc. presented in [8]) which are rather familiar and intuitive to interpret can be used to increase the user's awareness of the hierarchical relationships within the data or allow users to define their own hierarchies. More comprehensive and specialized techniques are appreciated for complex analysis, scientific visualization and data mining. [18] presents some advances in hierarchy visualization and its use for exploring user-defined hierarchies. A well structured classification of the "state-of-the-art" visualization and interaction techniques with respect to the type and the dimensionality of the data is produced in [9].

2.3 Exploration Tools

There is an abundance of tools and interfaces for exploring multidimensional data. We limit ourselves to naming a few products which offer distinguished features relevant for our work. One developed system called Polaris [20] extends the Pivot Table interface by offering a combination of a variety of displays and tools for visual specification of analysis tasks. Polaris is a predecessor of a recently released business intelligence product called Tableau Software [2]. ProClarity was the first to enhance business intelligence with *Decomposition Trees* [16] for visual node-by-node disaggregation of data cubes. XMLA enriches the idea of hierarchical disaggregation by arranging the decomposed subtotals of each parent value into a nested chart (Bar- and Pie-Chart Trees) in its Report Portal OLAP client [21]. Visual Insights has developed a family of tools, called ADVIZOR, with an intuitive framework for parallel exploration of multiple measures [7].

Our interface differs from the standard OLAP tools in the way data navigation is built (attribute hierarchies with data on-demand) and the way the data is presented (hierarchical visualizations instead of spreadsheets and charts).

3 Handling Complex Multidimensional Data

OLAP architecture performs well on the facts that are summarizable along each dimension, i.e. where all dimensions are balanced hierarchies, however, it fails to adequately support irregular dimension hierarchies [14]. Our ambition is to tackle some of the frequently observed irregularity patterns in complex dimensions.

Throughout the remainder of the paper we will refer to the following fragment of a (simplified) university data warehouse consisting of two OLAP cubes:

1. **Orders** with the facts about the university's expenditures.
   Measure: *total amount in €*. Dimensions: *Interval*, *Category*, *Institution*, *Project*, and *Funds*.
2. **Students** with the facts about the number of enrolled students.
   Measures: *number of cases* and *number of heads*[1]. Dimensions: *Semester*, *Term*, *Nationality*, *TeachingUnit*, *Degree*, *Gender*, *Eligibility*.

The logical design of the above database roughly corresponds to the *snowflake schema* [3] which explicitly decomposes hierarchical dimensions into per-level subdimensional tables. Fact tables contain measure attribute(s) and their dimensional characteristics. The latter are the foreign keys referencing the respective dimension table. In case of a hierarchical dimension, each subdimensional table is connected to the next level table(s) by means of foreign keys. Thereby, a snowflake shape is produced by the fact table in the center, surrounded by the directly referenced bottom-level dimensions with all their referenced upper levels at periphery. The data schema described above is depicted in Fig. 1.

We have deliberately chosen a rather complex data warehouse fragment in order to examine various patterns in hierarchical dimensions as well as the ability of our navigational framework to handle them in a way intuitive for the user.

3.1 Classification of Dimensional Hierarchies

The OLAP cube has a relation schema $D_1 \cup D_2 \cup ...D_n$, where each D_i is a dimensional attribute with its corresponding relation d_i referred to as a *dimension.* Hierarchical dimensions consist of subdimensions, or nodes, for each of its levels. The tuples in a relation are the members, or *entities*, of the respective dimension, a tuple of d_i is denoted $t[d_i]$.

We extend the notion of dimension to include *abstract* nodes, i.e. without associated relations and entities. Abstract nodes are used simply as an upper class for uniting multiple child categories or as a root node at the top of the entire underlying hierarchy: D_i is abstract if its $d_i = \emptyset$. The next-level subdimension D_k of any D_i is called its *child*, $D_k = child(D_i)$, and the set of all node's children is given by the function $children(D_i)$. The cardinality of node D_i equals the number of its children: $|D_i| = |children(D_i)|$.

[1] Head statistics counts physical persons, assigning each "head" to the supervising faculty of his/her major. Case statistics splits single enrollment into separate cases, one for each major/minor, to register a student as a "case" at each involved faculty.

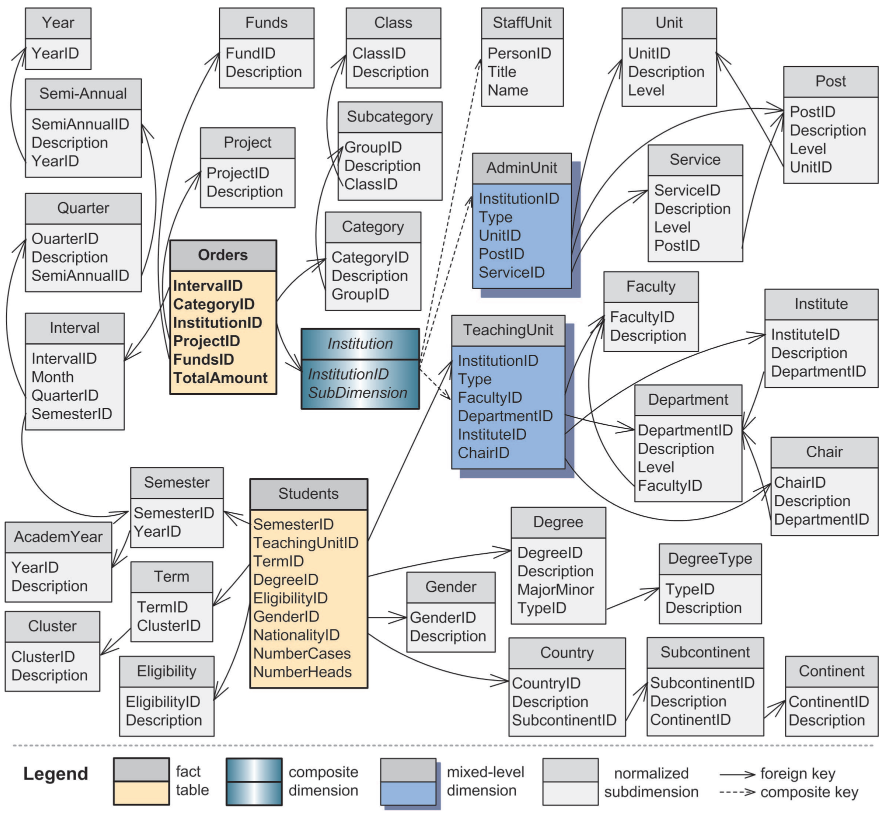

Fig. 1: Logical schema of the university data warehouse fragment

The hierarchical behavior of a dimensional node can be described based on 1) its relation (or members), 2) cardinality, and 3) its relationship w.r.t. its children. Notice, that the type characterizes solely the node itself, and not its entire subtree (the descendants may be of various types). Based on the logical schema in Fig. 1, one can identify at least the following five distinct behaviors:

- **Simple**: a non-hierarchical dimension (e.g., *Gender*, *Project*); D_i is *simple* $\rightarrow |D_i| = 0 \wedge d_i \neq \emptyset$.
- **Single Hierarchy**: a strict hierarchy (e.g., *Interval*, *Category*) has just a single decomposition path; D_i is *single* $\rightarrow |D_i| = 1 \wedge d_i \neq \emptyset$.
- **Multiple Hierarchy**: a dimension is subdivided in multiple ways. For instance, *Intervals* can be aggregated along semester $\rightarrow$ academic year, or along quarter $\rightarrow$ semi-annual $\rightarrow$ calendar year. Multiple paths are placed into the same abstract parent node; D_i is *multiple* $\rightarrow |D_i| > 1 \wedge d_i = \emptyset$.
- **Composite Hierarchy**: an "umbrella" dimension uniting heterogeneous members from multiple relations in a single superclass (e.g., the members of *Institution* may refer to *StaffUnit*, *AdminUnit*, or *TeachingUnit*):

D_i is *composite* $\rightarrow |D_i| > 1 \wedge d_i \neq \emptyset \wedge \forall\ D_k \in children(D_i) : d_k \subset d_i$. Since *InstituionID* in *Orders* may point to the entry from any of the three tables, a composite *Institution* dimension is built by extracting the primary keys of original dimensional tables, along with the table's name, into a new table.
- **Mixed-Level Hierarchy**: the entities from upper hierarchy levels do not merely serve for aggregating (as in single hierarchy), but also participate as end-entities in the fact table. Therefore, an additional relation is built on the top of the respective hierarchy by denormalizing the latter into a single table (as in *AdminUnit* or *TeachingUnit*). To separate its twofold role, the dimension's node has to contain its own level's relation as a *simple* child subdimension (see section 5 for further details);
  D_i is *mixed-level* $\rightarrow |D_i| \geq 1 \wedge d_i \neq \emptyset \wedge \exists D_k \in children(D_i) : d_k = d_i$

The two cubes do not have any directly shared dimensions within their schemata, and, therefore, cannot be drilled-across for parallel exploration by means of a natural join. However, a closer inspection reveals two linking options:

- *Semester* in *Students* and *Intervals* in *Orders* are summarizable by semester,
- *TeachingUnit* in *Students* is a subclass of *Institution* in *Orders*.

These linkages encourage the anticipation that both fact tables can be joined for cross-cube exploration at their shared aggregation levels.

4 OLAP Cube as a Decomposition Tree

OLAP operations, such as *drill-down*, *roll-up*, and *cube*, transform the data from a fact table into a hierarchy by aggregating or disaggregating the measure along specified dimensions. A series of successive disaggregation steps can be presented as a *Decomposition Tree*. Notice that decomposition is a process contrary to aggregation. The measure's total, aggregated along all selected attributes, forms the root node of the tree. The next level emerges by computing the subtotals of a disaggregation along any specified dimension. Each subsequent k-th level will contain the subtotals disaggregated by k specified dimensions[2]. Back to our example, the measures of cube *Students* may be decomposed along the following sequence of dimensional attributes (see Fig. 2):

$$AcademYear \rightarrow Semester \rightarrow Gender \rightarrow Degree \rightarrow ...$$

Unlike standard spreadsheet views, the hierarchical presentation in Fig. 2 by its very nature has an advantage of supporting arbitrary number of split dimensions in arbitrary order while preserving this order in its levels. All nodes at the same level correspond to the same granularity whereas nested charts accelerate identification of interesting values and directions for further expansion. Interactive filtering can be applied to eliminate or temporarily hide irrelevant subtrees.

[2] In terms of a SQL statement, decomposition adds the chosen dimension's attribute to the `GROUP BY` clause

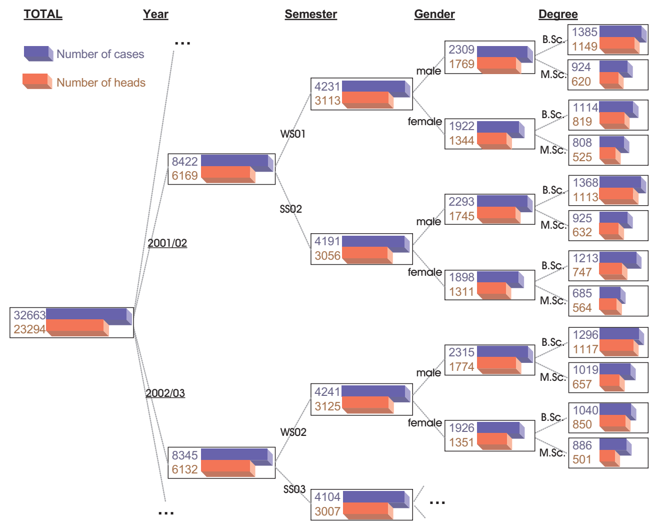

Fig. 2: A user-driven hierarchical decomposition of cube *Students*

Further perceptional improvement is achieved by clearly separating the structural information from the actual data: the split dimensions are used as titles for their respective tree levels, the dimension entities label the edges and the measure values are displayed within the nodes. The contents of the nodes can be heterogeneous, such as text, numbers, charts or a combination thereof.

Since there are as many disaggregation operations possible within a cube as the total number of its dimensions including all subdimensions, and since the order of splitting can be arbitrary, OLAP cubes offer a huge exploration potential ($n!$ disaggregation paths in case of n dimensions) by means of hierarchical decompositions. However, it is rather challenging to incorporate the required framework for interactive construction of user-defined hierarchical visualizations into OLAP interfaces in a fast, intuitive and user-friendly way. In the remaining sections we describe our proposed solution to empowering an OLAP tool with the above exploration technique.

5 Designing the Navigational Framework

Probably the most popular paradigm underlying the OLAP navigation structure is that of a file browser, with each cube as a folder containing the list of top-level dimensions and the list of available measures, as found in Cognos PowerPlay [6], BusinessObjects [1], CNS DataWarehouse Explorer [4], and many other commercial OLAP tools. Each hierarchical dimension is itself a folder containing its child entities. Hierarchical entities can be recursively expanded to show the subtrees of their descendants. The entities of the highest granularity (i.e. the leaf nodes) are represented as files and are non-expandable.

Standard OLAP interfaces allow users to navigate directly in the dimensional data rather than in a dimensional hierarchy. Our approach, however, pursues a clear distinction between the dimension's structure and its instances. Therefore, expansion of a dimension folder reveals solely the nested folders of its subdimensions, contrary to the standard OLAP navigation displaying the child-level data. The instances of any subdimension can be retrieved on-demand. Fig. 3 a and 3 b demonstrates the differences between the standard "show-data" and our proposed "show-structure" interfaces, respectively, at the example of a hierarchical dimension *Nationality*. Notice that expanding the top-level dimension *Nationality* in Fig. 3 b reveals its entire descendant hierarchy, thus enabling the user to "jump over" right to the desired granularity level. The data view is available on explicit demand by clicking the preview button of the respective category. Fig. 3 c shows the activated preview of *Subcontinents* with the option to drill-down into any subcontinent's descendant subtree. The advantages of our proposed navigation structure for building hierarchies can be summarized as follows:

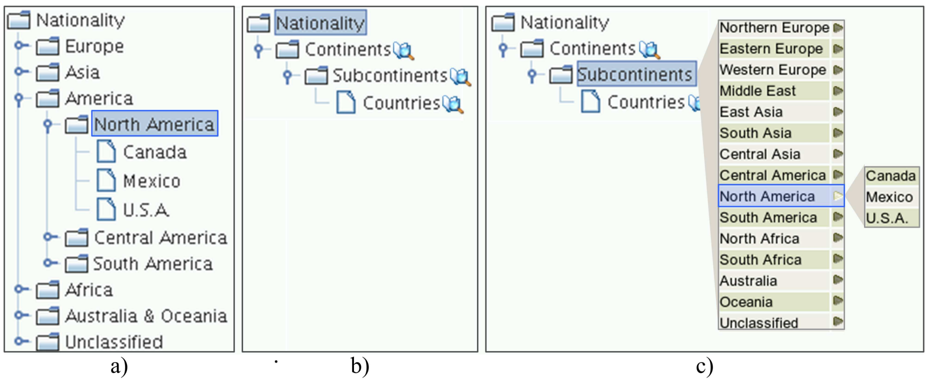

Fig. 3: Navigating in the hierarchical dimension *Nationality*
a) "show-data"-approach b) "show-structure"-approach
c) on-demand preview in the "show-structure"-approach

– clear distinction between the dimension's structure and its contents
– immediate overview of all granularity levels in a hierarchical dimension
– the ability to drill-through directly to any descendant subdimension
– on-demand preview of the data as well as any data node's descendant entities
– compactness on the display due to moderate expansion at most steps
– the entire navigation is built from a single meta table of the kind

title	**table**	**parent**	**root**	**hierarchy**
Nationality	`NULL`	`NULL`	`NULL`	*single*
Subcontinents	*dim_subcontinent*	*Continents*	*Nationality*	*single*
Countries	*dim_country*	*Subcontinents*	*Nationality*	*simple*
...	...	...	...	...

containing, for each dimension entry, its title and table, references to its parent (NULL for top-level) and root[3] dimensions, and its hierarchy type (as classified in section 3)
– the actual data is retrieved only if explicitly requested
– it is easier to find the entries of interest even somewhere deep in the hierarchy without knowing the data (e.g. any country can be accessed directly through the preview of *Countries* without searching for and drilling through its ancestors in *Continents* and *Subcontinents*).

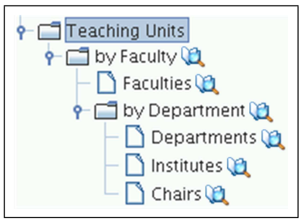

Fig. 4: A mixed-level hierarchy navigation node

As for various hierarchy types defined in section 3, our approach can handle each of them accordingly, using solely the above meta table[4], as pseudo-coded in Algorithm 1. The basic rule is to discontinue recursive expansion whenever mutually exclusive child paths arise since at those points the user is called upon to stick to just one of them. The case of a mixed-level hierarchy deserves a closer inspection. To reflect the twofold role of its subdimensions (i.e. both as leaf nodes and as aggregation levels), each of such subdimension contains, apart from its child level, its own self as a simple, i.e. non-hierarchical, subdimension, as can be seen in Fig. 4 at the example of expanding *TeachingUnit* in the cube *Orders*. Decomposing along *by Faculty* computes the subtotals for each faculty including all its subordinate institutions, whereas choosing its child *Faculties* computes the subtotal only for the faculties themselves as end-entities.

5.1 Parallel Exploration of Multiple Cubes

OLAP Join, or drill-across, allows linking multiple OLAP cubes to compare their measures or derive new ones under the condition that the cubes share at least one dimension. We define a dimension to be *partially shared* if the cubes impose different hierarchies upon it which share at least one aggregation level. Apparently, the cubes can also be joined on partially shared dimensions, as long as each cube is pre-aggregated to the shared level. Let us extend the proposed navigation framework to support parallel exploration of multiple cubes for each shared subdimension. For any number of cubes, pre-selected for a drill-across, the navigation structure can be built in the following steps:

1. Unnest the *top-level dimensions* and the *measures* from their respective fact table folders into a common navigational hierarchy.
2. Identify all partially or fully shared dimensions and the actually shared subdimensions therein (this phase is critical since sharing is not always obvious, e.g. implied by foreign key or other constraints).

[3] *root* reference helps to identify top-level nodes and to avoid recursive SQL queries when retrieving descendant dimensions.
[4] Implementation of the data display routines behind the *Preview* buttons involves more complicated algorithms and is not considered at this stage.

Algorithm 1: Expanding a Dimension's Navigation Node

```
input : dimension name $D$, nesting counter $level$, recursion $propagate$
result: the dimension's sub-tree is displayed

procedure expandNode (Node $D$, int $level$)
begin
    $type$ ← SQL: SELECT type FROM meta WHERE title='$\$D$';
    if $type = simple$ then return; // cannot be expanded, so no action
    else
        Array $children$ ← SQL: SELECT title FROM meta WHERE parent='$\$D$';
        switch $type$ do
            case $single\ hierarchy$
                drawNode ($children[0]$, $++level$, TRUE);// expand recursively
                break;
            case $mixed\text{-}level\ hierarchy$
                foreach $child$ in $children$ do
                    drawNode ($child$, $++level$, TRUE);// expand recursively
                break;
            case $composite$
            case $multiple\ hierarchy$
                foreach $child$ in $children$ do
                    drawNode ($child$, $++level$, FALSE);// no recursion
                break;
            case ... // define further cases
end

procedure drawNode (Node $D$, int $level$, boolean $propagate$)
begin
    Array $info$ ← SQL: SELECT type, table FROM meta WHERE title='$\$D$';
    $icon$ ← getIcon ( $info[type]$);
    indent according to $level$, display $icon$ and $D$'s title
    if $info[table]$ is not NULL then
        display preview icon // there is data to preview
    if $propagate$ then
        expandNode ($D$, $level$); // propagate expansion
end
```

3. For each group of partially shared dimensions, create a new upper-level dimension to serve as their parent and place the former ones underneath the new parent as a multiple hierarchy.
4. Single paths within the created multiple hierarchy might need to be adjusted to contain newly enabled additional aggregation opportunities.

The process of merging the *Interval* dimension of *Orders* and the *Semester* dimension of *Students* is shown in Fig. 5, with their shared levels highlighted.

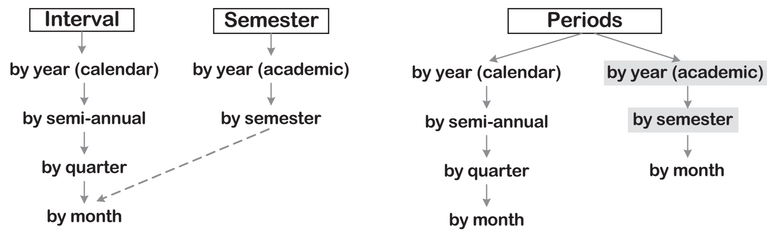

Fig. 5: Unifying partially shared dimensions

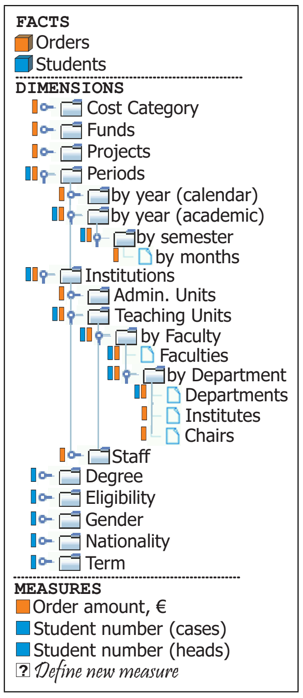

Fig. 6: Navigating in multiple cubes

Visual distinction between shared and non-shared navigation paths can be done by assigning each cube a unique color. The same colors are then used for marking each cube's measures and dimensions. All partially and fully shared top-level dimensions have the color marks of all involved cubes thus giving user a hint about the linking potential. Subdimensions, on the contrary, carry the marks of exclusively those cubes actually sharing that aggregation level. Fig. 6 demonstrates the above idea of using color marks.

6 Interactive Generation of Visual Hierarchies

The purpose of the navigational framework is to enable interactive retrieval of the data to be displayed in the visualization window according to the specified layout (e.g., Pivot Table, Decomposition Tree, etc.). In case of a hierarchical visualization the only supported direction of retrieval is disaggregation: each level of the hierarchy is produced by adding a new dimension or drilling down any already added hierarchical dimension. In what follows we explain the basic steps of generating a tree-like visualization at the example of a bar-chart tree built from the *Orders* cube:

– *Measure selection*: Selecting / de-selecting measures in the navigation panel causes them to be added to / removed from the visualization, whereas the following modalities can be distinguished:
  • *Displaying multiple measures per node*: when more than one measure is dragged, a dialog window will pop up prompting the user to specify the measures' display options (plain numbers, nested charts, or both)
  • *Specifying no measure*: with no measure chosen, one can display the structure of a hierarchical dimension without associated subtotals

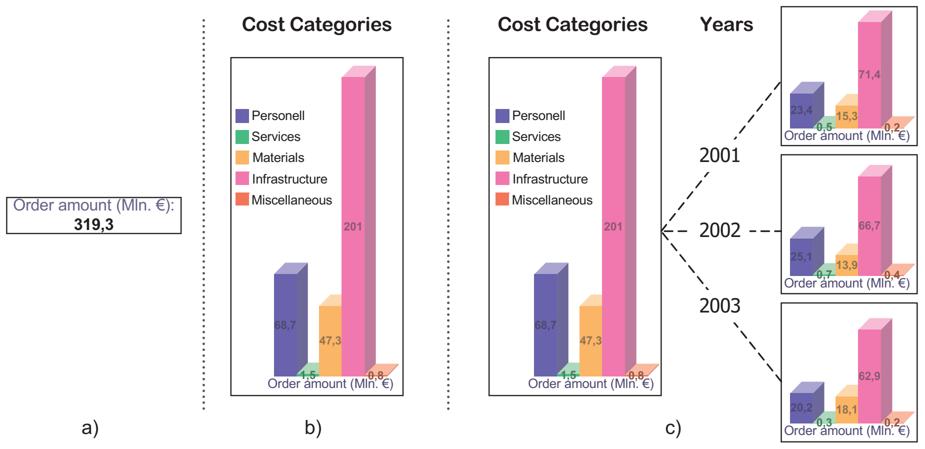

Fig. 7: Generating a bar-chart tree visualization
a) initialization b) creating the root node c) adding a new level

- *Adjusting the measure's format*: via the *Options* menu, the measure's display options, such as rounding, range, units etc., can be specified
- *Defining a new measure*: advanced users can use this option (see bottom of Fig. 6) to define a new measure by combining existing ones through arithmetic operations or functions.

Back to our scenario, dragging the measure *Order Amount, €* into an empty plane displays its total value as a root node, as shown in Fig. 7 a.

– *Decomposition*: Dragging any dimension into the visualization window is interpreted as *disaggregation* along that dimension. The dragged dimension along with all its ancestor hierarchy up to the root are added to the list of *split dimensions* and are made undraggable in order to disable upward steps (roll-up) invalid in this context. Decomposition causes the new level with decomposed subtotals to be added to the visual hierarchy, except in the case of a nested-chart-tree where the following options need to be distinguished:
  - *Initializing*: the first chosen dimension is used for decomposing the root node value into a nested chart, thus defining the chart's granularity within the node, and is denoted Dim_{inner}. Fig. 7 b shows the results of choosing *Cost Category* to be Dim_{inner}.
  - *Outer Decomposition*: Splitting along any dimension which is not a descendant of Dim_{inner} produces a new level with unchanged entities in the nested charts but with the respectively decomposed values in them, as depicted in Fig. 7 c where the root node was split along *Year*.
  - *Inner Decomposition / Drill-down*: Drilling down into a descendant of Dim_{inner} turns the split dimension to be the new Dim_{inner} changing the nested chart's granularity to the new level. The entities of the previous Dim_{inner} serve as the outer split dimension, as shown in Fig. 8 a.

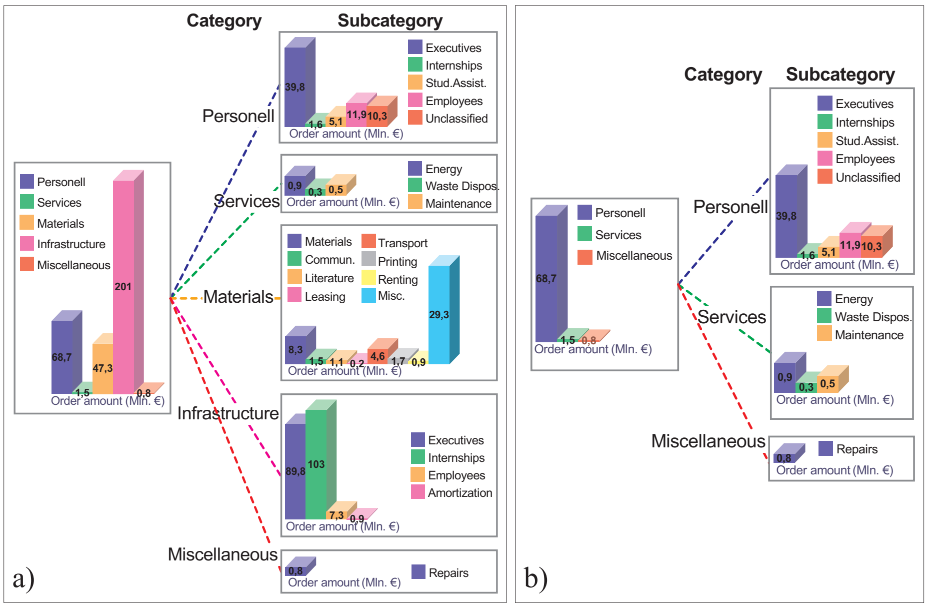

Fig. 8: Interacting with a bar-chart tree
a) Performing an inner decomposition b) Applying global filtering

– *Global Filtering*: Any dimension can be applied as a global filter if dragged into the filter panel. Filtering results in the measure being aggregated only for the explicitly selected entities of the filter dimension. Filtering along an *unsplit* dimension does not reduce the number of nodes, but rather influences the measure values in the nodes. For example, filtering the tree in Fig. 7 c by *Project* would simply recompute the subtotals in the nested charts based on the selected projects. Filtering along an already split dimension will not only recompute the subtotals at each level, but will actually remove the subtrees of deselected entities (or, in case of Dim_{inner} or its ancestor, the entries within the charts) from the visualization. Fig. 8 b shows the effect of eliminating two entities in the filter dimension *Cost Categories*.

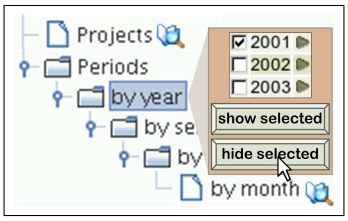

Fig. 9: Inline elimination of the nodes to display

– *Local Filtering*: By default, dragging a dimension into the visualization would create a child node for each of its entities. Alternatively, the user can explicitly specify the subset of entities in the current to-be-split dimension directly in the dimension's preview. Such inline filtering is interpreted as local, i.e. it affects only the current tree level leaving the upper levels unchanged. For example, inline elimination of the entry *2001* when performing a decomposition shown in Fig. 9 would cause that year's node in Fig. 7 c to be withdrawn. Local filtering is equivalent to simply deleting uninteresting nodes from the visualization.

6.1 Interaction-Preserving Navigation

In the process of constructing complex hierarchies the user may lose the orientation as more navigational nodes at various levels become expanded and used as inline or global filters. Manageability of the navigation can be improved by forcing the displayed navigational hierarchy to adapt to the course of interaction. The core idea is to visually separate the expired paths (i.e. those already used as decomposition axes) from the still available ones. This is achieved by partitioning the background behind the list of dimensions vertically into the *expired* (dark background) and the *active* areas. Initially, all dimensions are placed into the active area. Two lists are managed in the course of interaction:

- *ActiveList*: contains the top-level nodes of all unsplit paths
- *ExpiredList*: contains the nodes of all dimensions already split

Each time a decomposition step is performed, the split dimension along with all its ancestors are shifted into the expired area. The entire navigation gets adjusted according to the rules described in Algorithm 2.

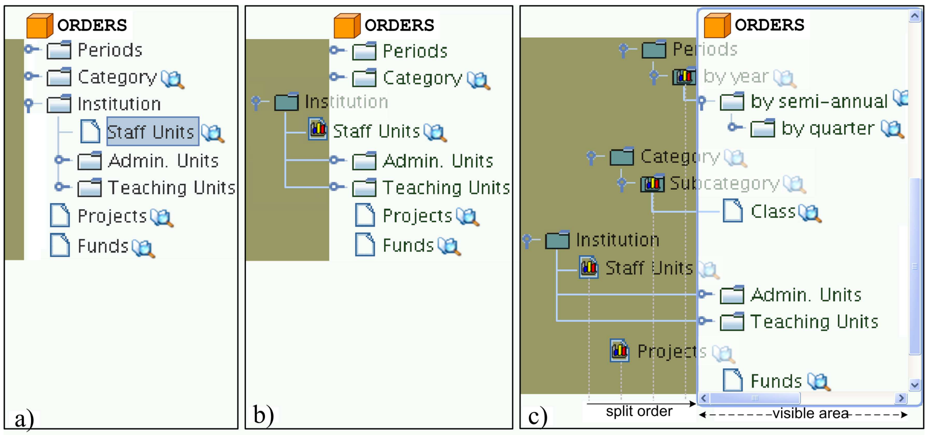

Fig. 10: Adaptive navigation structure
a) initial state b) after the first decomposition c) after multiple decompositions

Fig. 10 demonstrates the presented adaptation procedure at the example of decomposing the *Orders* cube, with the navigation structure prior to the first split operation, its adjustment after performing it, and its state after multiple interactions, as subfigures a, b, and c, respectively. Furthermore, we suggest

Algorithm 2: Adjusting the Navigation after a Decomposition Step

```
input : Split dimension $D$
result: re-arranged display of the navigation hierarchy
procedure shiftDimension (Node $D$)
begin
    $offsets = 1$; // offset = horizontal space between 2 adjacent nodes
    if $D$ in $ActiveList$ then
        $activeRoot \leftarrow D$ // only this node must be re-displayed
    else
        $activeRoot \leftarrow$ find $D$'s ancestor in $ActiveList$;
        $offsets$ += number of $D$'s ancestors up to $activeRoot$; // the number
            of shifts must correspond to the length of the expired path
    foreach $node$ in $ExpiredList$ do
        // shift all previously expired entities
        shift $node$'s segment backwards by 1 offset;
    Redisplay the segment $[ActiveRoot, D]$ moved backwards by $offsets$ shifts;
    Change $D$'s icon to split, its ancestors to expired;
    $ExpiredList \rightarrow$ add($D$);
    $ActiveList \rightarrow$ remove($activeRoot$);
    expandNode ($D$, 0); // replace the expired node with its subtree
    Array $children \leftarrow$ SQL: SELECT title FROM meta WHERE parent='$D';
    foreach $child$ in $children$ do
        $ActiveList \rightarrow$ add($child$);
end
```

that the entire expired area should be hidden from the display by putting the navigation structure into a horizontally scrollable window, as shown in Fig. 10 c. The advantages of the adaptive display can be summarized as follows:

– the expired segments are removed from the active area thus preventing the user from erroneous attempts to access them
– all valid decomposition paths and their still available granularity levels are clearly displayed in the active area
– the split dimensions in the expired area are horizontally ordered to preserve the order of splitting, with more recent steps being closer to the active area
– any expired split step can be undone, causing the corresponding tree level to be removed from the visualization. The navigation structure accounts for the undone split by re-activating the respective path.

7 Enhanced Decomposition Trees

Any particular visualization technique has its pros and cons depending on the type of task to be solved. In case of a dynamic disaggregation of OLAP cubes, the most common tasks are to "drill" into an aggregate in order to trace its behavior along certain dimensional axes and to compare the subtotals within the same granularity level against each other. Standard decomposition tree patented by Proclarity [16] are used to decompose an aggregate along multiple dimension axes. The measure's subtotals as numbers and percentage, as well as the corresponding split dimension's entity are placed inside the nodes. Only one node per interaction can be expandable. Our proposed enhancement of the standard decomposition tree technique is multi-directional and comprises the categories presented below.

Layout. Decomposition trees adopt the classical aesthetic layout due to its visual support of both vertical (parent - children) and horizontal (same level nodes) comparison: children are placed below their parent and each tree level is aligned. Both the top-down and the left-to-right layouts are supported. Directing the nested bar-charts orthogonal to the tree layout (i.e., horizontal bars in case of a top-down tree) puts the charts in each level onto the same axis and is therefore optimal for perceiving the entire level as a single chart (as in Fig. 2). The inherent wastefulness in terms of display area (scarcely populated upper levels consume as much area the bottom ones) can be minimized by adding space awareness to its nodes, as exemplified in Fig. 11. Feasibility of distinct display optimization measures depends on the type and behavior of the value(s) in the nodes. For instance, when decomposing a single measure, the children of each parent can be arranged into "Slice&Sice" treemaps [19], as shown in Fig. 12.

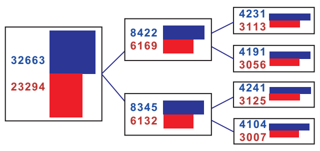

Fig. 11: Using area-aware chart bars

Node contents. The node contents may be heterogeneous, such as a single value or a set of values with their dimensional characteristics. Multiple values per node arise whenever the user has chosen multiple measures to display or a nested-chart technique for a single measure. Our intention in this respect is to migrate from a plain value display towards a value visualization within and across the nodes. Nested bar-charts appear to be a rather suitable way of presenting nested decomposition or comparing multiple measures by putting them onto the same scale (see Fig. 2). Visual enhancements in part of parent-child or child-child relationships are best achieved by applying enclosure mechanisms, such as bounding boxes, subtree area division [11], or recursive partitioning of the node region as in treemaps [19].

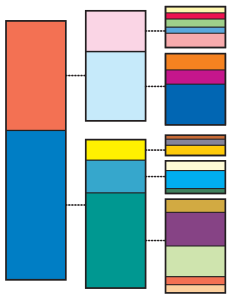
Fig. 12: "Slice&Dice" treemaps as nodes

Visual elements. We suggest that the dimensional characteristics are used for labeling the node's links instead of putting them inside the nodes. This approach contributes to display optimization by reducing the node's inner area and filling the sparsely filled link areas. Another benefit is an improved logical structuring of the data: the aggregate is inside the node whereas its dimensional coordinate is attached to the link, connecting the node to its parent.

Generating the visualization. Unlike with the standard node-by-node expansion approach, our navigational framework empowers fast generation of arbitrarily large trees. A single *drag&drop* interaction is required for generating the entire tree level. The navigational hierarchy adapts itself every time the visualization is re-rendered to hide no longer valid navigation paths from the display and thus leaving the user very little space to lose the orientation or attempt an erroneous operation.

Interaction Features. Interaction serves for exploring the visual hierarchy as well as for its dynamic modification. *Dragging* the nodes is straightforward and is used to deliberately re-arrange the nodes on the dispay. Single nodes can be minimized to icons (*temporary elimination*) by closing them to be reopened later. Deleting marked nodes or regions is equivalent to *local filtering. Zooming* is available in a form of a slider for resizing the entire visualization and as a dynamic zoom cursor for zooming on a single node. Power options, such as *sorting*, changing the display options, re-scaling the inner charts, etc. are accessed via an *"Options"* box placed next to each tree level.

8 Conclusion and Future Work

We have presented a navigation framework for advanced exploration and analysis of multidimensional data in a data warehouse context. The underlying OLAP technology empowers the decision support by allowing users to intuitively retrieve the desired data in a layout and granularity exactly matching the user's needs. We enhanced a standard OLAP interface by enabling user-defined dynamic decompositions of OLAP cubes using hierarchical visualization techniques. Since explorative analysis is driven by the insights acquired in the course of interaction, hierarchical visualization is especially appreciated for its natural preservation of the interaction history and for enabling gradual "descent" from a heavily aggregated overview to the desired level of detail.

The core component of our interface is the introduced navigation structure optimized for fast and easy generation of hierarchical visualizations from OLAP cube data by exploiting the logical data warehouse design. Our framework enables convenient navigation within a single OLAP cube as well as pursuing any valid drill-across paths for joined exploration of multiple facts. The visualization toolkit consists of the popular "state-of-the-art" hierarchical layouts. We also extended the classical node-link tree technique into an OLAP-aware *Enhanced Decomposition Tree*. The displayed data can be clearly structured by placing the aggregates inside the nodes, using their dimensional characteristics for labeling the nodes and the dimension titles for naming the tree levels. The values within the nodes can be arranged into nested charts to facilitate their visual perception.

The directions of our future activities are manifold: 1) to further explore challenging data patterns and new application domains with respect to their adequate mapping in the OLAP model, 2) to examine various visualization techniques as to what extent they qualify or can be adjusted for exploring OLAP data, 3) to refine our implementation to make it more generic and extendable to incorporate new data patterns and visualization techniques, and, 4) to obtain user feedback in order to evaluate and to revise our framework accordingly.

References

1. Business Objects SA, “BusinessObjects OLAP Intelligence,” 2005. [Online]. Available: http://www.businessobjects.com/products/queryanalysis/olapi.asp
2. C. Chabot, P. Hanrahan, C. Stolte, K. Brown, T. Walker, E. Johnson, and J. Mackinlay, “Tableau software,” 2005. [Online]. Available: http://www.tableausoftware.com
3. S. Chaudhuri, U. Dayal, and V. Ganti, “Database technology for decision support systems,” *Computer*, vol. 34, no. 12, pp. 48–55, 2001.
4. CNS International, “DataWarehouse Explorer,” 2005. [Online]. Available: http://www.dwexplorer.com/products/producttour
5. E. F. Codd, S. B. Codd, and C. T. Salley, “Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate,” *Technical report, E.F.Codd & Associates*, 1993.
6. Cognos Software Corporation, “Cognos PowerPlay: Overview–OLAP Software,” 2005. [Online]. Available: http://www.cognos.com/powerplay
7. S. G. Eick, “Visualizing multi-dimensional data,” *SIGGRAPH Comput. Graph.*, vol. 34, no. 1, pp. 61–67, 2000.
8. I. Herman, G. Melançon, and M. S. Marshall, “Graph visualization and navigation in information visualization: A survey,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24–43, 2000.
9. D. A. Keim, “Information visualization and visual data mining,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 1–8, 2002.
10. Microsoft, “Microsoft Excel –User’s Guide,” *Redmond, Wash.*, 1995.
11. Q. V. Nguyen and M. L. Huang, “Space-optimized tree: a connection+enclosure approach for the visualization of large hierarchies,” *Information Visualization*, vol. 2, no. 1, pp. 3–15, 2003.
12. T. Niemi, J. Nummenmaa, and P. Thanisch, “Logical multidimensional database design for ragged and unbalanced aggregation,” in *Design and Management of Data Warehouses*, 2001, p. 7.
13. V. K. Pang-Ning Tan, Michael Steinbach, *Introduction to Data Mining: Concepts and Techniques.* Addison Wesley, 2006.
14. T. B. Pedersen and C. S. Jensen, “Multidimensional database technology,” *IEEE Computer*, vol. 34, no. 12, pp. 40–46, 2001.
15. T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, “Extending practical pre-aggregation in on-line analytical processing,” in *The VLDB Journal*, 1999, pp. 663–674.
16. ProClarity, “Business management software overview,” 2005. [Online]. Available: http://www.proclarity.com/products
17. E. Reingold and J. Tilford, “Tidier drawing of trees,” *IEEE Transactions on Software Engineering*, vol. 7, pp. 223–228, 1981.
18. R. D. B. Richard M. Wilson, “Dynamic hierarchy specification and visualization,” in *Proceedings of the 1999 IEEE Symposium on Information Visualization*, 1999, p. 65.
19. B. Shneiderman, “Tree visualization with tree-maps: 2-d space-filling approach,” *ACM Trans. Graph.*, vol. 11, no. 1, pp. 92–99, 1992.
20. C. Stolte, D. Tang, and P. Hanrahan, “Polaris: A system for query, analysis, and visualization of multidimensional relational databases,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52–65, 2002.
21. XMLA, “Report Portal: Zero-footprint olap web client solution,” 2005. [Online]. Available: http://www.reportportal.com